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 комп'ютерної алгебри та дискретної математики
(повна назва кафедри (предметної, циклової комісії))

Дипломна робота

на здобуття ступеня вищої освіти «бакалавр»
(освітньо-кваліфікаційний рівень)

на тему: Цифровий підпис по ЕльГамалю
(назва українською)

ElGamal Digital signature
(назва англійською)

Виконав: студент денної форми навчання
спеціальності 123 Комп'ютерна інженерія
(шифр і назва напряму підготовки, спеціальності)
Вашневський Євгеній Степанович
(прізвище, ім'я, по-батькові)

Керівник доктор фізико-математичних наук,
професор Варбанець П. Д.
(науковий ступінь, вчене звання, прізвище та ініціали, **підпис**)

Рецензент Бєлозьоров Г.С.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
№___ від «___»_____ 2023 р.

Завідувач кафедри
_____ П.Д.Варбанець
(підпис) (прізвище, ініціали)

Захищено на засіданні ЕК № ___
протокол № __ від «___» ____ 2023 р.
Оцінка _____/_____/_____
(за національною шкалою, шкалою ECTS, бали)
Голова ЕК
_____ А.А.Кобозєва
(підпис) (прізвище, ініціали)

Одеса - 2023

АНОТАЦІЯ

У дипломній роботі розробляється тема «Цифровий підпис по ЕльГамалю».

Мета роботи – ознайомитися з цифровим підписом по ЕльГамалю.

У роботі розглянуто роль цифрового підпису в криптографії і його важливість у забезпеченні безпеки електронних комунікацій, основні аспекти дискретного логарифма, первісні корені , протокол обміну ключами Діффі-Хелмана, шифрування за методом ЕльГамаля та цифровий підпис по ЕльГамалю. Також досліджено проблему дискретного логарифма та методи взлому, що можуть бути застосовані для підриву безпеки методу ЕльГамаля :

1) метод великих і малих кроків;
2) метод Поліга-Сільвера-Хеллмана;
3) р-алгоритм Полларда.

За допомогою інформації, що була отримана у результаті виконання роботи було програмно реалізовано метод Поліга-Сільвера-Хеллмана та описано метод великих і малих кроків, р-алгоритм Полларда.

ABSTRACT

The thesis is devoted to the topic named «ElGamal digital signature».

The target of the work is to familiarize oneself with the digital signature based on ElGamal. The thesis examines the role of the digital signature in cryptography and it’s importance in ensuring the security of electronic communications, the basic aspects of the discrete logarithm, primitive roots, the Diffie-Hellman key exchange protocol, encryption using the ElGamal method, and the digital signature based on ElGamal. The problem of the discrete logarithm and the methods of attack that can be applied to undermine the security of the ElGamal method are also investigated:

- The method of big and small steps;
- The Polig-Silver-Hellman method;
- The Pollard’s rho method.

Using the information obtained the research, The Polig-Silver-Hellman method was implemented programmatically, and the methods of big and small steps and the Pollard’s rho method were described.

ЗМІСТ

ВСТУП

У наш час забезпечення безпеки інформації є важливою задачею, тому цифрові підписи стали важливим компонентом криптографічних систем. Цифрові підписи гарантують цілісність повідомлень, забезпечують їх ідентифікацію та автентифікацію.

Цифровий підпис, що розглядається у даній роботі – цифровий підпис по ЕльГамалю. Цей цифровий підпис є криптографічним алгоритмом. Він використовує протокол обміну ключами Діффі-Хеллмана. Ідея алгоритму полягає в тому, щоб створити приватний та публічний ключ для підпису повідомлень.

Під час виконання дипломної роботи треба вирішити такі питання:

1) Розглянути роль цифрового підпису в криптографії.
2) Ознайомитись з дискретним логарифмом.
   2.2) Первинні корені $mod\ p$;
   2.3) Протокол обміну ключами Діффі-Хеллмана;
   2.4) Шифрування по ЕльГамалю;
   2.5) Проблеми дискретного логарифму та методи злому.
      - метод великих і малих кроків;
      - метод Поліга-Сільвера-Хеллмана;
      - р-алгоритм Полларда.
3) Розібрати цифровий підпис ЕльГамаля.
4) Зробити програмну реалізацію для методу Поліга-Сільвера-Хеллмана.

1 РОЛЬ ЦИФРОВОГО ПІДПИСУ В КРИПТОГРАФІЇ

Криптографія являє собою науку про забезпечення автентичності, конфіденційності та цілісності інформації. Основною метою криптографії є те, щоб захистити дані від несанкціонованого доступу. Одні з принципів криптографії :

- Алгоритми – математичні процедури, функції, що визначають операції шифрування та розшифрування;
- Ключі – особлива послідовність чисел, або символів, що визначає операцію шифрування та розшифрування. Вони можуть бути симетричними, тобто однаковими для шифрування і розшифрування, або асиметричними – різні для шифрування та розшифрування;
- Шифрування – процес перетворення відкритого тексту у зашифровану форму;
- Розшифрування – це процес, при якому зашифрований текст перетворюється назад, у відкрите повідомлення з використанням спеціального ключа;
- Цифрові підписи.

Цифровим підписом є результат спеціального криптографічного перетворення, здійсненого над електронним документом, або повідомленням його власником. Ціль перетворення – доказ незаперечності тексту документа та факту перетворення даних конкретною особою. Він дозволяє переконатися отримувачу, що дані, які передалися, не були змінені після підпису, і що вони були відправлені від ім’я відправника.

Принцип роботи цифрового підпису полягає в тому, що він створюється з використанням криптографічних алгоритмів. Він формується за допомогою хешування вихідного повідомлення і наступного шифрування отриманого значення з використанням закритого ключа відправника.

Отриманий цифровий підпис закріплюється до повідомлення та надсилається отримувачу разом з повідомленням. Тобто, цифровий підпис використовує закриті та відкриті ключі для :

1) Створення цифрового підпису, де відправник використовує свій закритий ключ для створення цифрового підпису, для повідомлення чи документу. Закритий ключ відомий тільки відправнику ;
2) Перевірка цифрового підпису, де отримувач використовує відкритий ключ відправника для перевірки підпису, отриманого разом із повідомленням. Він може бути відомий усім персонам, що обмінюються інформацією.

Також, слід згадати про асиметричне шифрування, в якому і використовуються відкриті та закриті ключі. Процес асиметричного шифрування виглядає так :

1) Коли відправник хоче надіслати зашифроване повідомлення, то він просить відкритий ключ отримувача. Після цього, використовуючи відкритий ключ, відправник зашифровує повідомлення.
2) Отримувач, який має свій закритий ключ, розшифровує отримане повідомлення, яке було зашифроване.

Одними з переваг асиметричного шифрування є :

- Масштабованість – можливість безпечного обміну даними;
- Автентифікація – закритий ключ, який використовується для створення цифрового підпису, дозволяє розпізнати відправника повідомлення;
- Конфіденційність – це означає, що тільки власник відповідного закритого ключа має змогу розшифрувати дані.
- Безпечна передача відкритого ключа – означає, що асиметричне шифрування дає можливість безпечної передачі відкритого ключа відправника до отримувача. Так як відкритий ключ може бути

поширений відкрито, не треба передавати ключ секретно, що робить процес обміну безпечнішим, зручнішим.

Тобто, асиметричне шифрування використовується у процесі створення цифрового підпису та перевірки його оригінальності.

