Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Навчально-науковий фізико технічний інститут
Кафедра інформаційної безпеки

Звіт

З практичного завдання №2
із дисципліни «Методи реалізації криптографічних механізмів»
Тема: «Реалізація алгоритмів генерації ключів гібридних криптосистем»

Виконав:
Студент групи ФБ-41мн
Шерстюк А. В.
Вигрaновський М.В

Київ-2025

Схема генератора ПВЧ для User Endpoint terminal. Особливості побудови генератора простих чисел для моделей тонкого/товстого клієнта

Вступ

Генератори псевдовипадкових чисел (ГПВЧ) стали невід'ємною частиною сучасної інформаційної технології, знаходячи застосування в широкому спектрі областей, від криптографії до моделювання та ігор. Ці механізми, спроектовані для генерації чисел, що здаються випадковими, забезпечують надійність і криптографічну стійкість, необхідну для багатьох застосувань.

У цій сфері розрізняють три основні класи генераторів: генератори справжніх випадкових чисел, які використовують фізичні процеси для збору ентропії; генератори псевдовипадкових чисел, які використовують детерміновані алгоритми, але можуть не мати криптографічної стійкості; та криптографічно стійкі генератори псевдовипадкових чисел, які забезпечують вищий рівень надійності та стійкості, необхідний для безпечного застосування.

Методи генерації

Лінійні конгруентні генератори (ЛКГ) є одними з найпоширеніших і простих методів генерації псевдовипадкових чисел. Вони базуються на рекурентній формулі, яка генерує послідовність чисел шляхом лінійного перетворення попереднього числа. Формула, зазвичай, виглядає так: $\mathbf{X_{n+1} = (aX_n+c) mod\ m,}$ де $\mathbf{X_n}$- поточне число в послідовності, $\mathbf{X_{n+1}}$- наступне число в послідовності, a, c та m- параметри генератора, **mod**- операція залишку від ділення.

Лінійні конгруентні генератори (ЛКГ) хоч і є одними з найпростіших методів генерації псевдовипадкових чисел, вони не завжди вважаються сучасними у контексті криптографічних застосувань та вимогливих областей, де висока криптографічна стійкість є ключовою. Основні причини цього включають обмеженість в періоді, відносну передбачуваність та недостатню надійність у криптографічних аспектах.

У порівнянні з простими алгоритмами, які базуються на математичних формулах, криптографічно-стійкі генератори псевдовипадкових чисел надають вищий рівень криптографічної стійкості та надійності, та обладають наступними властивостями:

- **Непередбачуваність**- жоден зловмисник не повинен мати змоги передбачити наступне число, знаючи попередню послідовність.
- **Стійкість до атак**- генератор повинен витримувати криптографічні атаки, які можуть спробувати визначити його внутрішній стан.

- **Пряма секретність-** навіть якщо внутрішній стан генератора скомпрометовано в певний момент часу, послідовності, згенеровані раніше, повинні залишатися непередбачуваними.
- **Зворотна стійкість-** компрометація поточного стану не повинна дозволяти зловмиснику реконструювати послідовності, згенеровані після останнього відновлення посіву (NIST SP 800-90A).

Одним з типів КСГПВЧ є генератори, які базуються на **блокових шифрах**. Ці генератори використовують шифрування блоків з випадковим ключем та початковим вектором, щоб згенерувати послідовність псевдовипадкових чисел. Ключ та початковий вектор можуть бути випадково згенерованими або визначатися користувачем. Крім того, генератори псевдовипадкових чисел на основі блокових шифрів можуть використовувати різні режими роботи для забезпечення різноманітності та безпеки генерації випадкових послідовностей. Наприклад, режим лічильника (CTR) дозволяє генерувати послідовності псевдовипадкових бітів з великою швидкодією та масштабованістю. Режим лічильника (CTR) є одним із режимів роботи блочних шифрів, який використовується для генерації послідовностей псевдовипадкових бітів або блоків на основі ключа та початкового вектора. Основна ідея режиму CTR полягає в тому, щоб використовувати блоковий шифр для шифрування послідовності чисел, починаючи з деякого початкового значення, яке називається лічильником. Кожне наступне значення лічильника генерується шляхом інкременту попереднього значення.

Однак, теоретичний аналіз виявив, що CTR_DRBG має певні недоліки, пов'язані з розміром блоку шифру. Якщо використовується AES (з розміром блоку 128 біт), і генерується більше ніж 128 біт випадковості за один виклик, фактичний рівень безпеки може обмежуватися розміром блоку, а не розміром ключа. Це означає, що вихід може бути відрізнений від справжнього випадкового джерела, якщо береться понад 128 біт, хоча наразі невідомо, як можна використати цю проблему на практиці, коли використовується AES. Крім того, для забезпечення прямої секретності схема NIST CTR_DRBG вимагає видалення ключа (key erasure) після виводу запитаної випадковості, що досягається генеруванням додаткової випадковості для заміни ключа. Цей процес є неефективним з точки зору продуктивності.

