НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. Ігоря СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ №2

З дисципліни «Методи реалізації криптографічних механізмів»

«РЕАЛІЗАЦІЯ АЛГОРИТМІВ ГЕНЕРАЦІЇ КЛЮЧІВ ГІБРИДНИХ КРИПТОСИСТЕМ »

Виконала:
студентка групи ФІ-52мн
Балацька В. В.

КИЇВ – 2025

Мета роботи: Дослідження алгоритмів генерації псевдовипадкових послідовностей, тестування простоти чисел та генерації простих чисел з точки зору їх ефективності за часом та можливості використання для генерації ключів асиметричних криптосистем.
Завдання: 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-Штрассена та Міллера-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. Підгрупа 1А. Запропонувати схему генератора ПВЧ для інтелектуальної картки, токена та смартфону. Розглянути особливості побудови генератора простих чисел в умовах обмеження пам’яті та часу генерації.

Хід роботи

Для виконання даної лабораторної роботи було розроблено наступний план:

1. Теоретичні основи ГПВЧ
    a. Класи генераторів
        i. Некриптографічні: LCG, Mersenne Twister (чому не придатні для криптографії)
        ii. Криптографічні DRBG: CTR-DRBG (AES), HMAC-DRBG, Hash-DRBG, потоки на основі ChaCha/AES-CTR.
        iii. Апаратні джерела ентропії (огляд) та роль кондиціювання.
    b. Вимоги до ГПВЧ
        i. Непередбачуваність, стійкість до компрометації, reseed, forward/backward security.
        ii. Моделі безпеки та типові помилки (сіди, повторне використання, низька ентропія).
    c. Метрики оцінки (теоретично)
        i. Ентропія на біт (концептуально).
        ii. Перевірки якості (NIST SP 800-22 — лише як контекст, без практики).
        iii. Асимптотична вартість (ключові операції на біт виходу).
2. Тести простоти: класи та властивості
    a. Імовірнісні
        i. Ферма: ідея, псевдопрості, чому помилкові «свідки» трапляються.
        ii. Соловей–Штрассен: використання символу Якобі, ймовірність помилки.
        iii. Міллер–Рабін: поняття сильного псевдопростого, залежність похибки від числа раундів k, відомі детерміновані бази для обмежених діапазонів.

b. Детерміновані/гіпотетичні
    i. AKS: поліноміальність vs практичність.
    ii. Детермінізований Міллер–Рабін для чисел до певних меж (огляд результатів).
c. Порівняння
    i. Таблиця «складність перевірки» vs «ймовірність помилки».
    ii. Рекомендації для 1024/2048/4096 біт: скільки раундів Міллера–Рабіна теоретично достатньо для заданої цілі похибки.

3. Генерація простих чисел
    a. Підхід “Чебишова”
        i. Ідея через щільність простих ($\pi(x) \sim x/\log x$), вибір інтервалів і очікувана кількість перевірок.
        ii. Теоретична оцінка очікуваної кількості тестів простоти до успіху.
    b. Алгоритм Маурера
        i. Рекурсивна побудова простих та сертифікати простоти.
        ii. Оцінки складності; переваги на великих бітностях.
    c. Порівняльний аналіз
        i. Очікувана кількість кандидатів, вартість одного тесту, сумарна теоретична складність.
        ii. Пам’ять і структурні вимоги.
4. Зв’язок із генерацією ключів гібридних криптосистем
    a. Вимоги до розмірів ключів (1024/2048/4096 біт) з точки зору безпеки.
    b. Вплив якості ГПВЧ на стійкість секретних ключів і параметрів (наприклад, RSA p,q).
    c. Теоретичні ризики при зниженій ентропії або недостатній кількості раундів тесту простоти.
5. Порівняння ймовірнісних тестів за похибкою
    a. Формули/верхні межі похибок для Ферма, Соловей–Штрассен, Міллер–Рабін.
    b. Порівняння при різних k (кількість ітерацій) для цілей: 2^{-64}, 2^{-128}, 2^{-256}.
    c. Обговорення «похибки ПЕОМ» (числова стабільність, переповнення, модульна арифметика з довгою точністю як ідеалізована модель).
6. Обмежені середовища
    a. Схеми ГПВЧ
        i. Смарткарта/токен: мінімальний стан, апаратне джерело шуму + CTR-DRBG; економія пам’яті; енергобюджет.
        ii. Смартфон: поєднання кількох джерел ентропії ОС + ChaCha20/AES-CTR DRBG; періодичний reseed.
    b. Генерація простих при обмеженнях
        i. Вибір алгоритму: Міллер–Рабін з малим k + відсіювання малими простими (wheel factoring).

ii. Компроміс точність/швидкість: мінімальні безпечні k для 1024/2048/4096 біт.
iii. Стратегії економії пам’яті для Маурера (чи доцільний він у вбудованих умовах).

7. Проєктування теоретичної «бібліотеки»
   a. Модулі й інтерфейси:
      i. PRNG: інтерфейс, джерела сіду, reseed-політика.
      ii. Primality: API для Ферма, Соловей–Штрассен, Міллер–Рабін, (AKS — довідково).
      iii. PrimeGen: «Чебишов», Маурер; параметри цільової бітності, політика вибору k.
   b. Алгоритмічні інваріанти: коректність модульної арифметики, контроль похибки, відмова при нестачі ентропії
   c. Конфігураційні профілі: desktop/server vs. smartcard/token vs. smartphone.
8. Аналітична частина
   a. Асимптотики: вартість modular exponentiation, складність одного раунду Міллера–Рабіна, очікувана кількість кандидатів для заданої бітності.
   b. Узагальнені таблиці:
      i. «Бітність → очікувана кількість кандидатів»
      ii. «k → верхня межа похибки»
      iii. «Алгоритм → пам’ять/операції/теоретичний час».
9. Ризики, обмеження і припущення
   a. Модель «ідеальної довгої арифметики» vs апаратні обмеження.
   b. Якість ентропії як критичний фактор.
   c. Теоретичні межі порівнянь без емпіричних вимірів.
10. Висновки та рекомендації
   a. Рекомендовані поєднання:
      i. Server/Desktop: CTR-DRBG/ChaCha20-DRBG + Міллер–Рабін (k, що дає $\leq 2^{-128}$) + Маурер для сертифікованих простих.
      ii. Смарткарта/Токен: легкий DRBG + Міллер–Рабін з мінімальним безпечним k, відсіювання малими простими; «Чебишов» як практичніший за Маурера.
      iii. Смартфон: системний DRBG (Android/iOS) + Міллер–Рабін; за потреби — комбінований reseed.

