Лабараторна робота №2

Тема: реалізація алгоритмів генерації ключів гібридних криптосистем.

Виконали: студенти групи ФІ-32мн Ємець Єлизавета, Звєрєв Сергій, Коваленко Дар'я

Завдання: дослідити різні методи генерації випадкових послідовностей для засобів обчислювальної техніки. Дослідити ефективність за часом алгоритми тестування на простоту різних груп – імовірнісних, гипотетичних та детермінованих. Порівняти ймовірність похибки різних імовірнісних тестів (Ферма, Соловея-Штрассена 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 – 768, 1024 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.

І.Класифікація методів генерації випадкових послідовностей :

1. Фізичні генератори випадкових чисел (True Random Number Generators - TRNGs):

- Радіоактивний розпад: Використовує непередбачуваність розпаду радіоактивних елементів. Використання гейгерівського лічильника для вимірювання часу між радіоактивними розпадами.
- Термічний шум: Заснований на випадковості теплового руху частинок. Використання мікросхеми шуму, яка перетворює тепловий шум електронних компонентів у випадкові числа.
- Квантові явища: Використовує непередбачуваність квантової механіки, наприклад, через фотони, що проходять через напівпрозоре дзеркало. Quantum random number generators (QRNGs), які використовують квантову непевність, наприклад, у способі поведінки фотонів.

2. Апаратні генератори випадкових чисел (Hardware Random Number Generators - HRNGs):

- Електронний шум: Використовує шум напівпровідникових компонентів. Використання шуму напівпровідникових діодів або транзисторів для генерації випадкових чисел.

- Динамічні системи: Засновані на непередбачуваній поведінці комплексних динамічних систем. Генератори на основі хаотичних електронних коливань.

3. **Псевдовипадкові генератори чисел (Pseudorandom Number Generators - PRNGs):**
- Лінійні конгруентні методи: Найпростіші та найбільш розповсюджені ПВГ. Генератори випадкових чисел, які використовують лінійні конгруентні формули (наприклад, генератор випадкових чисел ANSI C).
- Методи, засновані на хеш-функціях: Використовують криптографічні хеш-функції для створення випадкових послідовностей. Використання криптографічних хеш-функцій, таких як SHA-256, для створення випадкових послідовностей.
- Методи, засновані на блочних шифрах: Використовують алгоритми шифрування для генерації послідовностей.Використання шифру AES у режимі генерації випадкових чисел.
- Мерсеннські спірали: Високоякісні ПВГ, що використовують періоди Мерсенна для створення великих послідовностей випадкових чисел. Мерсеннський спіральний генератор MT19937.
- Комбіновані методи: Поєднують декілька алгоритмів для підвищення якості випадковості. Використання комбінації декількох алгоритмів, наприклад, генератор випадкових чисел /dev/random у UNIX-системах.

4. **Криптографічно стійкі генератори випадкових чисел (Cryptographically Secure Pseudorandom Number Generators - CSPRNGs):**
- Методи, засновані на еліптичних кривих: Використовують властивості еліптичних кривих в складних обчислювальних просторах. Генератори, які використовують властивості еліптичних кривих, наприклад, Dual Elliptic Curve Deterministic Random Bit Generator (Dual_EC_DRBG).
- Фортуну та Ярров: Спеціальні алгоритми, розроблені для криптографічних застосувань. Генератор випадкових чисел Фортуну або генератор Яррова, які спеціально розроблені для криптографічних застосувань.
- Алгоритми з використанням ентропії: Сполучають вхідні дані з високою ентропією з математичними алгоритмами для створення випадкових чисел. Використання даних із високою ентропією з операційної системи, які потім обробляються алгоритмами для створення випадкових чисел (наприклад, Linux / dev/random).