Іншими методами є використання **хеш-функцій** для генерації псевдовипадкових чисел. Хеш-функції приймають вхідні дані будь-якої довжини і повертають фіксований вихідний хеш, який вважається псевдовипадковим. Застосування криптографічностійких хеш-функцій, таких як SHA-256 або SHA-3, забезпечує високий рівень безпеки та стійкості. Особливу увагу слід приділити вибору початкових значень (ініціалізаційних векторів або ключів) для генерації випадкових чисел у криптографічно-стійких генераторів. Неправильний вибір початкових значень може призвести до зрушення у безпекових властивостей генератора

Тестування простоти

Тести простоти є фундаментальною частиною криптографічної генерації ключів. Вони поділяються на **імовірнісні** (практичні), **гіпотетичні** (залежні від недоведених припущень) та **детерміновані** (теоретично доведені).

Основою багатьох **імовірнісних тестів** є мала теорема Ферма: якщо **p**- просте число, то для будь-якого цілого числа **a**, взаємно простого з **p**, виконується $\mathbf{a^{p-1} \equiv 1 \bmod p}$. На жаль, виконання цієї умови навіть для всіх можливих **a** не гарантує простоту **p** через існування складених чисел Кармайкла (наприклад, 561 = 3 * 11 * 17), які задовольняють цю умову. Вони доводять складеність, але не простоту (крім випадку, коли перевіряється достатня кількість баз). Тест Міллера-Рабіна є стандартом де-факто для криптографічних застосувань, оскільки він має найнижчу ймовірність похибки серед імовірнісних тестів з аналогічною часовою складністю.

Тест	Максимальна ймовірність похибки (1 ітерація)	Асимптотична складність (T - тест, k - ітерації)
Ферма	Непередбачувана (висока)	$T=O(k * \log^3 p)$
Соловея-Штрассена	< 1/2	$T=O(k * \log^3 p)$
Міллера-Рабіна	< 1/4	$T=O(k * \log^3 p)$

На відміну від імовірнісних тестів, **детерміновані алгоритми**, як-от AKS (Agrawal–Kayal–Saxena), безумовно доводять простоту числа за поліноміальний час, але практична часова складність ($O(\log^6 p)$ або гірше) робить його значно повільнішим за Міллера-Рабіна, тому на практиці він не застосовується. Історично існував тест Міллера, який був детермінованим, але лише за умови істинності недоведеної узагальненої гіпотези Рімана (GRH). Рабін пізніше модифікував його, щоб зробити метод імовірнісним і незалежним від GRH, створивши тест Міллера-Рабіна.

Для криптографічних застосувань необхідно визначити достатню кількість ітерацій k тесту Міллера-Рабіна, щоб гарантувати рівень безпеки, порівнянний із довжиною ключа (наприклад, 2^{-128} або 2^{-256}). Якщо потрібен рівень безпеки 2^{-128}, необхідно виконати $2k > 128$, тобто $k > 64$ ітерацій. Для безпеки 2^{-256} необхідно $k > 128$ ітерацій. Стандарт FIPS 186-4 часто вимагає $k > 40$, що дає математичну ймовірність похибки $P_{MR}(40) < 2^{-80}$.

Порівняння цієї математичної ймовірності з імовірністю апаратних помилок на ПЕОМ (Soft Error Rate, SER) є ключовим для прийняття рішення щодо практичної кількості ітерацій. Типовий рівень SER для сучасних мікросхем, спричинений радіацією,

оцінюється приблизно в 10^{-17} до 10^{-20} помилок на біт-годину. Це відповідає ймовірності 2^{-56} до 2^{-66}.

Аналіз показує, що ймовірність того, що фізична помилка (Soft Error) призведе до зміни критичного біта під час виконання модульного експоненціювання 4096-бітного числа, є набагато порядків вищою, ніж ймовірність того, що алгоритм Міллера-Рабіна помилково ідентифікує складене число як просте при кількості ітерацій k=80. Оскільки детерміновані алгоритми, такі як ECPP або AKS, вимагають значно більше часу на виконання, вони збільшують загальний час експозиції до апаратних помилок. Таким чином, при $k > 80$, математична недосконаліть тесту Міллера-Рабіна стає вторинною порівняно з фізичною надійністю самого обчислювального пристрою. Це обґрунтовує використання швидкого імовірнісного тесту Міллера-Рабіна як оптимального та достатнього для високошвидкісної криптографічної генерації ключів.