1) Теоретичні основи ГПВЧ

Сучасні криптографічні протоколи та гібридні криптосистеми критично залежать від якості випадкових чисел. Випадковість використовується для формування ключів, ініціалізаційних векторів, сольових значень та інших параметрів, що впливають на безпеку всієї системи. Оскільки апаратні джерела істинної випадковості обмежені та повільні, у більшості випадків застосовуються генератори псевдовипадкових чисел (ГПВЧ) — алгоритми, що з використанням початкового випадкового сіду здатні відтворювати довгі непередбачувані послідовності бітів. Для криптографії важливо, щоб ці послідовності були максимально непередбачуваними, мали високу ентропію та були стійкими до атак, що намагаються відновити внутрішній стан генератора.

1.1 Класи генераторів

Некриптографічні генератори

До найвідоміших належать лінійний конгруентний генератор (LCG) та Mersenne Twister.

- LCG працює за простою формулою $X_{n+1}=(aX_n+c) \bmod m$, що робить його швидким та зручним для симуляцій.
- Mersenne Twister забезпечує дуже довгий період ($2^{19937}-1$) і хорошу статистичну рівномірність, що робить його популярним у наукових розрахунках.

Однак обидва підходи непридатні для криптографії, оскільки:

- Внутрішній стан легко відновити, якщо спостерігати обмежену кількість вихідних значень (для LCG достатньо кількох послідовних чисел, для Mersenne Twister — 624 32-бітних значень).
- Генератори не забезпечують захист від атак на прогнозування майбутніх бітів.
- Відсутня властивість forward/backward security: компрометація поточного стану дозволяє відновити як майбутні, так і минулі значення.

Отже, некриптографічні генератори можуть підходити для моделювання або статистики, але категорично не повинні використовуватися для побудови ключів.

Криптографічні DRBG (Deterministic Random Bit Generators)

Криптографічні генератори проектуються на основі надійних примітивів — блочних шифрів та геш-функцій. Основні стандартизовані варіанти (згідно з NIST SP 800-90A):

- CTR-DRBG: використовує блочний шифр AES у режимі лічильника (CTR). Стан генератора періодично оновлюється за допомогою ключа AES, а вихідні біти утворюються через шифрування лічильника.
- HMAC-DRBG: базується на механізмі HMAC із криптографічними геш-функціями (SHA-256/512). Забезпечує строгі докази безпеки у моделі псевдовипадкових функцій.
- Hash-DRBG: використовує геш-функцію напряму для розширення сіду; підходить для платформ без AES.

- Потокові генератори на основі ChaCha20 та AES-CTR: дуже швидкі й одночасно стійкі до криптоаналізу, часто використовуються у сучасних бібліотеках (наприклад, в OpenSSL та Linux kernel).

Ключова відмінність таких DRBG — формальна модель безпеки: навіть якщо частина виходу або стан компрометовані, атакувальник не може передбачити попередні чи наступні значення, за умови правильного керування сідом та періодичного reseed.

Апаратні джерела ентропії та кондиціювання

У багатьох платформах використовуються апаратні джерела випадковості — наприклад, флуктуації температури, шум електронних схем, джиттер тактових генераторів. Відомий приклад — Intel Digital Random Number Generator (DRNG), який реалізує інструкції RDRAND та RDSEED.

Проте «сирі» апаратні бітові потоки не завжди рівномірні та можуть мати кореляції. Для підвищення якості випадковості застосовують кондиціювання — криптографічні хеш-функції, шифри або спеціальні конденсатори ентропії, які «вирівнюють» розподіл та забезпечують високу непередбачуваність. У сучасних системах зазвичай комбінують апаратний шум і програмний DRBG.

1.2 Вимоги до ГПВЧ

Непередбачуваність та стійкість до компрометації

Генератор повинен бути непередбачуваним навіть для зловмисника, що знає частину його виходу. Ключові вимоги:

- Forward security — компрометація поточного стану не дає відновити попередні вихідні біти.
- Backward security — компрометація стану не дає передбачити майбутні значення.
- Reseed — можливість безпечно оновлювати стан новою ентропією без повної ініціалізації.

Ці властивості запобігають атакам, коли атакувальник частково відновив стан (наприклад, через помилки в реалізації).

Моделі безпеки та типові помилки

До типових помилок належать:

- Використання передбачуваних сідів (час системи, MAC-адреса, PID процесу).
- Повторне використання одного сіду на різних пристроях.
- Низька ентропія під час ініціалізації, особливо на вбудованих платформах (смарткарти, IoT), де відсутні якісні джерела шуму.
- Відсутність захисту стану (зберігання в незахищеній пам’яті, можливість відкату).

1.3 Метрики оцінки

Ентропія на біт

Ентропія — міра невизначеності, яку містить випадкова змінна. Ідеальний генератор має 1 біт ентропії на кожен вихідний біт, тобто кожен біт непередбачуваний. Зниження ентропії означає зменшення простору можливих ключів та потенційну можливість атаки перебором.

Перевірки якості

Для оцінки випадковості використовують статистичні тести, наприклад, пакет NIST SP 800-22 (Frequency Test, Runs Test, Serial Test тощо). Однак навіть успішне проходження тестів не гарантує криптографічної безпеки, бо вони лише виявляють очевидні регулярності. Тому тести розглядають як допоміжний інструмент для виявлення грубих помилок, але не як доказ стійкості.

Асимптотична вартість

Оцінюючи ГПВЧ, важливо враховувати кількість криптографічних операцій на кожен біт виходу. Некриптографічні генератори мають O(1) із дуже малою константою, але небезпечні. Криптографічні DRBG також працюють за O(1), однак із «важчою» операцією — AES, HMAC або SHA. Для пристроїв із низькими ресурсами вибір алгоритму визначається компромісом між безпекою, швидкістю та енергоспоживанням.

2) Тести простоти: класи та властивості

Перевірка чисел на простоту є ключовим етапом у генерації криптографічних ключів, зокрема для алгоритмів типу RSA чи ElGamal. Простота великих чисел (1024–4096 біт) не перевіряється прямим діленням через надзвичайну складність, тому застосовують алгоритми, що поєднують ефективність і прийнятну ймовірність похибки. Сучасні тести поділяють на імовірнісні (швидкі, але з малою ймовірністю помилки) та детерміновані (абсолютно точні, проте значно повільніші).

2.1 Імовірнісні тести

Тест Ферма

Ідея ґрунтується на малій теоремі Ферма: якщо p — просте число, то для будь-якого aaa, що не ділиться на p, a^(p−1)≡1(modp).

Тест працює так: вибирають випадкове aaa та перевіряють, чи виконується рівність. Якщо ні — число точно складене. Якщо так — воно може бути простим або псевдопростим (числом, яке задовольняє умову для певних aaa, але не є простим).

