Звіт з виконання лабораторної роботи 2
Варіант 1А (Генерація ПВЧ. Тест на простоту. Генерація простих чисел.)

ФІ-42мн Бондар Петро
ФІ-42мн Кістаєв Матвій

January 5, 2025

Огляд

Підгрупа: 1А.

Мета: Дослідити криптографічно стійкі алгоритми генерації псевдо-випадкових послідовностей, тестів числа на простоту, а також алгоритмів генерації випадкового простого числа.

Завдання (модифіковане):

1. Теоретично дослідити алгоритми перевірки числа на простоту, генерації псевдо-випадкових послідовностей та псевдо-випадкових простих чисел.
2. На основі попереднього аналізу обрати алгоритми для реалізації та порівняння.
3. Реалізувати бібліотеку із обраних алгоритмів на `Python`.
4. Провести теоретичне та практичне порівняння алгоритмів, що виконують однакові (схожі) задачі.

Тести на простоту

Алгоритми перевірки числа на простоту поділяються на 3 типи:

1. Базуються на перевірці на складеність (ймовірнісні).
2. Точна перевірка на простоту (детермністичні).
3. Лише підтверджують простоту числа.

Детерміністичні тести (найбільш відомий – AKS) хоч і мають поліноміальну складність, проте, вона зазвичай є завеликою, щоб використовувати їх на практиці. Тому переважно використовують саме ймовірнісні тести, що базують на перевірці на складеність.

Найвідомішими такими тестами є:

- Ферма
- Соловея-Штрассена
- **Міллера-Рабіна** (було реалізовано)

Серед цих тестів найкращим є Міллера-Рабіна, оскільки він є ефективним з точки зору обчислень, ймовірність помилки (псевдопростоти за випадковою основою) не перевищує $\frac{1}{4}$ *для довільного складеного числа.*

Як слідує із назви, підтверджуючі тести не є придатними для задачі перевірки заданого числа на простоту, оскільки вони можуть лише підтвердити, коли число є простим. Але вони використовуються для певних швидких генераторів випадкових простих чисел (буде далі).

Велика таблиця із порівнянням більшості найвідоміших тестів наведена нижче.

Table 1: Comprehensive Comparison between different primality tests.

Reference	**Characteristics**	**Pros**	**cons**	**Time Complexity**	**Probability of error**
Fermat [20]	- Randomized (Monto -Carlo). - Compositeness test.	- Very simple to implement. - Base for many tests.	-Failure probability may reach 1. -Pseudoprime can pass the test.	$O(k\,logn)^3$	- For Carmicheal numbers =1. - For pseudoprimes $\approx 245 * 10^{-6}$.
Solovay Strassen [24]	- Randomized (Monto-Carlo). - Compositeness test.	- Pseudoprimes are successfully announced as composites.	- an Euler pseudoprime can pass the test. - Computation of Jacobi symbol adds more computation overhead.	$O(k\,logn)^3$	less than $\left(\frac{1}{2}\right)^k$, where k is the certainty value.
Miller-Rabin [25, 25]	- Randomized (Monto-Carlo). - Compositeness test.	- Fast & efficient. - Euler Pseudoprimes are successfully announced as composites.	- Strong pseudoprimes can pass the test.	$O(k\,logn)^3$	less than $\left(\frac{1}{2}\right)^k$, where k is the certainty value.
Lucas [30]	- Randomized (Las-Vegas). - Approving primality test.	- Valid for any generic, or special form numbers.	- Prime factors of $n-1$ are required to be already known. - Worst case scenario may take long time. (if n is composite, this test may not terminate).	$O(p^2\,log_p n)$	- 0 if the number is announced as a prime.
Proth [27]	- Randomized (Las-Vegas). - Approving primality test.	- Very fast and reliable test to decide about proth number.	- Working well only with proth numbers.	$O((k\,logk+logn)logn))$	- 0 if the number is announced as a prime. - ≈ 0 if the base a covers 50% of the range $[2, n-1]$.
Pocklington [31]	- Randomized (Las-Vegas). - Approving primality test.	- Very efficient if there is a factor $q > \sqrt{n-1}$.	- Prime factors of $n-1$ are required to be already known.	$O(lnln\,n)$	- 0 if the number is announced as a prime.
Lucas Lehmer [34]	- Deterministic. - Compositeness & Approving primality test.	- Testing Mersenne numbers accurately. - Practical test to find massive primes.	- Suffers slowness when numbers become huge. - Prime factors of n-1 are required to be already known.	$O(n^2\,logn\,loglogn)$	- 0.
Trivial Divison [3]	- Deterministic. - Compositeness & Approving primality test.	- Reliable and simple test for numbers consist of 10 digits or less.	- Computational infeasible for large numbers (exponential increasing).	$O(\sqrt{n})$	- 0.
Pepin [32]	- Deterministic. - Compositeness & Approving primality test.	- Efficient test for Fermat form numbers.	- It is computational infeasible for large Fermat numbers. - Since the base a is fixed, it works poorly with other formats.	$O(logn)^2$	- 0.
AKS [35]	- Deterministic. - Compositeness & Approving primality test.	- The only reliable deterministic test that works in polynomial time .	- Slow for small and moderate numbers.	$O(logn^6)$	- 0.
Ballie-PSW [37, 38]	- Huerestic.	- None of the pseudoprime classes passes this test.	- No Mathematical approving. - Considered as compositeness test.	$O(k\,logn)^3$	- 0 for any $n < 2^{64}$.