Порівняння ймовірності похибки: Математична vs. Апаратна

Подія	Рівень похибки P	Логарифмічна оцінка $\log_2(P)$	Висновок
Міллер-Рабін k=80	$< 2^{-160}$	-160	Криптографічна безпека 80 біт (набагато нижче SER)
Міллер-Рабін k=128	$< 2^{-256}$	-256	Криптографічна безпека 128 біт
Апаратна Soft Error (на біт-годину)	$\approx 10^{-17}$ до 10^{-20}	-56 до -66	Фактичний ліміт надійності системи

Генерації великих простих чисел

Для створення криптографічних ключів розмірністю 1024–4096 біт використовуються два основні підходи до генерації простих чисел: імовірнісний пошук за щільністю (метод "**Чебишова**") та конструктивний метод **Маурера**.

Пафнутій **Чебишов** зробив значний внесок у теорію чисел, зокрема в оцінки розподілу простих чисел, що дозволяє оцінити, скільки випадкових чисел потрібно перевірити, щоб знайти просте. Метод "Чебишова" тут інтерпретується як простий імовірнісний пошук:

1. Вибір випадкового кандидата N заданої розмірності (наприклад, 4096 біт), переконавшись, що він не є парним.

2. Послідовне тестування N, N+2, N+4, … за допомогою імовірнісного тесту простоти (Міллера-Рабіна) доти, доки не буде знайдено просте число.

Теорема про прості числа стверджує, що щільність простих чисел навколо N становить приблизно 1 / ln N. Отже, у середньому необхідно виконати ln N тестів Міллера-Рабіна, щоб знайти одне просте число. Загальна часова складність такого простого пошуку (PS) становить $T_{PS} = O(\ln N * T_{MR}(N))$, $T_{MR}(N) = O(\log^3 n)$$, що дає загалом $O(\log^4 n)$

Алгоритм **Маурера** є конструктивним рекурсивним методом генерації великих простих чисел. Його ключова перевага полягає в тому, що він не лише знаходить просте число, але й одночасно надає доказ його простоти, базуючись на властивостях великого простого дільника q числа P-1.

Алгоритм Маурера спеціально розроблений для відповідності криптографічним стандартам, які часто вимагають, щоб згенеровані прості числа мали певні структурні особливості (наприклад, щоб P-1 мало великий простий дільник, що необхідно для запобігання атакам, заснованим на факторизації P-1).

Практична часова складність алгоритму Маурера з використанням оптимізованих методів множення може бути доведена до $O(\log^4 n)$, що робить його конкурентним до простого пошуку. Хоча теоретична часова складність може бути вищою, висока якість вихідного матеріалу є вирішальним фактором.

Стратегічний вибір методу генерації простих чисел для криптографічної бібліотеки (4096 біт) повинен ґрунтуватися на балансі між швидкістю та криптографічною якістю.

Простий пошук може бути незначно швидшим у середньому, але він дає лише імовірнісно просте число, якщо не використовується надзвичайно велика кількість ітерацій Міллера-Рабіна.

Натомість, алгоритм Маурера генерує доказово прості числа, що відповідає найвищим стандартам криптографії (наприклад, FIPS 186-4 щодо генерації RSA-модулів). Використання Маурера усуває необхідність подальшого повного детермінованого доведення простоти (наприклад, ECPP), яке є значно повільнішим.

Оскільки вимоги до криптографічної стійкості (особливо для 4096-бітних ключів) диктують мінімізацію будь-яких математичних похибок, Маурер є кращим вибором для високобезпечної бібліотеки, незважаючи на схожу асимптотичну складність з імовірнісним пошуком.

Алгоритм	Оцінка складності	Тип виходу	Ключова перевага
Чебишов	$O(\log^4 n)$	Імовірнісно просте	Висока швидкість у середньому

Маурера	$O(\log^4 n)$	Доказово просте	Висока криптографічна надійність, відповідність стандартам

Архітектура генератора ПВЧ для User Endpoint terminal

Для терміналу користувача доцільно розглядати генератор ПВЧ як окремий криптографічний модуль із чітко визначеними межами. У його основу закладається кілька незалежних джерел ентропії: штатні механізми операційної системи та процесора, апаратні модулі на зразок TPM або TRNG, а за наявності- додаткові часові та мережеві джиттер-дані. Уся зібрана ентропія агрегується та проходить попередні статистичні перевірки, після чого пропускається через криптографічну хеш-функцію, що дає нормалізований посів сталої довжини. На цьому матеріалі ініціалізується криптографічний генератор випадкових бітів (CTR_DRBG або HMAC_DRBG). Внутрішній стан DRBG і ключі зберігаються в захищеній ділянці пам'яті, яка не вивантажується на диск, та регулярно оновлюються як за обсягом сформованих даних, так і при зміні контексту роботи терміналу. Зовні модуль надає простий сервісний інтерфейс на кшталт get_random_bytes, get_random_uint, а для криптографічних компонентів- спеціалізовані виклики для отримання матеріалу ключів і одноразових значень. При старті й у процесі роботи виконуються самотести- як перевірка коректності реалізації, так і онлайн-моніторинг на предмет статистичних аномалій; у разі збою генератор блокує видачу вихідних даних і формує сигнал у систему моніторингу.

