НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

МЕТОДИ РЕАЛІЗАЦІЇ КРИПТОГРАФІЧНИХ МЕХАНІЗМІВ

Комп'ютерний практикум № 2

Реалізація алгоритмів генерації ключів гібридних криптосистем.

Підгрупа 1А.

Виконали:
Волинець Сергій ФІ-42мн
Сковрон Роман ФІ-42мн
Радомир Беш ФІ-42мн

Київ — 2025

1 Мета

Дослідження алгоритмів генерації псевдовипадкових послідовностей, тестування простоти чисел та генерації простих чисел з точки зору їх ефективності за часом та можливості використання для гененерації ключів асиметричних криптосистем.

2 Завдання на лабораторну роботу

Дослідити різні методи генерації випадкових послідовностей для засобів обчислювальної техніки. Дослідити ефективність за часом алгоритми тестування на простоту різних груп – імовірнісних, гіпотетичних та детермінованих. Порівняти ймовірність похибки різних імовірнісних тестів (Ферма, Соловея-Штрасена та Мілера-Рабіна з різною кількістю ітерацій) з ймовірністю похибки при виконанні обчислень на ПЕОМ. Розглянути алгоритми генерації простих чисел "Чебишова" та Маурера та провести порівняльний аналіз їх складності. Розробити бібліотеку генерації псевдовипадкової послідовності, тестування простоти чисел та генерації простих чисел для Intel-сумісних ПЕОМ. Розмірність чисел – 1024/2048/4096 біт.

3 Дослідження

Генерація випадкових послідовностей є важливою частиною криптографії, моделювання, статистичного аналізу та інших сфер, де використовуються випадкові значення. Для цього розрізняють два основні типи генераторів випадкових чисел. Перший тип – це псевдовипадкові генератори (PRNGs) — засновані на детермінованих алгоритмах. Їх випадковість є лише зовнішньою, бо вони генерують числа на основі деякого початкового стану (наприклад сіду). Другий тип – це апаратні генератори випадкових чисел, які використовують фізичні процеси (наприклад, внутрішній шум, швидкість натискання клавіш користувачем на клавіатурі, тощо) для генерації справжньої випадковості.

3.1 Методи генерації випадкових послідовностей

Алгоритмвів генерації випадкових послідовностей досить багато. Наприклад, лінійний конгруентний метод (LCG). Це один з найпростіших псевдовипадкових генераторів. Нове просте число обчислюється за формулою:

$$X_{n+1} = (aX_n + c) \bmod m,$$

де a, c, m – деякі константи. Мінуси такого генератора полягають в обмеженій періодичності, що може призвести до погіршення випадковості на великих інтервалах. Цю проблему можна вирішити використовуючи лінійні регістри зсуву, що є певним узагальненям лінійного конгруентного алгоритму, і мають оволі великі періоди.

Ще один метод це вихор Мерсенна, що є одним з найпоширеніших і надійних псевдовипадкових генераторів, що використовує складний масив значень для створення довгих періодів. Основна ідея полягає в тому, що до початкової ітерації, яка

ініціює процедуру, застосовується серія бітових операцій. Після їх виконання отримують нову послідовність, перший член якої вважається псевдовипадковим числом. Цей алгоритм має величезний період: $2^{19937} - 1$, що дає високу якість генерації випадкових чисел.

Наступний приклад – це генератори випадкових чисел на основі криптографічних геш функцій, таких як SHA-256. Вони призначені для використання в криптографії і мають високий рівень випадковості, але можуть бути повільнішими через складність хешування.

Нами було використано NativePRNGBlocking є одним з типів генераторів випадкових чисел в Java, який належить до класу java.security.SecureRandom. Він використовує платформонезалежну реалізацію генератора випадкових чисел на основі апаратних джерел випадковості, якщо такі є. У разі їх відсутності, він використовує програмні алгоритми для генерації випадкових чисел

3.2 Алгоритми тестування на простоту чисел

Тестування на простоту чисел є важливим етапом у криптографічних алгоритмах, оскільки прості числа часто використовуються в процесах шифрування та підпису. Існують різні методи перевірки на простоту, які дозволяють зменшити обчислювальну складність. Один з найпростіших способів перевірки на простоту – це метод пробних ділень. За допомогою цього методу можна перевірити, чи тестове число ділиться на прості числа, менші за нього. Цей метод має високу обчислювальну складність, особливо для великих чисел, і стає непридатним для реальних криптографічних застосувань через значну кількість обчислень. Попри це, метод пробних ділень може бути корисним на етапах початкової перевірки числа на простоту, якщо обмежити список дільників кількома першими простими числами. Це значно зменшує кількість перевірок і може суттєво прискорити обчислення на перших етапах складних алгоритмів. Зокрема, такий підхід дозволяє швидко відкинути очевидно складні числа, не звертаючись до повної перевірки простоти.

Тест Ферма є імовірнісним тестом для перевірки простоти числа. Формула Перевірки виглядає наступним чином: якщо $a^{p-1} \equiv 1 \bmod p$, де p – це число, що перевіряється, a – випадкове число, то p ймовірно просте. Недоліком даного методу є те, що він може дати помилкові позитивні результати для складених чисел (повертає так звані псевдопрості числа).