Цифровий підпис грає важливу роль у криптографії. Він дозволяє, забезпечити цілісність даних, що передаються, а також :

1) Цифровий підпис дозволяє встановити, чи було повідомлення відправлено, і чи не було воно підроблено іншою особою. А відкритий ключ використовується для перевірки оригінальності цифрового підпису;
2) Відправник має можливість створити цифровий підпис на ключі, котрий може бути відправлений отримувачу. Також, отримувач може використовувати відкритий ключ відправника для того, щоб перевірити цифровий підпис;
3) Цифровий підпис не дає відправнику відмовитися від авторства відправленого документу, чи повідомлення, бо цифровий підпис створюється за допомогою закритого ключа, тільки власник цього ключа може згенерувати правильний підпис, унеможливлює заперечення щодо відправки документу, або повідомлення.

Нижче показано принцип роботи цифрового підпису (рис 1.1).

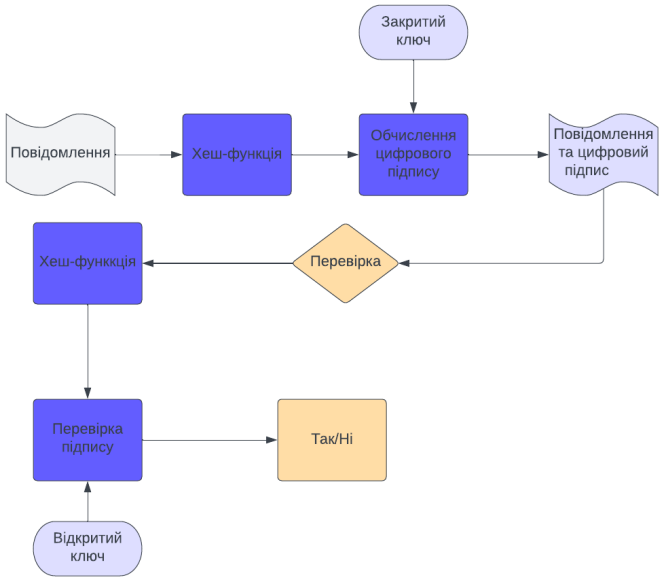


Рисунок 1.1 – Принцип роботи цифрового підпису

2 ДИСКРЕТНИЙ ЛОГАРИФМ

Дискретний логарифм має велике застосування у криптографії. Задача дискретного логарифму є алгоритмічною проблемою, яка є основою для багатьох використовуваних криптосистем. Він визначається у рамках кінцевих груп, тобто нехай є G, що являє собою кінцеву групу з операцією множення, та є b – її елемент. Для елемента h з групи G дискретним логарифмом за основою b є таке число x, що $b^x = h$. Тобто, $x = \log_b h$. Дискретний логарифм не завжди існує у групі G для всіх пар елементів, бо це залежить від вибору основи b та властивостей групи. Також, дискретний логарифм має властивість мультиплікативності: $\log_b h_1 * h_2 = \log_b h_1 + +\log_b h_2$, де можна побачити, що для елементів h_1 та h_2 із групи G, дискретний логарифм їх добутку за основою b дорівнює сумі їх дискретних логарифмів за основою b.

Дискретний логарифм використовується в системах автентифікації, алгоритмах шифрування, алгоритмах цифрового підпису, протоколах безпечного обміну даними.

2.1 Первинні корені $mod\ p$

Первинним коренем $mod\ p$ є число g, яке менше ніж p, та взаємно просте із ним. Тобто, g є первинним коренем, якщо усі степені g від 1 до $\mathrm{p}-1$ дають різні остачі при діленні на p.

Первинні корені застосовують через проблему дискретного логарифму. Дискретний логарифм визначає степінь, до якої треба піднести число g, щоб отримати чисто h по модулю p. Тобто, вирішити рівняння $g^x \equiv h(\mathrm{mod\ p})$ для невідомого x.

Пошук первинних коренів по модулю p є обчислювально складною задачею, але якщо p є простим числом, то можна використовувати різні алгоритми для пошуку. У приклад можна привести алгоритм Поул-Шелла, ідея якого полягає у тому, що потрібно перебирати числа g від 2 до $\mathrm{p}-1$ та перевіряти, чи є g первинним коренем. Цей алгоритм дає можливість ефективно перевіряти властивості первинних коренів.

Можна виділити такі властивості первинних коренів :

1) Якщо g є первинним коренем по модулю p, то для будь-якого числа h ($1 \le h \le p-1$) існує число невідоме число x ($0 \le x \le p-2$), що $g^x \equiv h(\mathrm{modp})$;
2) Якщо g є первинним коренем по модулю p, тоді всі первинні корені дорівнюють $g^x(\mathrm{mod\ p})$;
3) Кількість первинних коренів по модулю p дорівнює $\varphi(\mathrm{p}-1)$, де φ – це функція Ейлера, що визначає кількість чисел, які є взаємно простими з $\mathrm{p}-1$;
4) Кожне число від 1 до $\mathrm{p}-1$ є степенем первинного кореня g.

Первинні корені використовуються у протоколі обміну ключами Діффі-Хеллмана, також у алгоритмі ЕльГамаля тощо.

2.2 Протокол обміну ключами Діффі-Хеллмана

У наш час є багато протоколів для обміну ключами, які, зазвичай, забезпечують створення набору даних для генерації ключів і автентифікацію користувачів.

Одним з них є протокол обміну ключами Діффі-Хеллмана. Він дозволяє двом сторонам домовитися про загальний секретний ключ без передачі самого ключа.

Розглянемо роботу протоколу обміну ключами Діффі-Хеллмана. У приклад візьмемо особу A та особу B, які домовляються про просте число, допустимо, p, яке є спільним параметром, а також погоджуються зі спільним корнем числа g. Потім особа A обирає якесь випадкове, секретне число a, та вирішує: $A = g^a (\text{mod} p)$. Особа B, у свою чергу, обирає випадкове число b, та вирішує: $B = g^b (\text{mod} p)$. Після цього, особи A та B проводять обмін значеннями, які вони вирахували. Після цього особа A отримує значення особи B (B), та вираховує спільний секретний ключ: $K = B^a (\text{mod } p)$, а особа B навпаки отримує значення особи A, та рахує спільний секретний ключ: $K = A^b (\text{mod } p)$. Так як в обчисленнях використовується зведення в ступінь по модулю, то особа A та особа B отримають спільний секретний ключ K. Цей ключ може бути використаний в якості симетричного ключа шифрування для безпечного зв'язку між двома особами.

Симетричне шифрування – це такий метод, коли один ключ використовується для шифрування, дешифрування даних. Мається на увазі, що дві сторони використовують один і той самий ключ для конфіденційності повідомлень.

Одним із переваг протоколу обміну ключами Діффі-Хеллмана є те, що якщо зловмисник перехватить значення A та B, коли відбувається обмін, отримати ключ складно, якщо не знати секретних значень особи A (a) та особи B (b).

Протокол Діффі-Хеллмана забезпечує безпечний обмін ключами, але він не забезпечує захист від форм кібератаки, при яких для перехвату даних застосовуються методи, які дозволяють підключитися до процесу зв'язку. У приклад можно взяти ситуацію, коли особа А та особа В спілкуються, і їх прослуховує особа С. Особа А відправляє якесь повідомлення особі В, а особа С, в свою чергу, перехватує та змінює повідомлення. Ця ситуація є прикладом кібератаки, під назвою: «атака посередника» (рис 2.1).