Figure 1: Таблиця порівняння особливостей та складності алгоритмів перевірки числа на простоту. (Не наша, ref: `https://arxiv.org/abs/2006.08444`)

Генератори псевдо-випадкових чисел

Криптографічно стійкі генератори псевдо-випадкових чисел мають задовільняти наступним вимогам:

- Проходити усі статистичні тести на випадковість.
- Бути стійким до атак з компрометацією внутрішнього стану (як forward, так і backward security).

Такі генератори часто використовують т.зв. додаткові джерела ентропії (наприклад звичайний, не стійкий, системний генератор), а також можуть мати секрет (ключ).

Для досягнення Backward security (тобто неможливість відворити попередні значення, навіть знаючи

поточне значення та внутрішній стан генератора) функція переходу в наступний стан використовує певний стійкий криптографічний примітив (наприклад геш-функції або блокового шифру) або односторонню функцію, що базується на складній обчислювальній задачі (наприклад функція Рабіна).

Ми досліджували генератори псевдо-випадкових чисел в лабораторній роботі в курсі асиметричної криптографії 1. (див. файл `asym_lab1.ipynb`). Серед досліджених, найкращим виявився генератор Blum-Blum-Shub (`BBS`), який базується на функії Рабіна.

В цій лабораторній ми також реалізували генератор по схемі NIST – `Hash_DRGB`, що використовує криптографічно-стійку геш-функцію – `SHA256`.

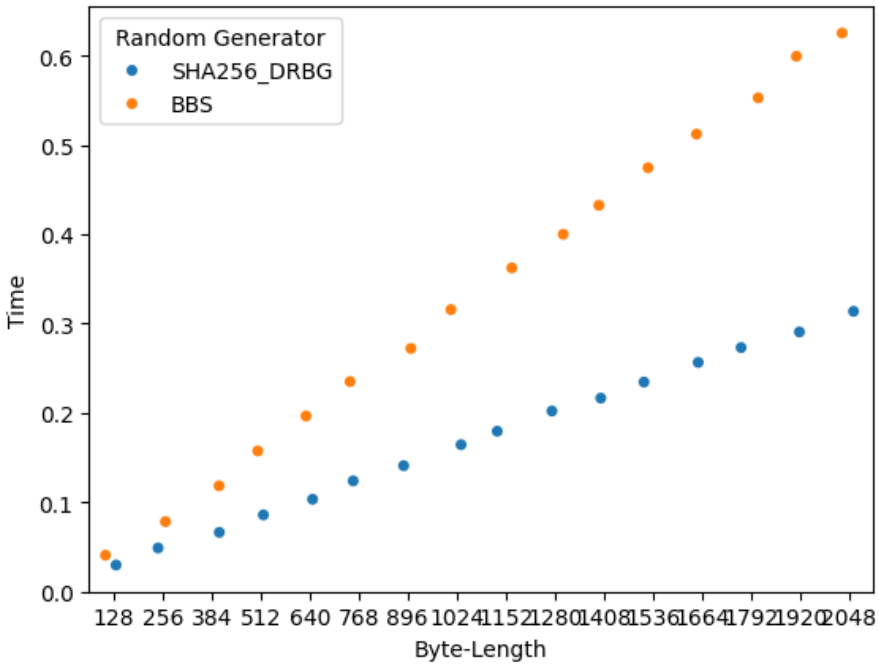


Figure 2: Порівняння часу роботи криптографічно стійких генераторів псевдовипадкових чисел `BBS` та `SHA256_DRBG`.

Генератори простих чисел

Для генерації псевдо-випадкового *простого* числа завичай використовується два підходи:

- Генерування випадкових чисел та перевірка їх на простоту ймовірнісним тестом.
- Випадкова "побудова" доведено-простого числа, використовуючи підтверджуючий тест на простоту.

Для першого типу генераторів в лабораторній реалізовано генератор на основі тесту Міллера-Рабіна. Для другого – генератор Маурера на основі тесту Поклінгтона.

Генератор Маурера будує "майже випадкові" складені числа R та F, $F > R$, такі щоб виконувалась теорема:

Теорема (Поклінгтон). *Якщо для $N = 2RF + 1$, із відомим розкладом $F = \prod q_i^{\beta_i} > \sqrt{N}$, для кожного q_i існує число a_i таке що виконуєтсья:*

$$a_i^{N-1} \equiv 1 \ (\mathrm{mod} N)$$

та

$$\gcd(a_i^{(N-1)/q_i},\ N) = 1$$

тоді число N – просте.

Алгоритм Маурера працює швидше та "надійніше" за випадковий перебір в поєднанні із тестом на простоту, проте він має дещо обмежений простір можливих згенерованих чисел, а також не повністю рівномірний розподіл на цьому просторі.