Іншим алгоритмом є тест Соловея-Штрасена. Це ймовірнісний метод перевірки простоти чисел, який використовує властивості символу Якобі. Він забезпечує більшу точність у порівнянні з тестом Ферма, оскільки здатний виявляти числа Кармайкла як складені. Алгоритм працює в кілька раундів, де кожен раунд знижує ймовірність помилки, і результат тесту дає високу ймовірність того, що число є простим.

Тест Мілера-Рабіна – це імовірнісний тест, який на основі теореми про залишки перевіряє простоту числа. Даний алгоритм виконує декілька ітерацій для кращої надійності, бо більша кількість ітерацій дозволяє знизити ймовірність похибки. Насправді, це працює для усіх ймовірнісниих тестів.

3.3 Генерація простих чисел

Для генерації простих чисел можна використати алгоритм Чебишова. Він використовує апроксимацію і перебір для генерування простих чисел, що є ефективним для малих чисел. Даний алгоритм може бути не дуже швидким при великих числах через необхідність перевірки кожного числа на простоту. Натомість, алгоритм Маурера є більш складним, але більш ефективним для великих чисел. Він працює швидше для генерації простих чисел з більшими бітовими розмірами, завдяки оптимізаціям в алгоритмі.

Складність алгоритма Чебишова залежить від кількості ітерацій перевірки простоти, і рівна $O(k\sqrt{n})$, де k – це кількість спроб для генерації простого числа довжини n. Складність генерації простих чисел алгоритм Маурера в найгіршому випадку рівна $O(mkn^2)$, де k – це кількість ітерацій тесту Мілера-Рабіна, n – це розмір числа в бітах, і m – це кількість спроб для генерації простого числа. Для великих чисел (2048 біт і більше) кількість спроб зазвичай невелика, але складність перевірки простоти залишається основною складовою часу.

4 Реалізація

У екосистемі мови програмування Java існує підтримка багатьох різних криптопримітивів таких як генераторів псевдовипадкових чисел, перевірки на простоту, хеш функцій, як симетричних та асиметричних алгоритмів та функцій. Було прийнято рішення реалізувати генератор простих чисел за допомогою вбудованого криптостійкого генератора псевдовипадкових чисел та обирати серед них прості числа і таким чином побудувати генератор простих чисел.

Було імплементовано та реалізовано два методи генерації простих чисел. Перший це обирати випадкове число довжини фіксованої, наприклад 1024, далі перевіряти тестом на простоту, обрано тест Мілера-Рабіна чи просте число чи ні. Другим способом було вирішено зробити модифікацію першого способу, але перед викликом тесту на простоту перевірити кандидата методом пробного ділення, певною кількістю перших малих простих чисел.

Результат виявися здебільшого очевидний. Виклик функції перевірки на простоту набагато складніший порівняно із діленням. З точки зору реалізації було усереднено кількість згенерованих простих чисел для формалізації та коректності метрики.

З таблиці очевидно, що максимальне ефективність метода пробних ділень, досягається при досить невеликій кількості простих чисел близько 2000 перших простих чисел, проте до 10_000 простих чисел все ще є дієвим способом, але вже не таким ефективним.

Табл. 1: Порівняння ефективності метода пробних ділень для генерування псевдовипадкових простих чисел

Максимум | кількість	**Час**
10 | 4	12898 / 8755 ≈ 1.2
100| 25	24607 / 10965 ≈ 2.1

1_000 | 168	35140 / 16642 $\approx$ 2.2
10_000 | 1229	50012 / 14013 $\approx$ 3.5
20_000 | 2262	55351 / 15590 $\approx$ 3.4
50_000 | 5133	59648 / 23184 $\approx$ 2.6
100_000 | 9592	89121 / 42280 $\approx$ 2.1
200_000 | 17984	65071 / 52967 $\approx$ 1.2
250_000 | 22044	91194 / 68906 $\approx$ 1.3

В таблиці було проведено порівняння прямого метода генерації випадкових простих чисел та його модифікацію за допомогою простих чисел, тобто просту комбінацію двох методів. Однак більш складні реалізації таких комбінованих методів вимагають більш сучасних та оптимізованих методів, до прикладу застосування інших методів перевірки на простоту, можливо навіть детермінованих або ж застосування методів для суміжних областей таких як задача факторизації тощо.

5 Висновки

У дослідженні було розглянуто методи генерації випадкових чисел та тестування на простоту. Використання NativePRNGBlocking в Java забезпечує високу безпеку генерації випадкових чисел за допомогою апаратних та програмних алгоритмів. Тестування на простоту за допомогою методів Ферма, Соловея-Штрасена та Мілера-Рабіна є ефективними алгоритмами, які мають велике значення для криптографії, де прості числа використовуються для генерації ключів та підписів.

У результаті аналізу алгоритмів генерації випадкових простих чисел, зокрема Чебишова та Маурера, виявлено, що алгоритм Чебишова ефективний для малих чисел, але для великих чисел його складність зростає. Алгоритм Маурера, хоча й складніший, значно швидший для великих чисел завдяки оптимізації. Порівняння ефективності методів генерації простих чисел показало, що використання методу пробних ділень перед тестом на простоту може значно прискорити процес генерації чисел, особливо для великих бітових розмірів. Також що використання комбінованих методів для фільтрації кандидатів до застосування ресурсозалежних функціях перевірки на псевдовипадкові прості числа повинно бути застосовано для генерації великої кількість в умовах обмежених ресурсів часу та пам'яті.