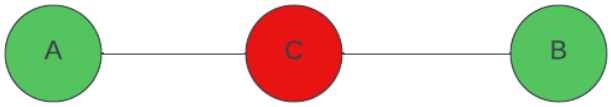


Рисунок 2.1 – Атака посередника

Для вирішення цієї проблеми та запобігання цього використовуються криптографічні протоколи, цифрові підписи тощо.

2.3 Шифрування по ЕльГамалю

Перед тим, як описати шифрування по ЕльГамалю, розглянемо, що таке схема ЕльГамаля.

Схема ЕльГамаля – це криптосистема з відкритим ключем, яка створена на основі обчислення дискретних логарифмів у кінцевому полі. Криптосистема ЕльГамаля є асиметричною (використовує пару ключів для шифрування і розшифрування), потім обирається якесь просте, велике число p. Далі обираємо два псевдовипадкових числа $\mathrm{p}-1$.

Одне з чисел, тобто g, повинно бути первинним коренем по модулю p. А друге число x обирається в якості закритого ключа. Відкритим ключем є трійка чисел: p,g,y.

Розглянемо шифрування :

1) Щоб зашифрувати, наприклад, якусь літеру m, спочатку обираємо якесь ціле число k (разовий ключ) за умови, що $1 < k < (p-1)$;
2) Далі обчислюємо число a: $a = (g^k) mod p$;
3) Обчислюємо число b: $b = (y^k) * m(mod p)$;
4) Отримуємо пару (a,b), яка є зашифрованим текстом.

Слід зауважити, що для одна літера відкритого тексту, відповідає двом літерам зашифрованого тексту.

Для розшифрування буде достатнім отримати співмножник y^k, що можна зробити, якщо вичислити значення $a^x = g^{kx}(p)$.

Розшифрування відбувається за наступною формулою (2.1).

$$m = b * a^{p-1-x}(mod p) \quad (2.1)$$

Розглянемо приклад шифрування літери алфавіту «Ґ». Номер цієї літери у алфавіті є 5-м, тому $m = 5$. Для відкритого ключа візьмемо трійку чисел (p,g,y), тобто, наприклад (11,2,3). Закритий ключ буде число 8. Оберемо разовий ключ $k = 9$.

– Обчислимо число a: $a = (g^k) mod p = 2^9 mod 11 = 6$;
– Обчислюємо число b: $b = (y^k) * m(mod p) = 3^9 * 5 mod 11 = 9$;
– Отримуємо пару (6,9), яка являє собою зашифрований текст.

Для того, щоб зробити розшифрування, треба використати формулу (2.1): $m = b * a^{p-1-x}(mod p) = 9 * 6^2(mod 11) = 324(mod 11) = 5$.

3 ПРОБЛЕМА ДИСКРЕТНОГО ЛОГАРИФМУ ТА МЕТОДИ ВЗЛОМУ

Проблема дискретного логарифму тісно пов’язана з важкістю обчислення логарифмів у циклічних групах, або кінцевих полях.

Проблему дискретного логарифму сформулюємо так: задамо якесь просте число p, ціле число h, основу g, та треба знайти невідомий показник x, щоб $g^x \equiv h(mod p)$.

У відношенні обчислювання, задача дискретного логарифму є важкою. Це означає, що при великих значеннях (p,g,h) немає ефективного алгоритму для її вирішення. Ця властивість лежить в основі багатьох протоколів, наприклад, у протоколі обміну ключами Діффі-Хеллмана, який описувався раніше.

Метою методів злому проблеми дискретного логарифму є пошук найбільш ефективних способів її рішення. Далі будуть описані кілька з них.

3.1 Метод великих-малих кроків

Метод великих-малих кроків є одним з методів для рішення проблеми дискретного логарифму.

Ідея даного методу полягає в тому, щоб прискорити пошук ступеня.

Суть полягає в тому, що треба доказати, що у таблиці є однакові елементи, тобто такі i, j, щоб $g^{hi} = a * g^j = g^{x+v}$. Ця рівність виконується при $x = hi - j(modp)$ або $hi = x + j(modp)$. Дана нерівність $0 \leq hi - j \leq h^2 - 1$ виконується при $1 \leq i \leq h$ та $1 \leq j \leq h$. Для ($i_1 j_1$) та ($i_2 j_2$) ми отримуємо $hi_1 - j_1 \neq hi_2 - j_2$, бо інакше було б: $h(i_1 - i_2) = (j_1 - j_2)$, що можливо тільки при $i_1 = i_2$ та $j_1 = j_2$.

Алгоритм великих і малих кроків виглядає наступним чином:

$G = < g >$ - циклічна група, $|G| = p$.

1) Спочатку треба знайти $h = \left[\sqrt{p}\right] + 1$. Тобто, треба знайти квадратний корінь p та округлити до цілого числа m;
2) Після цього обчислюємо $b = g^h(modp)$;
3) Будуємо таблиці для великих та малих кроків, де: $\{b^i : i = 1, \dots, h\}$ – великі кроки, а $\{a * g^j : j = 1, \dots, h\}$ – малі кроки;
4) Після побудови, треба знайти однакові елементи у таблицях $b^i = g^{hi} = a * g^j$ та обчислити $x = hi - j(modp)$.

Нижче наведено приклад обчислення даного алгоритму:

$G = Z_{22}^* < 4 >$, a = 2

1) $h = \left[\sqrt{9}\right] + 1$=4;
2) b = $4^4(mod23) = 6$;
3) Тепер побудуємо таблицю великих кроків. Спочатку обираємо $h =$ 4, потім проходимо i від 1-го до 5-ти:

Таблиця 3.1 – Великі кроки

i	1	2	3	4	5
$4^{ih} = 6^i$	6	11	16	21	1

Далі будуємо таблицю малих кроків:

Таблиця 3.2 – Малі кроки

j	1	2	3	4	5
$2 * 4^j$	8	9	13	5	1

4) $x = 4 * 5 - 5 = 15$
5) Відповідь: 15

Оцінимо складність даного алгоритму:

– Щоб обчислити $b = g^h(modp)$, нам потрібно виконати $logh$ множень;
– Щоб побудувати таблиці великих-малих кроків – потрібно $2h$ множень;
– Для пошуку однакових елементів у таблиці великих кроків і таблиці малих кроків треба використати спільне сортування цих таблиць за зростанням і спаданням, щоб розкласти поряд однакові значення.

Тобто, відсортувати таблиці великих-малих кроків можна за $2h * log2h$, а для пошуку однакових елементів у цих таблицях потрібно приблизно $2h$ порівнянь.

3.2 Метод Поліга-Сільвера-Хеллмана

Метод Поліга-Сільвера-Хеллмана використовується для рішення задачі дискретного логарифму. Він дозволяє швидко обчислювати логарифми, якщо прості множники невеличкі і якщо відомо розкладання порядку групи на прості множники.

Нехай дано :

$$a^x \equiv b(modp) \tag{3.1}$$

де p – просте число.

Будемо вважати, що $a(modp)$ має порядок $p-1$.

Розглянемо алгоритм :

1) Робимо таблицю для кожного простого числа q (3.2).

$$r_{q,j} \equiv a^{\frac{j(p-1)}{q}}(modp), j = 0, \dots., q-1 \tag{3.2}$$

2) Далі, для кожного простого q знаходимо $\log_a b(modq^a)$.

Нехай:

$$x \equiv \log_a b(modq^a) \equiv x_0 + x_1 q + \cdots + x_{m-1} q^{m-1}(modq^m) \tag{3.3}$$

де, $0 \le x_i \le q-1$.