Недоліки:

- Існують псевдопрості числа, що проходять тест для багатьох основ.
- Існують числа Кармайкла, які проходять перевірку Ферма для всіх aaa, взаємно простих із числом.

Отже, тест Ферма дає надто велику ймовірність помилки й не використовується самостійно у криптографії.

Тест Соловея–Штрассена

Поліпшений варіант, який використовує символ Якобі. Для непарного числа nnn та випадкового aaa перевіряється:

a^(n−1)/2≡J(a,n)(modn)

де J(a,n) — символ Якобі. Якщо рівність не виконується — число складене; якщо виконується — воно, імовірно, просте.

Ймовірність помилки: не більше 1/2 на один раунд. Тобто після kkk незалежних перевірок ймовірність помилки ≤ 2^(−k). Це значно краще за тест Ферма.

Тест Міллера–Рабіна

Один із найпоширеніших імовірнісних тестів, який базується на понятті сильного псевдопростого.

Алгоритм:

1. Подати n−1=2^sd (де d непарне).
2. Вибрати випадкове a та перевірити:
   a^d≡1(modn)або a^(2^rd)≡−1(modn)для деякого 0≤r<s

Якщо жодна умова не виконується — число складене.

Ймовірність помилки для одного раунду не перевищує 1/4. Після k раундів похибка ≤ 4^(−k). На практиці для 1024-бітних чисел достатньо 5–7 раундів для похибки < 2^(−100).

Важливий факт: для чисел до певних меж відомі детерміновані набори баз (напр., для 32-бітних і 64-бітних чисел існують фіксовані множини основ, які гарантують правильний результат без випадковості).

2.2 Детерміновані / гіпотетичні тести

AKS (Agrawal–Kayal–Saxena)

AKS — перший доведений поліноміальний тест простоти. Алгоритм ґрунтується на властивостях многочленів та перевіряє рівність:

(x+a)^n≡x^n+a(modn)

у спеціально підібраному кільці. Складність алгоритму поліноміальна за O(log^6n), але велика прихована константа робить його непридатним для практичного використання при 1024+ бітах.

Детермінований Міллер–Рабін

Доведено, що для чисел менше деяких меж достатньо перевірити лише фіксовану множину основ aaa. Наприклад:

- для n<2^32 достатньо баз {2,7,61};

- для n<2^64 — відомо набір із 12 фіксованих основ.

Для більших чисел такий підхід не доведений, тому тест Міллера–Рабіна використовується як імовірнісний із випадковим вибором основ.

2.3 Порівняння

Складність та ймовірність помилки

Тест	Складність (один раунд)	Ймовірність помилки за раунд	Характеристика
Ферма	O(logf_0^3n)	висока, числа Кармайкла	Дуже швидкий, але ненадійний
Соловей–Штрассен	O(logf_0^3n)	≤ 1/2	Швидкий, кращий за Ферма
Міллер–Рабін	O(logf_0^3n)	≤ 1/4	Найпопулярніший, контроль похибки через k
AKS	O(logf_0^6n)	0	Теоретично точний, але повільний

Рекомендації щодо кількості раундів Міллера–Рабіна

Для практичних цілей (RSA, ElGamal) рекомендують:

Бітність n	Рекомендовані раунди k	Похибка ≤
1024	5–7	2^(−100) 2^(−140)
2048	7–10	2^(−200) 2^(−140)
4096	10–12	2^(−200) 2^(−240)

Такі параметри забезпечують практично нульову ймовірність помилки для сучасних застосувань.

Висновок

Імовірнісні тести, особливо Міллера–Рабіна, є оптимальними для генерації простих чисел великої розрядності завдяки поєднанню високої швидкодії та керованої ймовірності похибки. Тест Ферма непридатний для безпеки. Соловей–Штрассен є проміжним варіантом, але витіснений Міллером–Рабіним через кращу ефективність. Детерміновані методи (AKS) залишаються важливими з теоретичної точки зору, проте практично не застосовуються для великих чисел.

3) Генерація простих чисел

Генерація великих простих чисел є критичною для асиметричних криптосистем (RSA, DH, ElGamal). Основна задача полягає у знаходженні простого числа заданої довжини (1024–4096 біт), яке можна використати як модуль чи основу для криптографічних операцій. Через велику розрядність неможливо перебирати всі варіанти, тому застосовують ймовірнісні тести простоти в поєднанні з ефективними алгоритмами побудови простих.

3.1 Підхід “Чебишова”

Ідея та оцінка щільності простих

Основою є асимптотичне наближення функції простих чисел π(x):

π(x)~x/lnx.

Звідси випливає, що ймовірність випадково вибраного великого непарного числа бути простим приблизно дорівнює 1/lnx

Наприклад, для 1024-бітного числа (x≈2^1024) ця ймовірність ~ 1/ln(2^1024).

Це означає, що в середньому потрібно протестувати ~700 випадкових непарних кандидатів, щоб знайти простий.

При використанні сучасних імовірнісних тестів (Міллер–Рабін, 5–10 раундів) пошук простого стає ефективним: кожен тест має складність O(log^3n), а кількість спроб масштабується як ln(n).

Теоретична оцінка кількості перевірок

Очікувана кількість тестів простоти E[T] для знаходження простого числа довжиною n біт:

E[T]≈ln2^n=n·ln2

Для 1024 біт отримаємо близько 710 перевірок, для 2048 біт — ~1419, для 4096 біт — ~2838.

Це значення відповідає середній кількості ітерацій при випадковому виборі кандидатів.

3.2 Алгоритм Маурера

Рекурсивна побудова простих та сертифікати простоти

Алгоритм Маурера (Maurer’s Algorithm) — це генератор простих із сертифікацією, який

забезпечує не лише знаходження простого, але й надає доказ (сертифікат), що отримане число дійсно просте.

Ідея:

1. Рекурсивно генеруються менші прості числа для побудови факторизації n−1.
2. Використовується теорема: якщо існує простий дільник q|n−1, що задовольняє певні умови, та виконується рівність a^(n−1)≡1(modn), то n — просте.
3. Процес будує ланцюг сертифікатів простоти, які можна перевірити за поліноміальний час

Оцінки складності та переваги

- Теоретична складність — майже лінійна в довжині числа: O(log^4n).
- Гарантує абсолютну простоту (не імовірну, а доведену).
- Добре масштабується для великих бітностей (2048+), проте повільніше за простий випадковий пошук із тестами Міллера–Рабіна.
- Використовується, коли потрібна формальна сертифікація простоти (наприклад, для високозахищених ключів або перевірки надійності ГПВЧ).