	Безпека	**Ефективність**	**Сильні сторони**	**Слабкі сторони**
Фізичні(TRNGs)	Висока, оскільки вони генерують справжні випадкові числа, не повторюються і не можуть бути відтворені.	Часто повільніші за інші методи та можуть вимагати спеціалізованого обладнання.	Висока випадковість і безпека.	Вища вартість, складність інтеграції з обчислювальними системами.

	Безпека	Ефективність	Сильні сторони	Слабкі сторони
Апаратні(HRNGs)	Забезпечують добру випадковість, але можуть бути схильні до фізичних втручань та атак.	Перевага у використанні фізичних явищ для випадковості, але залежні від апаратного обладнання.	Добра випадковість і порівняно прості в реалізації.	Вартість, обмеження у швидкості генерації, схильність до фізичних втручань.
Псевдовипадкові генератори чисел (PRNGs)	Залежить від алгоритму; деякі PRNGs не підходять для криптографічних застосувань.	Висока швидкість та низька вартість.	Простота у використанні, не вимагають спеціалізованого обладнання.	Випадковість і безпека залежать від вибору алгоритму і початкового сідла.
Криптографічно стійкі генератори випадкових чисел (CSPRNGs):	Дуже висока, підходять для криптографічних застосувань.	Можуть бути повільнішими за звичайні PRNGs через додаткові обчислення для забезпечення безпеки.	Висока криптографічна безпека і непередбачуваність.	Більша складність у реалізації та можливе зниження продуктивності.

Кожен з цих методів має свої унікальні характеристики, переваги та недоліки. Фізичні та апаратні генератори зазвичай надають високу ентропію та непередбачуваність, але можуть бути повільними та дорогими у виробництві. Псевдовипадкові генератори є швидшими та легшими у використанні, але їх якість залежить від алгоритму та початкового "сідла" (seed). Криптографічно стійкі методи найкраще підходять для застосувань, де безпека є критично важливою.

II. Аналіз ефективності алгоритмів тестування на простоту можна провести на прикладах трьох основних типів алгоритмів: імовірнісних, гіпотетичних і детермінованих.

1. Імовірнісні Алгоритми:

- Тест Ферма: Перевіряє, чи задовольняє число n рівнянню $a^{n-1} \equiv 1(mod n)$ для випадкового a. Цей тест швидкий, але може помилково класифікувати складені числа як прості (так звані числа Кармайкла).
- Тест Соловея-Штрассена: Використовує розширення тесту Ферма з використанням символу Якобі. Кращий за тест Ферма у виявленні складених чисел, але все ще є імовірнісним.
- Тест Міллера-Рабіна: Один з найпопулярніших імовірнісних тестів. Вважається більш надійним і точним, особливо при використанні декількох раундів перевірки.

2. Гіпотетичні Алгоритми:

- Тест АКС: Алгоритм тестування на простоту Агравала-Каяла-Саксени (AKS) є детермінованим і працює за поліноміальний час. Він важливий з теоретичної

точки зору, але його практична ефективність обмежена через високу складність алгоритму.

3. Детерміновані Алгоритми:
- Тест Елліптичних Кривих: Використовує властивості еліптичних кривих для перевірки простоти числа. Цей метод ефективніший для великих чисел, але вимагає більшої обчислювальної потужності.
- Метод Ератосфена: Це один з найдавніших відомих алгоритмів для знаходження всіх простих чисел до заданого числа n. Дуже ефективний для малих чисел, але не підходить для великих чисел через великі обчислювальні вимоги.

Оцінка ефективності:
- Швидкість: Імовірнісні алгоритми (особливо Міллера-Рабіна) часто швидші за детерміновані, особливо для великих чисел.
- Точність: Детерміновані алгоритми гарантують точність, але вони можуть бути повільнішими та більш складними у використанні.
- Практичність: Імовірнісні алгоритми зазвичай вважаються більш практичними для більшості застосувань, зокрема в криптографії, де висока швидкість є критичною.