Із формули (3.1) можемо побачити, що:

$$b^{\frac{p-1}{q}} \equiv a^{\frac{x_0(p-1)}{q}}(modp) \tag{3.4}$$

Знаходимо x_0 :

$$(ba^{-x_0})^{\frac{p-1}{q^2}} \equiv a^{\frac{x_i(p-1)}{q}}(modp) \tag{3.5}$$

Знаходимо x_1:

$$(ba^{-x_0-x1q+\ldots+x_iq^{i-1}})^{\frac{p-1}{q^{i+1}}} \equiv a^{\frac{x_i(p-1)}{q}}(modp)$$

(3.6)

3) Вирішуємо систему порівнянь та знаходимо відповідь.

Для прикладу візьмемо таку задачу: у полі F_{37} треба знайти дискретний логарифм елементу 28, $a = 2$. Тобто, за формулою (3.1): $2^x \equiv 28(mod37)$.

1) Спочатку робимо факторизацію числа (p-1): $p - 1 = 36 = 2^2 * 3^2$;
2) Після цього підрахуємо r (створимо таблицю $r_{i,j}$):

$r_{2,0} = 2^0(mod37) = 1$;

$r_{2,1} = 2^{18}(mod37) = 36$;

$r_{3,0} = 2^0(mod37) = 1$;

$r_{3,1} = 2^{12}(mod37) = 26$;

$r_{3,2} = 2^{24}(mod37) = 10$;

3) Підрахуємо проміжні x:

$q = 2.\ x \equiv x_0 + x_1q_1\left(modq_1^{m_i}\right) \equiv x_0 + 2 * x_1(mod4)$.

Знаходимо x_0 :

$(28 * 2^{x_0})^{18}(mod37) = (28 * 1)^{18}(mod37) = 1$;

$x_0 = \frac{0}{18} = 0$;

Далі знаходимо x_1:

$(28 * 2^{-0})^{\frac{36}{4}}(mod37) = (28 * 1)^9(mod37) = -1(mod37) = 36$;

$x_1 = \frac{18}{18} = 1$;

Після цього знаходимо x:

$x \equiv 1 * 0 + 1 * 2 \equiv 2(mod4)$;

$q = 3. x \equiv x_0 + x_1 q_1 \left(mod q_1^{m_i}\right) \equiv x_0 + 3 * x_1 (mod 9).$

Як і раніше знаходимо x_0 і x_1:

$(28 * 2^{x_0})^{\frac{36}{3}} (mod 37) = (28 * 1)^{12} (mod 37) = 26;$

$x_0 = \frac{12}{12} = 1;$

$(28 * 2^{-1})^{\frac{36}{9}} (mod 37) = (28 * 19)^4 (mod 37) = 10;$

$x_1 = \frac{24}{12} = 2;$

$x \equiv 1 * 1 + 3 * 2 \equiv 7 (mod 9);$

4) Після обчислень отримаємо таблицю:

$\begin{cases} x \equiv 2 (mod 4) \\ x \equiv 7 (mod 9) \end{cases}.$

Вирішуємо систему порівнянь по модулю:

$x = 2 * 9 * 1 + 7 * 4 * 7 (mod 36) = 34$

Відповідь: 34.

Складність даного алгоритму рівна $O(\sum_{i=1}^{k} a_i q_i + \log p)$.

3.3 Р-алгоритм Полларда

Алгоритм Полларда, або р-алгоритм Полларда, був розроблений Джоном Поллардом у 1978-му році. Він є алгоритмом для пошуку дискретного логарифму в кінцевих полях, які основані на циклічних групах.

Цей алгоритм пов’язаний із методом Флойда (метод черепахи та зайця). Даний метод побудований на ідеї, що якщо у послідовності є цикл, і черепаха з зайцем знаходяться на початку послідовності, то рано чи пізно заєць наздожене черепаху у цьому циклі. А якщо циклу немає, то досягне кінця послідовності.

Нехай в нас є циклічна група G = <g>, $a \in G \quad |G| = p$. Розділемо циклічну групу G на кілька підмножин, наприклад: S_1, S_2, S_3. Потім, задамо

$x_0 = 1$ та збудуємо таку послідовність, як: $x_0, x_1, x_2, x_3, \dots, x_n$ за таким принципом $x_{i+1} = f(x_i), i = 0,1, \dots, n$, де випадкова функція $f: G \to G$ визначається за наступними формулами:

$$f(x) = \begin{cases} a * x, \text{коли } x \in S_1, \\ x^2, \text{коли } x \in S_2, \\ g * x, \text{коли } x \in S_3 \end{cases}$$

(3.7)

Можна побачити, що $x_i = a^{b_i} * g^{y_i}$, де $i = 0,1,2, \dots,$ та $y_0 = b_0 = 0$. Згідно формулі (3.7) y та b обчислюються за наступними формулами:

$$y_{i+1} = \begin{cases} y_i, \text{коли } x_i \in S_1, \\ 2y_i (mod p), \text{коли } x_i \in S_2, \\ y_i + 1 (mod p), \text{коли } x_i \in S_3 \end{cases}$$

(3.8)

$$b_{i+1} = \begin{cases} b_i + 1 (mod p), \text{коли } x_i \in S_1, \\ 2b_i (mod p), \text{коли } x_i \in S_2, \\ b_i, \text{коли } x_i \in S_3 \end{cases}$$

(3.9)

Тепер введемо функцію, яка буде мати вигляд: $F: G \times Z_p \times Z_p \to G \times Z_p \times Z_p \to$, що буде обчислювати $F(x_i, y_i, b_i) = (x_{i+1}, y_{i+1}, b_{i+1})$ за формулами, що були наведені вище.

Знайдемо однакові значення x, звідки:

- $a^{b_i} * g^{y_i} = a^{b_{2i}} * g^{y_{2i}}$;
- $a^{b_i - b_{2i}} = g^{y2_i - y_i}$;
- $(b_i - b_{2i}) * \log_g a = y_{2i} - y_i (mod p)$.

Після, треба вирішити останнє порівняння відносно $\log_g a$. Виглядає це так: $d = (b_i - b_{2i}, p)$ рішень $Z_0, Z_1, \dots Z_{d-1}$. Потрібне із них будемо знаходити перебираючи зводячи g у степінь Z_i.

Розглянемо р-алгоритм Полларда для дискретного логарифмування:

1) Кладемо $x_1 = x_2 = 1 \in G, b_1 = b_2 = y_1 = y_2 = 0 \in Z$;
2) Далі кладемо $(x_1, y_1 b_1) = F(x_1, y_1 b_1)$ та $(x_2, y_2 b_2) = F(F(x_2, y_2 b_2))$;
3) У випадку, якщо $x_1 \neq x_2$ – повертаємось до 2-го кроку;
4) Далі треба обчислити: $r = b_1 - b_2 (mod p)$ та якщо r буде дорівнювати 0, тоді дати відмову та закінчити;
5) Потім знаходимо $d = (r, p)$ рішень $Z_0, Z_1, \dots Z_{d-1}$ порівняння $rx = a_2 - a_1 (mod p)$;
6) Знаходимо i, умова для якого є $g^{2i} = a$;
7) Знаходимо відповідь.