3.3 Порівняльний аналіз

Очікувана кількість кандидатів і сумарна складність

Метод	Очікувана кількість кандидатів	Складність одного тесту	Сумарна теоретична складність
Підхід Чебишова + Міллер–Рабін	~ ln2^n=nln2	O(log^3n)	O(n·log^3n)
Алгоритм Маурера	~ logn рекурсивних рівнів	O(log^3n)	O(log^4n)

- Метод Чебишова простіший для реалізації та швидший при практичній генерації (особливо з паралельним тестуванням).
- Алгоритм Маурера повільніший, але надає сертифікат простоти, що робить його корисним для критично безпечних систем.

Вимоги до пам’яті та структури

- Чебишов + Міллер–Рабін потребують мінімальної пам’яті: лише збереження поточного кандидата та параметрів тесту.
- Алгоритм Маурера вимагає більше пам’яті для збереження рекурсивних сертифікатів та проміжних простих дільників n−1.

- Для пристроїв із обмеженою пам’яттю (смарт-картки, токени) практичнішим є простий підхід Чебишова.

Висновок

Для більшості прикладних криптосистем достатньо випадкового вибору непарних чисел із наступною перевіркою тестами Міллера–Рабіна. Такий підхід є швидким і практичним навіть для 4096-бітних ключів. Алгоритм Маурера застосовують там, де потрібне формальне підтвердження простоти, хоча його вартість за часом і пам’яттю вища.

4) Зв’язок із генерацією ключів гібридних криптосистем

Гібридні криптосистеми поєднують швидкодію симетричних алгоритмів (AES, ChaCha20) із безпекою асиметричних (RSA, ECC, постквантових схем). У таких системах генерація великих простих чисел та використання надійних генераторів псевдовипадкових чисел безпосередньо визначають стійкість ключової інфраструктури. Помилки на цьому етапі призводять до катастрофічних компрометацій навіть при формально сильних алгоритмах шифрування.

4.1 Вимоги до розмірів ключів

Довжина ключів визначається поточними та прогнозованими можливостями атакуючих обчислювальних систем.

- RSA:
  - 1024 біт вважається мінімально безпечним лише для короткочасних задач, але більше не рекомендований.
  - 2048 біт — стандартна довжина для сучасних систем із запасом безпеки на 10–15 років.
  - 3072/4096 біт — рекомендується для довгострокової безпеки та протидії майбутнім покращенням факторизаційних алгоритмів.
- Гібридні схеми (RSA + симетрія):
  Використання 2048-бітних модулів із симетричними ключами AES-128 вважається достатнім для більшості застосувань. Для довготривалої конфіденційності рекомендується 3072–4096 біт (еквівалентно AES-192/256).
- Постквантовий контекст:
  При появі квантових атак (алгоритм Шора) навіть 4096-бітний RSA не гарантує безпеки, тому сучасні гібридні системи комбінують RSA/ECC із постквантовими алгоритмами (Kyber, BIKE, NTRU).

4.2 Вплив якості ГПВЧ на стійкість ключів і параметрів

Генератор псевдовипадкових чисел є фундаментальним джерелом секретності. Якщо ГПВЧ має низьку ентропію або передбачуваний стан:

- Компрометація RSA:

Якщо прості числа p,q у RSA згенеровані з недостатньо випадковими бітами, можливий partial key exposure attack — факторизація модулів із частковим знанням p або q. Відомі атаки на слабкі ГПВЧ (наприклад, Debian OpenSSL 2008): ключі були відновлені через передбачуваний сід.

- Відновлення симетричних ключів:
  Низька ентропія при генерації ключа для AES дозволяє повний перебір набагато швидше, ніж планувалося.
- Forward/backward security:
  Якщо ГПВЧ не пересідається (reseed) і атакуючий отримав доступ до внутрішнього стану, він може відновити як минулі, так і майбутні ключі.

4.3 Теоретичні ризики при зниженій ентропії або недостатній кількості раундів тесту простоти

Знижена ентропія сіду:

- Ймовірність колізій у вибраних простих значно зростає.
- Можливі атаки на RSA через пошук спільних дільників між модулями (GCD-атака). Відомі випадки (2012 рік), коли ~0,5% публічних ключів RSA мали спільний простий множник через поганий ГПВЧ.

Недостатня кількість раундів Міллера–Рабіна:

- Імовірність пропустити складене число (псевдопросте) зростає.
- Якщо таке число використати як ppp чи qqq у RSA, безпека зникає: модуль стає легко факторизованим.
- Для 1024-бітних чисел при 1–2 раундах імовірність помилки може сягати 2^(−2)-2^(-4), що критично високо.

Каскадні наслідки:

- Ненадійні ключі компрометують усю гібридну схему — навіть при сильному симетричному шифрі.
- Підвищується ризик масових атак: зловмисник може перевіряти велику кількість відкритих ключів на слабкість (як це робив сканер “RsaCtfTool”).

Висновок

Безпечна генерація ключів для гібридних криптосистем критично залежить від якісного ГПВЧ та правильного вибору параметрів тестування простоти. Використання надійних криптографічних DRBG (наприклад, CTR-DRBG або HMAC-DRBG), регулярний reseed, достатня кількість раундів Міллера–Рабіна та дотримання сучасних рекомендацій щодо розмірів ключів (2048+ біт для RSA) — обов’язкові умови для захисту системи.

5) Порівняння ймовірнісних тестів за похибкою

Ймовірнісні тести простоти — ключовий інструмент для генерації великих простих чисел у криптографії (RSA, Diffie–Hellman, ElGamal). Вони не гарантують 100%

точності, але дають контрольовану ймовірність помилки. Розуміння верхніх меж похибок та залежності від кількості раундів дозволяє правильно налаштувати безпеку на практиці.

5.1 Формули та верхні межі похибок

Тест Ферма

- Ідея: якщо p — просте, то для будь-якого aaa, взаємно простого з p, виконується
$$a^{p-1} \equiv 1 (mod p).$$
- Псевдопрості: складені числа, що задовольняють умову (наприклад, числа Кармайкла).
- Верхня межа похибки: для випадкового aaa складене число може пройти тест із імовірністю до 1 (тобто тест Ферма ненадійний без додаткових перевірок).