У підсумку, вибір між імовірнісними, гіпотетичними та детермінованими алгоритмами залежить від конкретних потреб та обмежень задачі, включаючи необхідний баланс між швидкістю, точністю та обчислювальною складністю.

Імовірність похибки в тестах на простоту Ферма, Соловея-Штрассена, і Міллера-Рабіна можна оцінити, враховуючи їх алгоритмічні властивості та поведінку при різній кількості ітерацій:

1. Тест Ферма:
- Цей тест базується на малій теоремі Ферма, яка стверджує, що якщо p є простим числом і a є цілим числом, що не ділиться на p, то $a^{p-1} \equiv 1(mod p)$.
- Імовірність помилки: Тест Ферма може невірно класифікувати складене число як просте. Ця помилка особливо проблематична для чисел Кармайкла. Втім, для випадково обраного основи a, імовірність такої помилки не дуже висока, але вона не зменшується з кількістю ітерацій.

2. Тест Соловея-Штрассена:
- Цей тест використовує символ Якобі в своєму алгоритмі. Імовірність помилки в тесті Соловея-Штрассена зазвичай нижча, ніж у тесту Ферма.
- Імовірність помилки: Для випадково вибраного числа a імовірність помилкового виявлення складеного числа як простого є не більше ніж 1/2. З кожною ітерацією тесту ця імовірність зменшується експоненційно.

3. Тест Міллера-Рабіна:

- Цей тест є поліпшенням тесту Ферма і є більш складним та надійним.
- Імовірність помилки: Для випадково вибраного числа a, імовірність помилкового виявлення складеного числа як простого є не більше ніж 1/4. Так само, як і у тесті Соловея-Штрассена, з кожною ітерацією тесту імовірність помилки зменшується експоненційно.

Кореляція з імовірністю помилок на ПЕОМ:
Імовірність помилок у цих тестах не прямо пов'язана з імовірністю помилок на ПЕОМ, яка зазвичай обумовлена апаратними несправностями, помилками у програмному забезпеченні або некоректними обчисленнями. Однак, якщо ПЕОМ має недоліки у точності обчислень або стабільності, це може вплинути на результати тестування на простоту, зокрема, при великих числах або багатьох ітераціях.

У підсумку, імовірнісні тести на простоту зазвичай надають гарний баланс між швидкістю та надійністю для більшості практичних застосувань, особливо в криптографії, де важлива швидкість та можливість обробки великих чисел. Тест Міллера-Рабіна, зокрема, є досить популярним вибором через його надійність та ефективність.

Давайте розглянемо детальніше кожен з цих імовірнісних тестів на простоту, а також їх імовірності помилок:

1. Тест Ферма:
- Принцип: Заснований на малій теоремі Ферма, яка стверджує: якщо p є простим числом, то для будь-якого цілого числа a що не ділиться на p, буде виконуватись рівняння $a^{p-1} \equiv 1(mod p)$.
- Імовірність помилки: Хоча тест добре виявляє багато складених чисел, існують складені числа, звані числами Кармайкла, які завжди проходять тест Ферма. Тому імовірність помилки для цих чисел дуже висока. Однак, для випадково обраного числа a імовірність помилки є низькою. Підвищення кількості ітерацій тесту з різними основами a не значно зменшує імовірність помилки.

2. Тест Соловея-Штрассена:
- Принцип: Цей тест використовує символ Якобі і перевіряє, чи задовольняє число n рівнянню $a^{\frac{n-1}{2}} \equiv (\frac{a}{n}) mod n$, де $(\frac{a}{n})$ є символом Якобі.
- Імовірність помилки: Імовірність помилкового виявлення складеного числа як простого є не більше ніж 1/2 для кожного випробування. Імовірність помилки зменшується експоненційно з кількістю ітерацій (1/2^k для k ітерацій).