Вирішимо приклад:

$G = Z_{29}^* < 2 >, a = 5$;

Таблиця 3.3 – Покладені значення

x_1	y_1	b_1	x_2	y_2	b_2
1	0	0	1	0	0
5	0	1	25	0	2

Продовження таблиці 3.3

25	0	2	18	1	3
9	0	3	6	2	4
18	1	3	24	4	4
7	2	3	8	5	5
6	2	4	12	11	10
12	3	4	19	13	10
24	4	4	6	25	22

19	5	4	24	0	22
8	5	5	8	1	23

– Отримаємо порівняння, яке має вигляд: $2^5 * 5^5 = 2^1 * 5^{23}(mod29)$;

– Вирішуємо:

$r = 5 - 23(mod28) = 10;$

$10x = 1 - 5 = 24(mod28);$

$d = (10,28) = 2;$

$5x = 12(mod14);$

$5^{-1}mod14 = 3;$

$x = 12 * 3 = 8(mod14);$

$Z_0 = 8, Z_1 = 22;$

Зробимо перевірку: $256(mod29) = 24;$

$24 \neq 5;$

Відповідь: $\log_2 5 = 22.$

Цей алгоритм краще використовувати у групах простого порядку, бо при цьому ймовірність відмови є малою.

Обчислювальна складність даного алгоритму залежить від багатьох факторів, включаючи розмір групи, що виконується для операцій дискретного логарифмування, вибору шляху реалізації алгоритму тощо.

Під час генерації послідовності алгоритм порівнює два значення і намагається знайти збіг, що може вказувати на знаходження дискретного логарифму. У випадку, якщо алгоритм не може знайти дискретний логарифм, обчислювальна складність становить приблизно $O\sqrt{N}$, де N – порядок групи.

4 ЦИФРОВИЙ ПІДПИС ПО ЕЛЬГАМАЛЮ

Для того, щоб описати алгоритм цифрового підпису ЕльГамаля, згадаємо про схему ЕльГамаля, яка використовується для шифрування та цифрового підпису. У розділі 2.3 було сказано, що схема ЕльГамаля є ассиметричною криптосистемою, у якій використовуються пара закритого і відкритого ключів для шифрування та розшифрування.

Далі слід згадати, що таке хеш-функція та хеш-значення.

– Хеш-функція є функцією, що на вході отримує дані та перетворює їх у хеш-значення.

– Хеш-функція, у свою чергу, є результатом хеш-функції. Тобто вона є рядком символів фіксованої довжини, що являє собою унікальне представлення вхідних даних.

Тепер опишемо створення цифрового підпису алгоритмом ЕльГамаля.

1) Спочатку розглянемо генерацію ключів:
   – Спочатку треба обрати якесь просте, велике число p та число $g(1 < g < p)$;
   – Далі обираємо якесь випадкове ціле число $x(1 < x < p - 1)$;
   – $y = g^x(modp)$
   – Відкритий ключ буде (p,g,y), а закритий - x;
2) Далі розглянемо генерацію цифрового підпису:
   – Оберемо послання хеш-функцією та отримаємо хеш-значення m;
   – Далі обираємо разовий ключ $k(1 < k < p - 1)$, яке взаємно просте з $p - 1$;
   – Після цього обчислюємо: $a = g^k(modp)$;
   – Обчислимо b із даного виразу: $m = (xa + kb)(mod(p - 1))$;
   – (a,b) – це зашифрований текст, (p,g,y) – відкритий ключ. Відправляємо їх.
3) Перевіримо цифровий підпис:
   – Обчислюємо хеш-значення для повідомлення J;
   – Обчислюємо пару чисел: $y^a a^b(modp)$ та $g^m(modp)$;
   – Якщо отримана пара чисел рівні, то цифровий підпис не був підроблений.

Із переваг цифрового підпису ЕльГамаля можна виділити наступне:

1) Цифровий підпис не потребує безпечної передачі ключа між відправником та отримувачем. Відкритий ключ може бути поширено публічно, а закритий ключ, у свою чергу, не розкривається та залишається у таємниці відправника;

2) Цифровий підпис ЕльГамаля дає можливість переконатися, що відправник повідомлення є справжнім. Тобто, отримувач може перевірити цифровий підпис з використанням відкритого ключа відправника, щоб переконатися, що повідомлення не було змінено;

3) Закритий ключ відправника є унікальним і не може бути повторений на основі відкритого ключа.

Із недоліків цифрового підпису ЕльГамаля слід виділити:

1) Для забезпечення безпеки потрібні ключі довгої довжини. Через це можуть бути великі обчислювальні витрати при обробці та передачі ключів;

2) Якщо цифровий підпис ЕльГамаля реалізовано неправильно, то це може привести до слабкості та вразливості у системі цифрового підпису.

5 ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПРОЕКТУ

Одним із завдань дипломної роботи було написання програми, яка б демонструвала роботу алгоритму Поліга-Сільвера-Хеллмана. Для цього був обраний такий інструментарій:

- VisualStudio Code – середовище розробки;
- JavaScript – мова програмування;
- CSS – мова стилів;

VisualStudio Code має такі переваги:

1) VisualStudio Code безкоштовний у використанні та завантаженні;

2) VisualStudio Code має великий інструментарій для розробки на різноманітних мовах програмування: C++, Java, JavaScript тощо;

3) VisualStudio Code підтримує інтеграцію з різними системами збірки та автоматизації задач. Це дозволяє автоматизувати задачі, що повторюються, наприклад: збірка проекту, компіляція коду і тд. ;

4) VisualStudio Code має швидкий, ефективний інтерфейс, що дозволяє ефективно працювати над проектом;

5) VisualStudio Code надає великий функціонал, такий як: автоматичне доповнення, налаштування, підсвічування синтаксису тощо;

6) VisualStudio Code має потужні інструменти для налагодження коду. Він підтримує покрокове виконання коду тощо;

7) VisualStudio Code надає можливість перевірки та автодоповлення коду, які дозволяють понизити кількість помилок та повисити продуктивність;

8) VisualStudio Code підтримує шаблони, що надає можливість швидко створювати нові файли на основі вже готових шаблонів.

JavaScript має наступні переваги:

1) JavaScript має простий синтаксис, що робить її привабливою для новачків;

2) JavaScript дозволяє легку взаємодію з елементами HTML, та їх стилізацію за допомогою CSS;

3) JavaScript має дуже велику кількість різноманітних плагінів, бібліотек, фреймворків тощо. Це надає великий вибір рішень різноманітних задач;

4) JavaScript може працювати на стороні клієнта, тобто у браузері, та на стороні серверу. Це дозволяє створювати frontend та backend – частини веб-додатків.

5) У мові JavaScript підтримується асинхронне програмування з використанням асинхронних функцій, що дозволяє швидко обробляти операції, які потребують часу;

6) JavaScript є єдиною мовою програмування, яка підтримується усіма основними веб-браузерами, та не потребує встановлення додаткових розширень;

7) У мові JavaScript змінні можуть автоматично приймати різноманітні типи даних. Це прискорює та робить більш легким процес розробки.

Ці переваги роблять мову JavaScript однією з найбільш популярних мов програмування для створення веб-додатків.

Кілька переваг CSS:

1) Використовуючи CSS можна задати стиль один раз і використовувати його для різних елементів веб-додатку;

2) Завдяки CSS можна налаштовувати різні стилі для різних пристроїв. Тобто, можна оптимізувати веб-сторінку для телефонів, комп'ютерів тощо;

3) Можна покращити продуктивність завдяки тому, що можна використовувати окремих CSS-файлів.

Тому, CSS є популярним та потужним інструментом створення, редагування зовнішнього вигляду веб-додатку.