Тест Соловея–Штрассена

- Використовує символ Якобі:
$$a^{(n-1)/2} \equiv (n/a) (mod n).$$
- Якщо n складене, то щонайменше половина можливих свідків a його «викривають»
- Імовірність помилки за один раунд — не більше 1/2. Для k раундів:
$$P_{помилки} \leq 2^{(-k)}.$$

Тест Міллера–Рабіна

- Найбільш поширений у криптографії. Ґрунтується на представленні $n-1=2^s d$ (із непарним d).
- Для кожного випадкового aaa складене число стає сильним псевдопростим з імовірністю не більшою ніж 1/4.
- Імовірність помилки після k незалежних раундів:
$$P_{помилки} \leq 4^{(-k)} = 2^{(-2k)}.$$

5.2 Порівняння при різних k (кількість ітерацій)

Зручно оцінювати, скільки раундів потрібно для досягнення певного рівня безпеки (цільова похибка):

Тест	Помилка за 1 раунд	Формула для k	k для 2^{-64}	k для 2^{-128}	k для 2^{-256}
Ферма	≤1	—	— (небезпечно)	—	—

Соловей–Штрассен	≤1/2	2^{-k}	64	128	256
Міллер–Рабін	≤1/4	4^{-k}=2^{-2k}	32	64	128

- Для досягнення ймовірності помилки ≤ 2^{-128} потрібно лише 64 раунди Міллера–Рабіна, тоді як Соловею–Штрассену знадобиться 128.
- Тест Ферма практично не застосовують у криптографії через відсутність гарантованої верхньої межі.

5.3 Обговорення «похибки ПЕОМ» (числові похибки та моделювання)

У реальних системах обчислення виконуються на ПЕОМ (персональних ЕОМ) із обмеженою точністю та специфікою роботи процесорів. Це створює додаткові ризики:

1. Числова стабільність та переповнення
   - При обчисленнях a^{d}mod n застосовують методи швидкого піднесення до степеня.
   - Якщо реалізація не враховує переповнення при проміжних множеннях (особливо в мовах без автоматичного керування довгими цілими), можливі неправильні залишки та хибні результати тесту простоти.
2. Модульна арифметика великої точності
   - Алгоритми передбачають ідеалізовану арифметику без втрат точності.
   - У реальності потрібно застосовувати бібліотеки довгої арифметики (GMP, OpenSSL BN).
   - Використання звичайних 64-бітних типів може призвести до помилкових свідків через переповнення.
3. Сторонні ефекти оптимізацій
   - Компілятори можуть оптимізувати множення/редукцію модулю, порушуючи математичну коректність.
   - Наприклад, неправильна реалізація Montgomery reduction або Barrett reduction здатна дати «помилкове підтвердження простоти».
4. Ідеалізована модель vs реальні ПЕОМ
   - Теоретично вважається, що ми маємо нескінченну точність та безпомилкові операції.
   - Практично потрібно враховувати:
     - довжину машинного слова,
     - ризики переповнення,
     - апаратне прискорення (наприклад, використання інструкцій MULX на x86-64).

Висновок

- Міллер–Рабін є найефективнішим і вимагає найменше раундів для досягнення заданої ймовірності помилки.
- Соловей–Штрассен простіший за доказами, але менш ефективний у плані кількості перевірок.
- Ферма — непридатний для криптографії без додаткових механізмів (через Кармайклові числа).
- У реальній реалізації потрібно враховувати апаратні та програмні похибки: переповнення, нестабільність модульної арифметики, правильне використання довгої арифметики.
- Рекомендація для безпечних систем: ≥64 раундів Міллера–Рабіна дають похибку ≈ 2{-128}, що відповідає рівню безпеки сучасних асиметричних алгоритмів.

6) Обмежені середовища

У реальних системах генерація простих чисел та ключових параметрів часто відбувається не на потужних серверах, а на пристроях із обмеженими ресурсами — смарткартах, токенах автентифікації, мобільних телефонах, IoT-пристроях. Такі середовища накладають суворі обмеження на пам’ять, енергоспоживання, продуктивність та доступність якісної ентропії. Тому стандартні підходи до тестування простоти та генерації ключів потребують адаптації.

6.1 Схеми ГПВЧ (генераторів псевдовипадкових чисел)

Смарткарта/токен

- Особливості середовища:
  Смарткарти та токени автентифікації мають мінімальний обсяг оперативної пам’яті (кілька десятків кілобайт), слабкий процесор і суворі обмеження на енергоспоживання. Будь-який алгоритм має бути надзвичайно економним та стабільним у виконанні.
- Джерело випадковості:
  Часто використовують апаратне джерело шуму (TRNG), яке забезпечує базову ентропію. Для побудови криптографічно безпечної послідовності застосовують CTR-DRBG (Deterministic Random Bit Generator у режимі лічильника).
- Оптимізації:
  - Мінімальний стан генератора (зберігаються лише ключ і лічильник для AES або інша компактна структура).
  - Економія пам’яті за рахунок відсутності великих буферів та кешування.
  - Енергобюджет: важливо мінімізувати кількість викликів апаратного TRNG (отримання ентропії дороге за часом та енергією), тому роблять періодичний *reseeding*.

Смартфон

- Особливості середовища:
  Смартфони потужніші за смарткарти, але все одно мають обмеження енергії (батарея), а також проблеми з надійністю ентропії на старті системи.
- Джерела випадковості:
  - Поєднання кількох ентропійних джерел ОС: сенсорні дані, рух, радіомодуль, шум мікрофона.
  - Системні DRBG: AES-CTR DRBG, ChaCha20 DRBG (ChaCha20 часто переважає через ефективність на ARM-процесорах).
  - Регулярний *reseed* для підтримання непередбачуваності.
- Оптимізації:
  Можливе використання апаратних модулів безпеки (Secure Enclave, TEE) для ізоляції генератора та підвищення надійності.

6.2 Генерація простих при обмеженнях

Вибір алгоритму

- Міллер–Рабін із малим k — популярний вибір для пристроїв із обмеженими ресурсами.
  Зазвичай роблять 8–16 раундів, що дає помилку ≈2^{-16… 32}, достатню для багатьох практичних застосувань.

- Відсіювання малими простими (wheel factoring):
  Перед запуском тесту простоти число перевіряють на подільність невеликими простими (наприклад, до 1000). Це значно зменшує кількість кандидатів, що проходять до складнішої перевірки.

Компроміс точність/швидкість

- Для великих ключів (1024/2048/4096 біт) кількість раундів kkk визначають, виходячи з компромісу між безпекою та швидкодією:
  - 1024 біт — часто k=8…16.
  - 2048 біт — рекомендовано k≥32.
  - 4096 біт — іноді k=64k=64k=64 для ймовірності помилки близько 2^{-128}.
- У пристроях із дуже обмеженими ресурсами іноді зменшують k і покладаються на додаткові фільтри (відсіювання малими простими, додаткові перевірки).

Стратегії економії пам’яті для Майєра

- Алгоритм Майєра (Maurer) дає детерміновані сертифікати простоти, але потребує збереження попередніх простих та виконання рекурсивних перевірок.
- У пристроях із низькою пам’яттю:

    - Часткове збереження лише найважливіших проміжних результатів.
    - Потокове обчислення без повного кешу факторизації.
    - Використання невеликих таблиць простих чисел (замість повних списків).
- Практичний висновок: на більшості смарткарт Майєр не доцільний, оскільки вимагає значних обсягів пам'яті й складніший у реалізації. Тут майже завжди використовують ймовірнісні тести (Міллер–Рабін).