Генерація простих чисел у такій архітектурі спирається на ГПВЧ як на базовий будівельний блок. Для отримання великого простого числа потрібної довжини модуль спершу запитує у DRBG випадковий бітовий вектор, коригує його до непарного значення з установленими старшими бітами, а потім проганяє через попереднє "сівання" діленням на набір малих простих. Кандидати, що пройшли фільтр, тестуються на простоту ймовірнісними методами, як-от багаторазовим тестом Міллера–Рабіна з різними базами. Якщо число успішно проходить усі раунди, воно вважається криптографічно придатним простим; у разі потреби виконуються додаткові перевірки, пов'язані з конкретним алгоритмом (наприклад, перевірка взаємної простоти $p-1$ та відкритого експонента E для RSA). Усе це працює повністю усередині довіреного криптомодуля, не розкриваючи проміжні значення назовні.

У моделі тонкого клієнта, де кінцева точка має обмежені ресурси й переважно функціонує як "розумний термінал" поверх серверної логіки, генератор ПВЧ зазвичай зводиться до використання наявного в платформі КСГПВЧ. Браузер або легкий агент покладається на WebCrypto чи на системний ГПВЧ для формування локальних одноразових значень, токенів сесії, тощо. Генерацію ж великих простих і довготривалих ключів логічно винести на серверний бік, де є повноцінний криптомодуль. Там же

відбувається проведення усіх потрібних тестів, аудит і сертифікація. Тонкий клієнт у цьому випадку отримує вже готові ключі й параметри за захищеним каналом, а основні ризики зміщуються в площину захисту транспорту: надійний TLS, коректна перевірка сертифіката. Локальний інтерфейс до ПВЧ на тонкому клієнті обмежують простими операціями, не допускаючи витоку інформації про внутрішній стан або статистику генератора через помилки бізнес-логіки.

У моделі товстого клієнта ситуація протилежна: застосунок або спеціалізований термінал має достатньо обчислювальних ресурсів, може взаємодіяти з TPM, смарткартами чи власними TRNG, і тому доцільно розміщувати повноцінний криптомодуль безпосередньо на endpoint'і. Тут ГПВЧ реалізується як окрема бібліотека чи сервіс із DRBG, інтегрований з апаратними джерелами ентропії та з резервним використанням RNG операційної системи. На цій основі вже локально виконуються всі операції з генерації великих простих та ключових пар- як для офлайн-режиму, так і для сценаріїв наскрізного шифрування, де сервер не повинен бачити приватні ключі. Для підвищення продуктивності використовуються оптимізовані big-integer-бібліотеки, паралельні тести Міллера–Рабіна, кеші малих простих. Приватні ключі зберігаються лише у зашифрованому вигляді, причому ключ шифрування може бути пов'язаний із TPM або іншими апаратними засобами. Особлива увага приділяється захисту від локального шкідливого ПЗ і побічних каналів: мінімізація поверхні атаки, ізоляція криптомодуля, вирівнювання часу виконання критичних операцій. Таким чином, для тонкого клієнта генератор ПВЧ – це легкий шлюз до серверної криптографії, тоді як для товстого клієнта він перетворюється на повноцінне ядро локальної системи керування ключами та генерації простих чисел.

Висновки

У роботі розглянуто принципи побудови генераторів псевдовипадкових чисел та методи формування простих чисел для криптографічних застосувань. Проаналізовано властивості ЛКГ, криптографічно стійких DRBG та механізмів на основі блочних шифрів і хеш-функцій. Показано, що використання CTR_DRBG та HMAC_DRBG забезпечує необхідну стійкість за умови коректного збирання ентропії та регулярного оновлення внутрішнього стану.

Досліджено тести простоти та наведено їх порівняльний аналіз. Визначено, що тест Міллера–Рабіна є оптимальним для практичної генерації великих простих чисел, оскільки його ймовірність похибки при достатній кількості ітерацій є нижчою за рівень апаратних помилок. Доведено доцільність використання алгоритму Маурера у високобезпечних системах завдяки отриманню доказово простих чисел.

Описано архітектуру генератора випадкових чисел для endpoint-терміналів та показано відмінності між моделями тонкого і товстого клієнта. Уточнено, що у тонкому

клієнті генерація великих простих переноситься на сервер, тоді як товстий клієнт може повністю реалізовувати криптомодуль локально.

Отримані результати підтверджують, що коректно побудований генератор ПВЧ та грамотно організована система тестування простоти є критично важливими елементами гібридних криптосистем і забезпечують необхідний рівень криптографічної стійкості.