6 РОЗБІР ПРОГРАМНОГО КОДУ ПРОЕКТУ

Метою проекту є знаходження закритого ключа x, $a^x \equiv b(mod p)$ за допомогою алгоритму Поліга-Сільвера-Хеллмана. Для цього потрібно реалізувати наступні функції:

- Факторизація числа $p - 1$;
- Підрахунок усіх r;
- Підрахунок проміжних x для кожного q;

– Знаходження рішення системи порівнянь по модулю.

Розглянемо ці функції більш детально.

Спочатку розберемо факторизацію числа $p - 1$ (рис 6.1).

```
const factorize = (number) => {
    rishenya += `Хід рішення: <br> <br>`;
    rishenya += `1) Факторизація числа p - 1: <br>`;
    rishenya += `<br>`;
    rishenya += `      p - 1 = ${number} = `;
    let res = [];
    if(number < 0) {
        res.push(-1);
        number *= -1;
    }
    for(let i = 2; i <= Math.sqrt(number); i++) {
        let counter = 0;
        while(number % i === 0) {
            counter++;
            number /= i;
        }
        if(counter !== 0) {
            res.push({num: i, power: counter});
        }
    }
    if(number !== 1) {
        res.push({num: number, power: 1});
```

Рисунок 6.1 – Факторизація числа $p - 1$

У даному фрагменті коду виконується факторизація числа $p - 1$.

Спочатку оголошуємо змінну «`res`», у якій буде зберігатися результат факторизації числа «`number`». Ця змінна ініціалізується порожнім масивом «`[]`».

Далі перевіряється, чи є число від'ємним. Якщо воно від'ємне, то до масиву «`res`» додається -1, щоб враховувати від'ємність числа, а саме число помножується на -1, щоб стати додатнім.

Після цього починається цикл «for», який перевіряє числа від 2-х до квадратного корня з «number». Потім нулем ініціалізується змінна «counter». У циклі «while» перевіряється, чи ділиться «number» на «i» без остачі, якщо так, то «counter» збільшується на 1, а «number» ділиться на «i». У випадку, якщо «counter» не дорівнює 0, після виконання циклу, тоді «i» буде одним з множників «number», а степінь дорівнюватиме «counter». У даному випадку об’єкт додається до масиву «res».

Після цього перевіряється, чи «number» = 1. Якщо дорівнює, то «number» є простим числом, тому об’єкт додається до масиву «res».

Тобто, «res» буде зберігати множники та їх степені, які дають число «number» після факторизації.

Щоб перейти до наступної функції – слід розглянути функцію зведення у степінь по модулю (рис 6.2).

```
const modPow = (a, b, p) {
    if(b === 0) {
    return 1;
    }
    let rest = 1;
    while(b) {
    if(b & 1) {
    rest *= a;
    rest %= p;
        }
    a *= (a % p);
    a %= p;
    b >>= 1;
    }
    return rest % p;
}
```

Рисунок 6.2 – Зведення у степінь по модулю

У даній функції використовується функція «modPow», яка виконує зведення у степінь.

Функція приймає три параметри, такі як: «a,b,p», які є основою, показником степеню і модулем відповідно.

Спочатку йде перевірка, чи дорівнює «b» нулю. Якщо так, тоді повертається значення 1, бо число, зведене у ступінь 0 буде дорівнювати 1.

Далі створюємо змінну «rest», яка буде зберігати проміжні результати обчисленні степеню числа «a».

Після цього у циклі «while» йде перевірка, чи дорівнює молодший біт «b» одиниці, де «&» - бітова операція, що виконує бітове І між «b» та одиницею, і якщо результат буде дорівнювати одиниці, то «b» = 1, та треба враховувати «a» в обчисленнях . Цикл буде виконуватися до тих пір, доки «b» не буде рівним 0. Якщо «b» = 1, тоді помножуємо «rest» на «a». Після цього отриманий добуток береться по модулю: «rest %= p». Далі треба звести «a» по модулю. У свою чергу «b» рухається на один біт праворуч: «b >>= 1». Це є рівним діленню «b» на 2 та відкиданню остачі.

У кінці повертаємо «rest % p», що є результатом зведення «a» у степінь «b» по модулю «p».

Тепер розглянемо підрахунок усіх r (створення таблиці для кожного простого q) (рис 6.3).

```
let output = [];
let temp = [];
for(let i = 0; i < factors.length; i++) {
    for(let j = 0; j < factors[i].num; j++) {
        temp.push({r: modPow(a, j * ((p - 1) / factors[i].num), p), power: j * ((p - 1) / factors[i].num)});
    }
    output.push(temp);
    temp = [];
```

Рисунок 6.3 – Код для знаходження r

Створюємо пустий масив «output», в якому будуть усі значення r.

Далі йдуть два цикли (зовнішній та внутрішній) «for». Перший, тобто зовнішній, цикл виконується для кожного елементу масиву «factors», а другий, тобто внутрішній, виконується «factors[i].num» разів, де «factors[i].num» є числом множником із факторизації. Далі виконується рішення, де «a» є основою, «p» - модуль, «j» - це ітераційна змінна, яка є показником степеню. Знайдений результат додається до масиву «temp» разом з показником значення степеню. Після цього масив «temp» додається у масив «output», та знову стає порожнім для наступної ітерації.

Тепер слід розглянути підрахунок x для кожного q (рис 6.4).

```
let x_array = [];
for(let i = 0; i < factors.length; i++) {

rishenya += `    q = ${factors[i].num}: <br>`;
rishenya += `             x = `;
for(let j = 0; j < factors[i].power; j++) {
if(j !== factors[i].power - 1) {
rishenya += `${Math.pow(factors[i].num, j)}·x<sub>${j}</sub> + `;
        }
else {
rishenya += `${Math.pow(factors[i].num, j)}·x<sub>${j}</sub> (mod${Math.pow(factors[i].num, factors[i].power)}) `;
        }
     }
    rishenya += `<br>`;


    let temp_x = Array(factors[i].power).fill(0, 0, factors[i].power);
    let counter = 0;
    for(let j = 0; j < factors[i].power; j++) {
        let power = 0;
        for(let k = 0; k < factors[i].power; k++) {
            power += temp_x[k] * Math.pow(factors[i].num, k);
        }
        let temp_1 = inverseMod(modPow(a, power, p), p);
        let temp_2 = (p - 1) / Math.pow(factors[i].num, j + 1);
        let z = modPow((b * temp_1), temp_2, p);
        rishenya += `        (${b} · ${a}<sup>-${power}</sup>)<sup>${p - 1}/${Math.pow(factors[i].num, j + 1)}</sup>(mod${p}) = `;
        rishenya += `(${b} · ${temp_1})<sup>${temp_2}</sup>(mod${p}) = ${z} `;
        for(let k = 0; k < r_array[i].length; k++) {
        if(z === r_array[i][k].r) {
        let temp_3 = r_array[i][k].power / ((p - 1) / factors[i].num);
        temp_x[counter] = temp_3;
        rishenya += `; <br>        x<sub>${j}</sub> = ${r_array[i][k].power} / ${(p - 1) / factors[i].num} = ${temp_3} <br>`;
            break;
            }
        }
    counter++;
    }


    rishenya += `           x = `;
    let x = 0;
      for(let j = 0; j < temp_x.length; j++) {
      x += temp_x[j] * Math.pow(factors[i].num, j);
      if(j !== temp_x.length - 1) {
      rishenya += `${Math.pow(factors[i].num, j)}·${temp_x[j]} + `;
        }
        else {
      rishenya += `${Math.pow(factors[i].num, j)}·${temp_x[j]} (mod${Math.pow(factors[i].num, factors[i].power)}) = ${x} <br>`;
        }
    }
    x_array.push(x % Math.pow(factors[i].num, factors[i].power));
}
   return x_array;
```

Рисунок 6.4 – Підрахунок x для кожного q

Створюємо порожній масив «x_array», який буде зберігати у собі проміжні значення x. Далі виконується цикл «for» для кожного елементу масиву, що представляючого факторизацію числа «p-1».