Висновки

- У смарткартах і токенах критичними є економія пам'яті, мінімізація викликів TRNG та енергоспоживання.
- На смартфонах можна використовувати більш складні DRBG (ChaCha20/AES-CTR), але треба враховувати надійність початкової ентропії.
- Міллер–Рабін із попереднім відсіюванням малими простими — стандарт де-факто для обмежених середовищ.
- Майєр теоретично кращий для отримання сертифікатів простоти, але майже не застосовується в малопотужних пристроях через складність і потребу в пам'яті.
- Головний принцип: адаптувати кількість раундів k і оптимізувати обчислення під конкретні обмеження апаратного середовища.

7) Проєктування теоретичної «бібліотеки»

У практичній криптографії часто виникає потреба в універсальній бібліотеці для генерації простих чисел і перевірки їхньої простоти. Такий інструмент повинен бути гнучким, безпечним та масштабованим: однаково працювати на потужних серверах, настільних системах та у середовищах із обмеженими ресурсами (смарткарти, мобільні пристрої).

Проектування подібної бібліотеки потребує чіткої модульної архітектури, забезпечення математичної коректності, захисту від помилок та контрольованого використання ентропії.

7.1 Модулі й інтерфейси

Бібліотека повинна складатися з чітко розділених модулів, кожен із яких відповідає за окремий етап процесу: генерація випадкових чисел, перевірка простоти, побудова простих певної бітності.

PRNG (Генератор псевдовипадкових чисел)

- Інтерфейс

  Єдина точка доступу для ініціалізації та отримання випадкових бітів. Має дозволяти підключення різних джерел випадковості (апаратних і програмних).
- Джерела сіду
    - Апаратура (TRNG).

    - Системні джерела ОС (наприклад /dev/urandom у Linux або CryptGenRandom у Windows).
    - Зовнішній ентропійний модуль (HSM, Secure Element).
- Reseed-політика
    - Регулярне оновлення початкового стану для підтримки непередбачуваності.
    - Параметри періодичності залежно від середовища: частіший *reseeding* у довготривалих сесіях сервера, рідший — на смарткарті з обмеженим TRNG.

Primality (API для перевірки простоти)

- Загальний інтерфейс:
  Надати функції is_prime(n, k) або розширені варіанти з параметрами безпеки.
- Реалізовані тести:
    - Ферма — як базовий швидкий, але ненадійний тест.
    - Соловей–Штрассен — з використанням символу Якобі.
    - Міллер–Рабін — основний ймовірнісний тест із керованою кількістю раундів k.
    - AKS — детермінований, використовується як довідковий для порівняння (через непрактичність на великих бітностях).
- Політика вибору тесту:
    - Для серверів — Міллер–Рабін із достатньою кількістю раундів.
    - Для пристроїв із обмеженнями — Міллер–Рабін із малим k + відсіювання малими простими.
    - Для наукових або довірених середовищ — AKS як еталон перевірки.

PrimeGen (Генерація простих)

- Методи:
    - Підхід «Чебишова»: генерація випадкових чисел у заданому діапазоні, оцінка ймовірності простоти за щільністю $\pi(x) \sim x/\log x$.
    - Алгоритм Майєра: рекурсивне побудування простого числа з сертифікатом простоти.
- Параметри:
    - Цільова бітність (1024/2048/4096).
    - Політика вибору k для тестів Міллера–Рабіна залежно від вимог безпеки та швидкодії.
    - Можливість встановлення обмежень пам’яті та часу (корисно для мобільних пристроїв і смарткарт).

7.2 Алгоритмічні інваріанти

Для забезпечення надійності бібліотека повинна контролювати коректність і безпеку обчислень на кожному етапі.

- Коректність модульної арифметики
  Виконання операцій за модулем великих чисел має бути стабільним навіть при обмеженій точності обчислень. Необхідна перевірка переповнень та контроль правильності редукції.
- Контроль похибки
  - Усі ймовірнісні тести повинні повертати оцінку верхньої межі ймовірності помилки (наприклад, 2^{-k} для Міллера–Рабіна).
  - Додатковий контроль стабільності (уникнення помилок через апаратні/числові обмеження).
- Відмова при нестачі ентропії
  Якщо PRNG не може зібрати достатньо випадкових бітів для безпечної генерації, бібліотека повинна сигналізувати помилку, а не генерувати передбачувані ключі.

7.3 Конфігураційні профілі

Одна з ключових переваг бібліотеки — можливість адаптувати роботу до різних апаратних платформ.

- Desktop/Server
  - Необмежений доступ до пам’яті й швидких PRNG (наприклад, /dev/urandom).
  - Використання потужних алгоритмів (Міллер–Рабін із великим k, AKS для перевірки).
  - Оптимізація на швидкодію, менші обмеження щодо енергоспоживання.
- Smartcard/Token
  - Сувора економія пам’яті та енергії.
  - Легка версія Міллера–Рабіна із малим k та попереднім відсіюванням.
  - Простий PRNG із мінімальним станом (CTR-DRBG).
  - Чітка політика *reseeding*, мінімізація запитів до апаратного TRNG.
- Smartphone
  - Компроміс між продуктивністю та безпекою.
  - Можливість використання ChaCha20/AES-CTR DRBG.
  - Динамічний вибір кількості раундів Міллера–Рабіна залежно від потужності пристрою та потреби в безпеці.
  - Додатковий *reseed* при довготривалих операціях (наприклад, генерація ключів у фоновому режимі).

Висновки

- Архітектура повинна бути модульною: PRNG, тести простоти, генерація простих.

- Гнучкість налаштування дозволяє застосовувати одну бібліотеку в різних середовищах — від потужних серверів до смарткарт.
- Безпека забезпечується за рахунок інваріантів: перевірка коректності модульної арифметики, контроль ентропії, прогнозована ймовірність помилки.
- Такий підхід дозволяє створити універсальний інструмент для досліджень, прототипування та практичного впровадження криптографічних алгоритмів.

8) Аналітична частина

Аналітична частина спрямована на кількісне оцінювання ефективності алгоритмів генерації простих чисел та тестування простоти з урахуванням бітності чисел, вартості основних арифметичних операцій та вірогідності похибки. Цей розділ дозволяє порівняти різні підходи не лише концептуально, але й за конкретними показниками, важливими для практичного застосування в криптосистемах.