3. Тест Міллера-Рабіна:

- Принцип: Поліпшення тесту Ферма, яке зводиться до перевірки, чи число n проходить серію тестів на основі певних умов та рівнянь, пов'язаних з $n - 1$.
- Імовірність помилки: Для випадково вибраного основи a, імовірність помилкового класифікування складеного числа як простого є не більше ніж 1/4. Подібно до тесту Соловея-Штрассена, імовірність помилки зменшується експоненційно з кількістю ітерацій (1/4^k для k ітерацій).

Кореляція з помилками на ПЕОМ:
Імовірність помилок в цих тестах визначається алгоритмічними характеристиками, а не апаратними чи програмними недоліками ПЕОМ. Втім, якщо ПЕОМ має проблеми з точністю обчислень або надійністю, це може вплинути на результати тестування. Наприклад, помилки округлення або некоректні арифметичні операції можуть призвести до неправильних результатів тестів, особливо при роботі з великими числами.

У підсумку, тести Міллера-Рабіна та Соловея-Штрассена вважаються більш надійними та ефективними для використання у практичних сценаріях, особливо у криптографії, де важливі швидкість та точність. Ці тести забезпечують гарний баланс між швидкістю обчислень та імовірністю помилки, особливо при проведенні кількох ітерацій.

III.Розробка бібліотеки для генерації псевдовипадкової послідовності, тестування на простоту чисел, та генерації простих чисел на Python може бути реалізована з використанням стандартних бібліотек мови. Для підтримки чисел з розмірністю 768 та 1024 біти, можна використовувати бібліотеку random для генерації псевдовипадкових чисел та бібліотеку sympy для тестування на простоту та генерації простих чисел.

Псевдовипадкове число (768 біт):
123376342355629432106638878683069159802412973711851598852270683971123387559588322972930686593632919379545191311443447979395648878329421834827970008583092441604279871230582532587568544007651186980182211873882369426057559056424976272 8

Просте число (1024 біти):
859470612908627382633217982959305622074141300165607973260119699417139368126718034781057406398672021549719095839238744509837341727220812630810465828104909102827822072014345268401442923771262475485495225547469244843124443257480585374625485596372343401122813602874298537135518950047027761895129114382004686293 31

```python
import random
from sympy import isprime, primerange

class CryptoLibrary:
    def generate_random_number(self, bit_length):
        """Генерує псевдовипадкове число заданої довжини в бітах."""
        return random.getrandbits(bit_length)

    def is_prime(self, n):
        """Перевіряє, чи є число простим."""
        return isprime(n)

    def generate_prime_number(self, bit_length):
        """Генерує просте число заданої довжини в бітах."""
        prime_candidate = 4  # починаємо з не простого числа
        while not self.is_prime(prime_candidate):
            prime_candidate = self.generate_random_number(bit_length)
        return prime_candidate

# Створення екземпляра бібліотеки
crypto_lib = CryptoLibrary()

# Генерація псевдовипадкового числа з довжиною 768 біт
random_number_768 = crypto_lib.generate_random_number(768)

# Генерація простого числа з довжиною 1024 біти
prime_number_1024 = crypto_lib.generate_prime_number(1024)

print(f"Псевдовипадкове число (768 біт): {random_number_768}")
print(f"Просте число (1024 біти): {prime_number_1024}")
```

IV.Створення безпечного генератора псевдовипадкових чисел (ПВЧ) для інтелектуальних карток, токенів та смартфонів - це важлива завдання з точки зору криптографічної безпеки. Такі пристрої потребують надійного інформаційного забезпечення для запобігання можливим атакам. Ось загальна схема, яка може бути використана для генерації ПВЧ:

- **Збір ентропії:**
  - Використовуйте різні датчики на пристрої, такі як гіроскоп, акселерометр, сенсори температури, освітлення і т.д., для збору ентропії.
  - Записуйте значення з цих датчиків у вигляді сирого вхідного матеріалу.
- **Фільтрація та обробка ентропії:**