У циклі виводиться інформацію про q у змінну «rishenya». Далі створюється тимчасовий масив, який ініціалізується нулями та розміром «factors[i].power». Цей масив буде набувати значень x для q.

Після цього створюється лічильник «counter». Цей лічильник буде відстежувати поточну позицію у масиві «temp_x».

Потім запускається наступний цикл. У ньому обчислюється значення «power» для всіх значень від 0 до «factors[i].power-1».

Далі задаються та обчислюються тимчасові змінні «temp_1» та «temp_2». Змінна «temp_1» є зворотнім елементом «a^(power)(modp)». Обчислюємо «temp_2». Після цього обчислюємо z.

Наступний цикл «for» виконується для кожного елемента масиву «r_array», який представляє значення x для q. У циклі перевіряється, чи «z» є рівним «r_array[i][k].r», тоді обчислюється «temp_3» та присваюється «counter». Далі виводиться значення «temp_3» у змінну «rishenya». Значення «counter» збільшується.

Потім обчислюємо x, яке є сумою для всіх значень j від 0 до «length-1». Далі виводиться інформацію про поточний x до змінної «rishenya».

Після цих дій, повертається масив «x_array», який тримає в собі проміжні значення x для q.

Тепер залишилося розглянути функцію для знаходження рішення системи порівнянь по модулю (рис. 6.5).

```
let mod = 1;
for(let i = 0; i < factors.length; i++) {
    mod *= Math.pow(factors[i].num, factors[i].power);
}
let M_1 = [];
for(let i = 0; i < factors.length; i++) {
    M_1.push(mod / Math.pow(factors[i].num, factors[i].power));
}
let M_2 = [];
for(let i = 0; i < factors.length; i++) {
    M_2.push(inverseMod(M_1[i], Math.pow(factors[i].num, factors[i].power)));
}
let rest = 0;
for(let i = 0; i < factors.length; i++) {
    rest += (x_array[i] * M_1[i] * M_2[i]);
}

rishenya += `     x = `;
  for(let i = 0; i < factors.length; i++) {
    if(i !== factors.length - 1) {
        rishenya += `${x_array[i]}·${M_1[i]}·${M_2[i]} + `;
    }
    else {
        rishenya += `${x_array[i]}·${M_1[i]}·${M_2[i]} (mod${mod}) = `;
    }
}
rishenya += `${rest % mod}`;

return  rest % mod;
```

Рисунок 6.5 – Рішення системи порівнянь по модулю

Створюємо змінну «mod», яка представляє добуток усіх значень для кожного елементу масиву «factors».

Далі створюємо масив «M_1», який містить значення для кожного елементу масиву. Ці значення будуть використані при обчисленні.

Створюється ще один масив «M_2», який містить зворотні елементи масиву «M_1».

Змінна «rest», яка ініціалізується нулем, буде містити підсумкове рішення системи порівнянь.

У циклі «for», що виконується для кожного елементу масиву «factors» збільшується значення «rest».

Виводиться інформація про рішення системи порівнянь до змінної «rishenya».

І в останьому циклі «for» виводиться інформація про поточне обчислення до змінної «rishenya». Виводимо останнє обчислення з модулем.

Значення «rest % mod» повертається у вигляді результату функції, що представляє підсумкове рішення системи порівнянь.

Робота алгоритму завершена.

7 ІНСТРУКЦІЯ ПО ВИКОРИСТАННЮ І ТЕСТУВАННЯ

Головне вікно додатку виглядає наступним чином (рис 7.1).

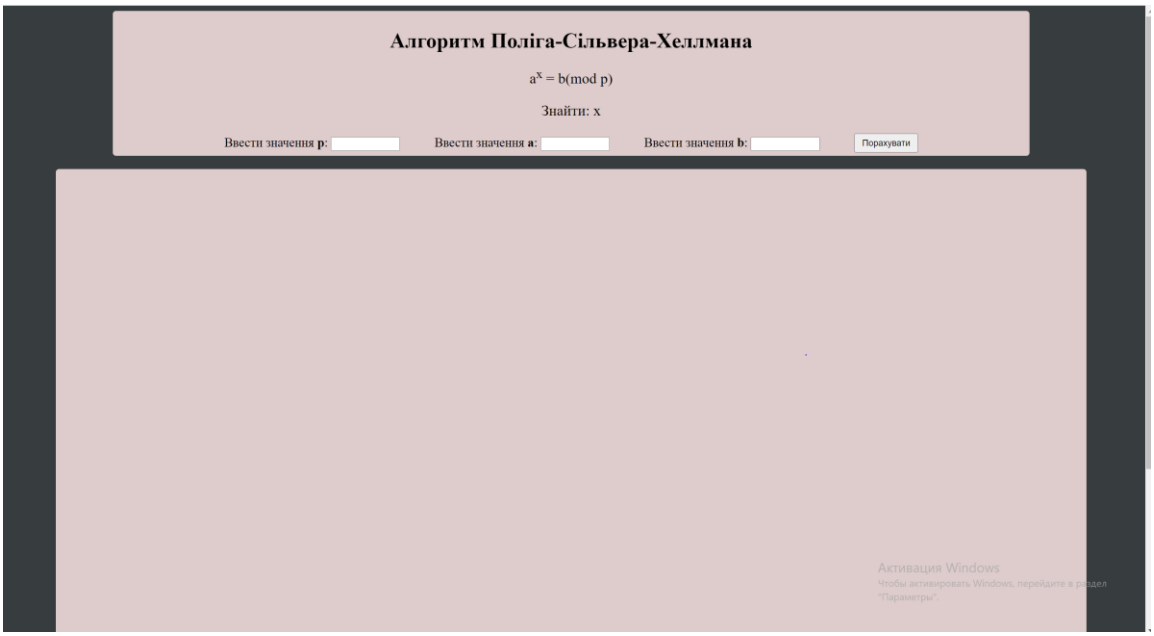


Рисунок 7.1 – Головне вікно додатку

Інтерфейс поділено на дві частини:

1) У першій частині користувач задає значення, які потрібні для обчислення алгоритму;

2) У другій частині відбувається обчислення алгоритму Поліга-Сільвера-Хеллмана.

На рисунку 7.2 можна побачити, як виглядає перша частина, коли її заповнили значеннями.