8.1 Асимптотики та ключові витрати

1. Вартість modular exponentiation

- Модульне піднесення до степеня (modular exponentiation) є базовою операцією у більшості тестів простоти, зокрема Міллера–Рабіна та Ферма.
- Для числа довжини nnn біт класична реалізація за методом square-and-multiply має складність:
  O(log⁡e·M(n)), де M(n) — вартість множення двох nnn-бітних чисел.
- При використанні швидких алгоритмів множення (наприклад, Карацуби O(n^{log_2 3}), або FFT-множення O(nlog⁡n) вартість зменшується, але для практичних розмірів (1024–4096 біт) часто використовують оптимізоване класичне множення.
- Висновок: час однієї modular exponentiation для 2048-бітного числа приблизно відповідає кільком тисячам базових множень, що є ключовим фактором продуктивності тестів простоти.

2. Складність одного раунду Міллера–Рабіна

- Один раунд тесту Міллера–Рабіна складається з:
  1. Представлення n−1=2^s·d, де d непарне.
  2. Випадкового вибору бази a.
  3. Обчислення x=a^d mod n
  4. Послідовного піднесення до квадрату (до s−1 разів).
- Основна обчислювальна вартість — це одна modular exponentiation плюс до s−1 modular squaring.
- Асимптотично один раунд коштує:
  O(log⁡n·M(n)),
  що відповідає кільком modular exponentiation залежно від кількості бітів.

3. Очікувана кількість кандидатів до простоти

- За теоремою про розподіл простих чисел: $\pi(x) \sim x/\ln x$.
- Ймовірність того, що випадкове непарне число довжини nnn біт буде простим: p_n≈1/ln⁡2^n=1/nln⁡2.
- Очікувана кількість перевірок для знаходження простого: E_n≈nln⁡2.

Бітність	**Ймовірність простоти**	**Очікувана кількість перевірок**
1024	≈ 1 / 710	~ 710 кандидатів
2048	≈ 1 / 1420	~ 1420 кандидатів
4096	≈ 1 / 2840	~ 2840 кандидатів

Для 2048-бітного ключа середньо потрібно протестувати ≈1400 випадкових непарних чисел перед знаходженням простого.

8.2 Узагальнені таблиці

Для практичного порівняння різних алгоритмів та налаштувань доцільно скласти узагальнені таблиці.

Таблиця «Бітність → очікувана кількість кандидатів»

Бітність (n)	**Ймовірність простоти (p_n)**	**Очікувана кількість перевірок**
1024	(1/710)	≈710
2048	(1/1420)	≈1420
4096	(1/2840)	≈2840

Таблиця «k → верхня межа похибки» для Міллера–Рабіна

Кількість раундів (k)	Ймовірність похибки ≤
1	$(2^{-2}) \approx 0.25$
5	$(2^{-10}) \approx 9.8 \cdot 10^{-4}$
10	$(2^{-20}) \approx 9.5 \cdot 10^{-7}$
20	$(2^{-40}) \approx 9.1 \cdot 10^{-13}$
40	$(2^{-80}) \approx 8.3 \cdot 10^{-25}$

Для криптографічної безпеки (2048–4096 біт) зазвичай достатньо k = 40, що дає похибку $\approx 2^{-80}$.

Таблиця «Алгоритм → пам'ять / операції / теоретичний час»

Алгоритм	Пам'ять	Операції (асимптотика)	Коментар
Ферма	Дуже мала	O(log n M(n))	Дуже швидкий, але ненадійний
Соловей–Штрассен	Мала	O(\log n M(n))	Краща надійність, але все ще ймовірнісний
Міллер–Рабін	Мала	O(k log nM(n))	Баланс безпеки та швидкості
AKS	Велика	O(poly(n))	Детермінований, але повільний на практиці
Маурер	Середня	Залежить від сертифікації	Оптимальний для побудови простих із доказом

Висновок

- Міллер–Рабін залишається найкращим компромісом для великих ключів (1024–4096 біт) за рахунок поєднання швидкості та низької ймовірності похибки при достатньому числі раундів.
- AKS корисний лише як теоретичний еталон через високу складність на практиці.

- Модульне піднесення до степеня — ключова операція, що визначає загальну швидкодію.
- Для 2048-бітних ключів середньо потрібно протестувати ≈1400 чисел, що добре узгоджується з оцінками щільності простих.

9) Ризики, обмеження і припущення

Будь-яка криптографічна бібліотека, що реалізує генерацію простих чисел і тестування простоти, неминуче стикається з низкою припущень та обмежень, які впливають на надійність, продуктивність та безпеку. Цей розділ описує ключові ризики, що можуть виникати при практичній реалізації, а також підкреслює відмінність між теоретичною моделлю та реальними умовами виконання.

9.1 Модель «ідеальної довгої арифметики» vs апаратні обмеження

- Теоретична модель.
  При аналізі алгоритмів зазвичай робиться припущення, що довгі цілі числа можна додавати, множити й ділити за «ідеальною» складністю M(n), без обмежень пам’яті та часу доступу. Це дозволяє виводити чисті асимптотичні формули.
- Реальні апаратні обмеження.
  - У практичних реалізаціях використання довгої арифметики залежить від типу процесора, наявності інструкцій для роботи з великими числами (наприклад, MULX, ADCX для x86-64) та розміру кешу.
  - Витрати на операції можуть зростати через обмеження пропускної здатності пам’яті, перемикання між різними розмірами слів та внутрішні оптимізації компілятора.
  - Певні пристрої (наприклад, смарткарти чи токени) мають апаратні ліміти — малі обсяги оперативної пам’яті, низьку тактову частоту, що робить виконання великих модульних множень значно повільнішим, ніж на ПК/сервері.

Висновок: розрахунки асимптотичної складності добре описують тренди, але можуть давати надто оптимістичну оцінку швидкості для обмежених пристроїв.

9.2 Якість ентропії як критичний фактор

- Криптографічна безпека залежить від випадковості. Якість генератора випадкових чисел (PRNG) прямо впливає на безпеку: недостатня ентропія може зробити ключі передбачуваними.
- Основні ризики:
  - Використання слабкого початкового сіда (seed), особливо на вбудованих пристроях без достатнього джерела шуму.

    - Повторне використання однакових або схожих сидів (seed reuse) через погану політику reseed.
    - Недостатній моніторинг стану ентропії (система може вважати себе «готовою» генерувати, хоча реально ентропії мало).
- Приклади наслідків:
    - Відомі атаки на RSA та DSA-ключі через передбачувані прості числа (наприклад, атаки на слабкі ключі Debian OpenSSL 2008).
    - Можливість факторизації ключів, якщо відомо кілька старших або молодших бітів простих.