    - Використовуйте фільтрацію та обробку сирої ентропії для видалення споживчого шуму і виправлення можливих помилок датчиків.
    - Використовуйте криптографічні хеш-функції для обчислення хеш-значень отриманих даних.
- **Інструкції процесора:**
    - Використовуйте інструкції процесора для збирання додаткової ентропії. Наприклад, ви можете використовувати час виконання інструкцій, накладені якости кешів та інші характеристики процесора.
- **Шумові генератори:**
    - Використовуйте шумові джерела, такі як діоди Шотткі або транзистори, для створення додаткової ентропії.
- **Міксування ентропії:**
    - Змішайте ентропію з усіх джерел для отримання загальної ентропії.
    - Використовуйте криптографічні алгоритми міксування, такі як HMAC, для змішування даних із різних джерел.
- **Генерація ключів:**
    - Використовуйте криптографічний алгоритм, наприклад, блочний шифр, для генерації випадкових бітів зі змішаної ентропії.
    - Важливо використовувати безпечні криптографічні алгоритми і налагодити їх належним чином.
- **Виведення ПВЧ:**
    - Виведіть згенеровані випадкові біти у вигляді ключів або сидів для криптографічних операцій на пристрої.
- **Перевірка якості та моніторинг:**
    - Періодично перевіряйте якість генерованих ПВЧ і визначайте, чи не виникає аномалій.
    - Встановіть системи моніторингу та журналювання для виявлення аномальної поведінки генератора.

Ця схема враховує різні джерела ентропії, змішує їх, фільтрує та обробляє дані, а потім генерує випадкові числа з використанням криптографічних методів. Важливо враховувати безпеку на всіх етапах та дотримуватися кращих практик забезпечення інформаційної безпеки.

Розглянемо конкретну схему для збору ентропії та генерації ПВЧ, яка може бути використана на смартфонах, інтелектуальних картках та токенах. Ця схема базується на апаратному та програмному забезпеченні пристрою:

1. Збір ентропії з датчиків:

- Використовуйте датчики апаратного забезпечення пристрою, такі як гіроскоп, акселерометр, сенсори температури і освітлення, для збору вхідних даних.
- Записуйте значення цих датчиків у буфер для подальшого використання.

2. Збір шуму від апаратного обладнання:
- Використовуйте шумові джерела апаратного обладнання, такі як діоди Шотткі або транзистори, для генерації електричного шуму.
- Зберіть цей шум і додайте його до буфера ентропії.

3. Вибірковий опитувальник (entropy mixer):
- Використовуйте вибірковий опитувальник для вибору випадкових даних з буфера ентропії. Цей опитувальник може використовувати різні джерела ентропії, наприклад, дані з датчиків і шумового джерела.
- Забезпечте, щоб вибірковий опитувальник був добре збалансованим і мав високий ступінь випадковості.

4. Фільтрація та обробка ентропії:
- Використовуйте криптографічні хеш-функції (наприклад, SHA-256) для обробки ентропії та обчислення хеш-значень.
- Фільтруйте дані, щоб видалити шум та надійшли дані.

5. Криптографічні перетворення:
- Використовуйте криптографічні алгоритми, такі як блочні шифри або HMAC, для перетворення оброблених даних в випадкові числа.
- Використовуйте ключі, які змінюються з часом, або інші криптографічні методи для підвищення безпеки.

6. Виведення ПВЧ:
- Виводьте отримані випадкові числа у вигляді ключів або сидів для криптографічних операцій на пристрої.

7. Моніторинг та тестування:
- Періодично перевіряйте якість генерованих ПВЧ та визначайте, чи не виникає аномалій.
- Встановіть системи моніторингу та журналювання для виявлення аномальної поведінки генератора.

Ця схема поєднує декілька джерел ентропії, обробляє їх, перетворює і виводить випадкові числа з використанням криптографічних методів. Важливо використовувати безпечні алгоритми та враховувати криптографічні вимоги для забезпечення високого рівня безпеки ПВЧ.