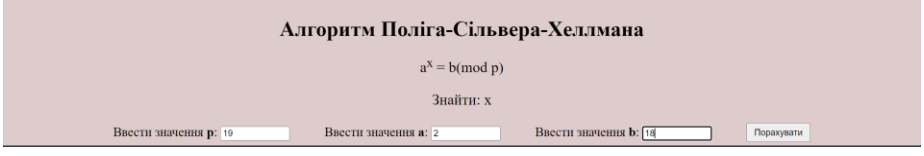
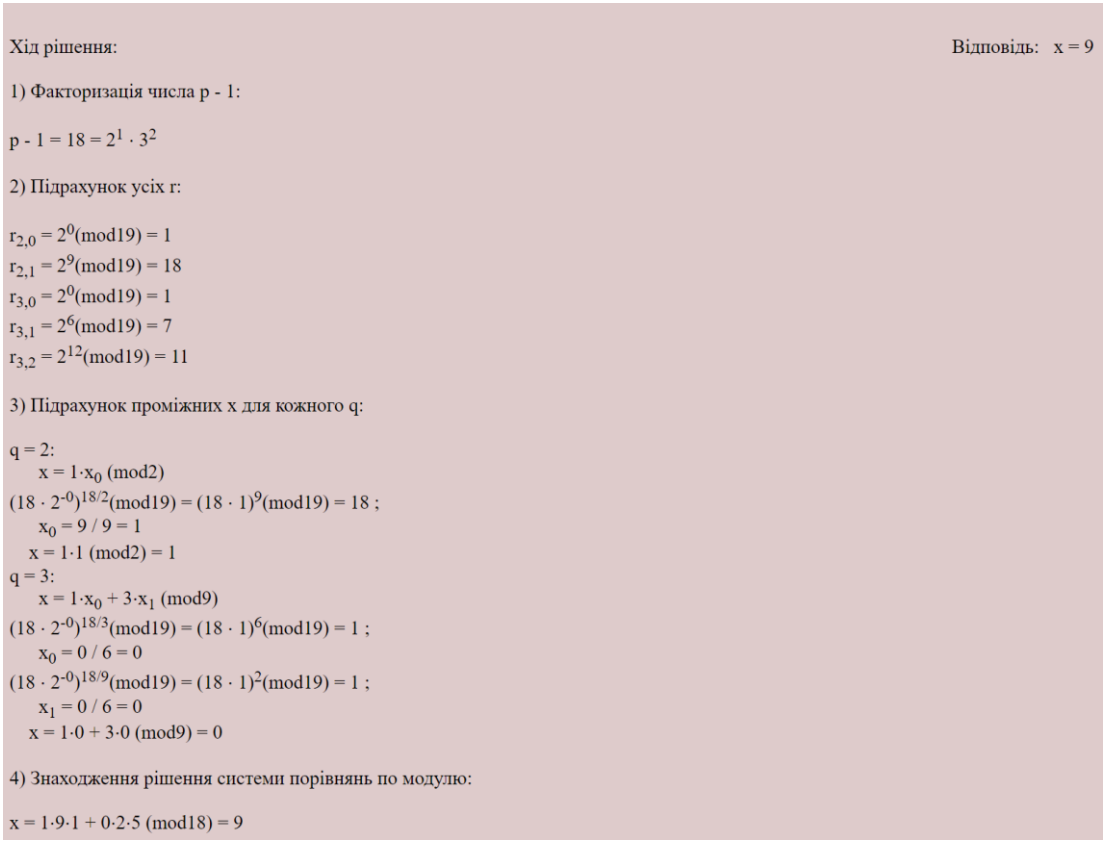


Рисунок 7.2 – Вигляд першої частини

Щоб отримати обчислення алгоритму – треба натиснути на кнопку «Порахувати». Якщо не заповнити усі поля, тоді додаток видасть повідомлення, що треба заповнити ці поля.

На рисунку 7.3 можна побачити другу частину додатку та обчислення алгоритму.



Хід рішення: Відповідь: x = 9

1) Факторизація числа p - 1:

$p - 1 = 18 = 2^1 \cdot 3^2$

2) Підрахунок усіх r:

$r_{2,0} = 2^0 (\text{mod} 19) = 1$
$r_{2,1} = 2^9 (\text{mod} 19) = 18$
$r_{3,0} = 2^0 (\text{mod} 19) = 1$
$r_{3,1} = 2^6 (\text{mod} 19) = 7$
$r_{3,2} = 2^{12} (\text{mod} 19) = 11$

3) Підрахунок проміжних x для кожного q:

q = 2:
$x = 1 \cdot x_0 \ (\text{mod} 2)$
$(18 \cdot 2^{-0})^{18/2} (\text{mod} 19) = (18 \cdot 1)^9 (\text{mod} 19) = 18$;
$x_0 = 9 / 9 = 1$
$x = 1 \cdot 1 \ (\text{mod} 2) = 1$
q = 3:
$x = 1 \cdot x_0 + 3 \cdot x_1 \ (\text{mod} 9)$
$(18 \cdot 2^{-0})^{18/3} (\text{mod} 19) = (18 \cdot 1)^6 (\text{mod} 19) = 1$;
$x_0 = 0 / 6 = 0$
$(18 \cdot 2^{-0})^{18/9} (\text{mod} 19) = (18 \cdot 1)^2 (\text{mod} 19) = 1$;
$x_1 = 0 / 6 = 0$
$x = 1 \cdot 0 + 3 \cdot 0 \ (\text{mod} 9) = 0$

4) Знаходження рішення системи порівнянь по модулю:

$x = 1 \cdot 9 \cdot 1 + 0 \cdot 2 \cdot 5 \ (\text{mod} 18) = 9$

Рисунок 7.3 – Обчислення алгоритму Поліга-Сільвера-Хеллмана

Тепер візьмемо приклад з книги, у якому : p = 37, a = 2, b = 28. Відповідь у прикладі: x = 34 (рис. 7.4).

Рисунок 7.4 – Результат обчислення прикладу з книги

Можна побачити, що відповідь збігається з відповіддю з книги, тому додаток працює правильно.

ВИСНОВКИ

У результаті виконання дипломної роботи було розглянуто роль цифрового підпису у криптографії, описано, що таке дискретний логарифм, асиметричне та симетричне шифрування, а також:

– Описано, що таке первині корені $mod\ p$;
– Розглянуто протокол обміну ключами Діффі-Хеллмана;
– Був розглянутий алгоритм шифрування ЕльГамаля, а також наведено приклад;

Було досліджено проблему дискретного логарифму, та були розглянуті методи взлому:

– Метод великих-малих кроків;
– Метод Поліга-Сільвера-Хеллмана;
– Р-алгоритм Полларда.

Також було розглянуто алгоритм цифрового підпису ЕльГамаля.

Для методу Поліга-Сільвера-Хеллмана був написаний додаток, завдяки якому можна знайти закритий ключ x. Для цього було реалізовано функції факторизації числа $p - 1$, знаходження усіх r, підрахунок проміжних x для q та знаходження рішення системи порівнянь по модулю.

Результат роботи може мати застосування у криптографії, інформаційній безпеці, криптовалюті тощо.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Мухачев В.А., Хорошко В.А. Методи практичної криптографії – К.: ООО «Поліграф-Консалтинг», 2005. – 215 с.
2) Панкратова І.А. Теорико-числові методи криптографії. – К.: ТДУ, 2009. – 120 с.
3) Що таке цифровий підпис ? [Електронний ресурс] – Режим доступу: https://www.e-notary.ru/vse-ob-elektronnoj-cifrovoj-podpisi/.
4) Криптографічні методи [Електронний ресурс] – Режим доступу: https://apmi.bsu.by/assets/files/agievich/dlog.pdf.
5) А.Саломаа, Криптографія з відкритим ключем. : Пер. з англ. – М.: «Світ», 1995 – 318 с.
6) Електронний цифровий підпис Ель-Гамаля [Електронний ресурс] – Режим доступу: https://studfile.net/preview/16434365/page:4/.
7) Алгоритм Цифрового підпису Ель-Гамаля [Електронний ресурс] – Режим доступу: http://ni.biz.ua/5/5_7/5_7037_algoritm-tsifrovoy-podpisi-el-gamalya-EGSA.html.