Рекомендація: забезпечити збір ентропії з декількох джерел (апаратні RNG, системні джерела /dev/random, HWRNG), використовувати періодичне оновлення стану PRNG та моніторинг достатності випадковості.

9.3 Теоретичні межі порівнянь без емпіричних вимірів

- Теорія ≠ практика.
  Хоча аналітичні оцінки дають уявлення про складність (наприклад, O(k·log n·M(n)) для Міллера–Рабіна), вони не враховують:
    - накладних витрат на алокацію пам’яті;
    - кеш-промахів;
    - оптимізацій на рівні збірки (ASM);
    - багатопоточність.
- Без експериментів важко оцінити:
    - фактичну кількість кандидатів, що доведеться перевірити (реальний розподіл простих трохи відрізняється від теоретичного);
    - середню швидкість modular exponentiation на конкретному залізі;
    - вплив оптимізацій (наприклад, використання Montgomery multiplication або Barrett reduction).
- Ризик: теоретичне планування без вимірів може призвести до переоцінки продуктивності або до вибору надмірно складного алгоритму там, де простий був би достатнім.

Рекомендація: проводити емпіричні тести на цільових платформах: заміряти час генерації простих, кількість перевірок, середній час одного раунду тесту простоти.

Висновок

1. Модель довгої арифметики є зручною для теорії, але може спотворювати реальні оцінки продуктивності на обмежених пристроях.
2. Якість ентропії — критично важлива для безпеки: слабкий або передбачуваний seed робить усю криптографічну систему вразливою.

3. Теоретичний аналіз без експериментів призводить до ризику невідповідності продуктивності та безпеки реальним умовам, особливо при виборі апаратних платформ.

Таким чином, при розробці бібліотеки необхідно не лише враховувати асимптотичні оцінки, а й проводити тестування на реальних пристроях та перевіряти якість ентропії.

10) Ризики, обмеження і припущення

На основі проведеного аналізу алгоритмів генерації випадкових чисел, тестів простоти та апаратних обмежень можна сформулювати практичні поради для різних категорій пристроїв та середовищ виконання. Рекомендації враховують баланс між безпекою, продуктивністю та ресурсними обмеженнями.

10.1 Рекомендовані поєднання

Сервери та настільні системи (Server/Desktop)

- Рекомендована схема:
    - Надійний генератор випадкових чисел: CTR-DRBG або ChaCha20-DRBG.
    - Тестування простоти: Міллер–Рабін з достатньо великим числом раундів kkk, яке забезпечує ймовірність помилки ≤ 2^{-128}.
    - Для випадків, де потрібні *сертифіковані прості числа*, — використання алгоритму Мауєра (Maurer's algorithm).
- Обґрунтування:

  Сервери та десктопи мають достатню обчислювальну потужність і пам'ять, що дозволяє виконувати складні операції модульної арифметики та збирати ентропію з кількох джерел (системні RNG, апаратні модулі).

  Використання алгоритму Мауєра дає доведений сертифікат простоти, що особливо актуально для високозахищених систем (PKI, державні криптосистеми).

  ChaCha20-DRBG та CTR-DRBG — добре перевірені та ефективні DRBG, які забезпечують високу ентропію навіть при великій кількості запитів.

Смарткарти та токени (Smartcard/Token)

- Рекомендована схема:
    - Легкий та компактний DRBG (наприклад, апаратне джерело шуму + простий DRBG із низьким споживанням пам'яті).
    - Тестування простоти: Міллер–Рабін із мінімально безпечним числом раундів kkk, доповненим відсіюванням малими простими (wheel factoring).
    - Для швидкої перевірки придатності числа — метод Чебишова як практичніший за алгоритм Мауєра у середовищах із обмеженими ресурсами.

- Обґрунтування:
  Смарткарти мають суворі обмеження на пам’ять, тактову частоту й енергоспоживання, тому використання важких алгоритмів (як Мауєр) недоцільне.
  Оптимізація шляхом відсіювання малими простими дозволяє зменшити кількість дорогих раундів Міллера–Рабіна.
  Простий DRBG із апаратним шумом дозволяє зберегти енергобюджет і уникнути витрат на складні схеми з reseed-політикою.

Смартфони (Android / iOS)

- Рекомендована схема:
  - Використання системного DRBG, вбудованого в ОС (наприклад, getrandom() в Linux/Android, SecureRandom в iOS).
  - Тестування простоти: Міллер–Рабін із адаптивною кількістю раундів залежно від потрібного рівня безпеки (зазвичай k=40 забезпечує помилку ≤ 2^{-80}2−80, k=64 ≤ 2^{-128}).
  - За потреби — комбінований reseed з кількох джерел ентропії (сенсори, мережеві події, апаратний RNG).
- Обґрунтування:
  Смартфони мають більше обчислювальних ресурсів, ніж смарткарти, але менше, ніж сервери.
  ОС уже надає високоякісний генератор випадкових чисел, тому додаткове впровадження власного DRBG часто недоцільне.
  Міллер–Рабін забезпечує достатню швидкість та безпеку, а використання комбінованого reseed покращує стійкість при тривалому використанні без перезавантаження.

Загальні рекомендації

1. Забезпечення високої якості ентропії.
   Навіть найкращі тести простоти не гарантують безпеки, якщо генератор випадкових чисел передбачуваний.
   Рекомендується використовувати кілька джерел ентропії та періодичний reseed.
2. Гнучке налаштування кількості раундів kkk.
   - Для звичайних комерційних застосунків достатньо k=40k = 40k=40 (помилка ≤ 2^{-80}).
   - Для державних або довгострокових ключів — k=64k = 64k=64 (помилка ≤ 2^{-128}).
3. Оптимізація для обмежених пристроїв.
   - Використовуйте wheel factoring для швидкого відсіювання непростих чисел.

    - Уникайте складних сертифікуючих тестів (Мауєр) на смарткартах.
    - Слідкуйте за пам’яттю та енергоспоживанням.
4. Перевірка продуктивності на цільовому обладнанні.
   Теоретичні оцінки не завжди збігаються з реальною швидкістю, тому рекомендується проводити емпіричні вимірювання часу генерації простих на конкретних пристроях.

Висновок

- Сервери та десктопи: потужніші алгоритми (Мауєр + DRBG класу CTR/ChaCha20).
- Смарткарти/токени: максимально легкі схеми (простий DRBG + оптимізований Міллер–Рабін із мінімальним kkk).
- Смартфони: використання системного DRBG та Міллера–Рабіна з адаптивним налаштуванням безпеки.

Таке розділення дозволяє збалансувати безпеку та ефективність у різних апаратних середовищах і уникнути критичних проблем, пов’язаних із низькою ентропією, надмірною складністю або невиправданими витратами ресурсів.