На практиці, алгоритм Маурера використовується для генерації простих чисел в таких криптосистемах, де прості числа є публічними параметрами (наприклад Ель-Гамаля). Для таких криптосистем, як `RSA`, використовується генерація перебором в поєднанні із тестом Міллера-Рабіна, оскільки від цих простих чисел залежить стійкість ключа.

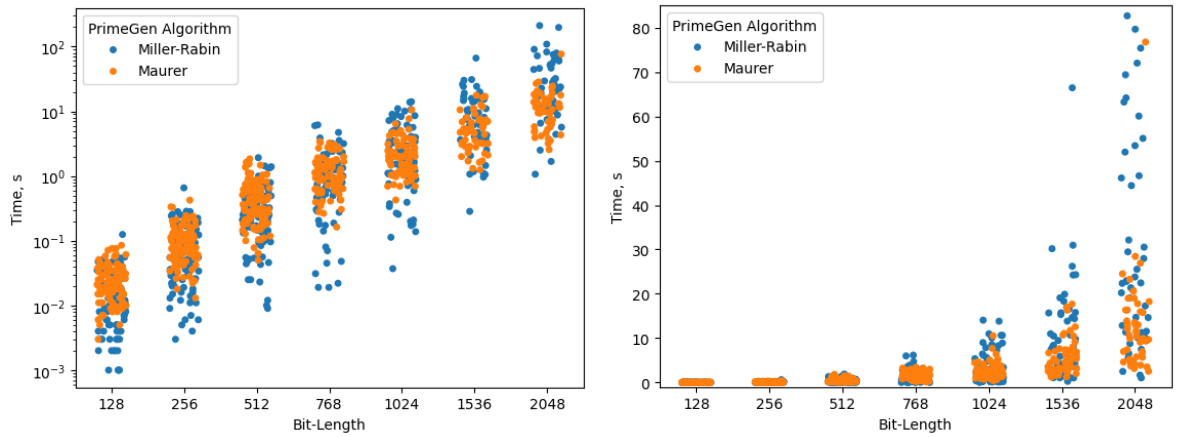


Figure 3: Порівняння генераторів простих чисел:
- випадковий перебір + тест Міллера-Рабіна
- генератор Маурера

Бібліотека

В `Python`-файлі `optimus.py` знаходяться імплементації всіх зазначених вище алгоритмів (та не тільки їх, а й інших теоретико-числових алгоритмів для криптографії).

В файлах `rng_test.py` та `primate.py` знаходяться скрипти для перевірок часу роботи розглянутих алгоритмів, за допомогою них були згенеровані наведені вище графіки.

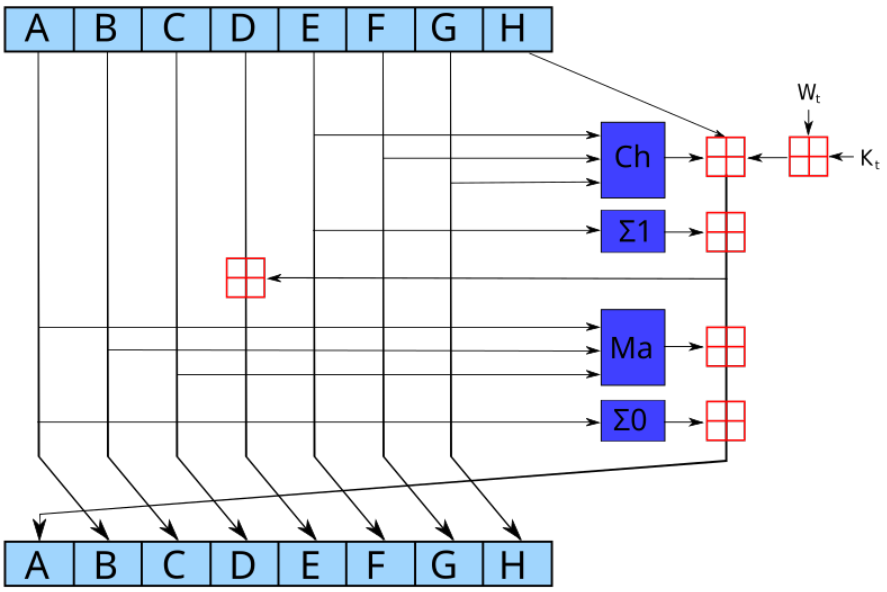
Генератор ПВЧ для інтелектуальної картки, токена та смартфону.

Розглянути особливості побудови генератора простих чисел в умовах обмеження пам'яті та часу генерації.

Серед двох розглянутих генераторів, `Hash_DRBG` на основі геш-функції `SHA-2` однозначно перемагає `BBS` в усіх сенсах:

- Швидше
- Не вимагає складних арифметичних операції над великими числами
- Не поступається в стійкості
- Використовує розповсюджений криптографічний примітив – `SHA-2`

Функцію `SHA-2` можна запрограмувати навіть на картоплі – схема Меркле-Дамгарда потребує 3 регістри, а стискаюча функція має наступний вигляд:



One iteration in a SHA-2 family compression function. The blue components perform the following operations:

$$\mathrm{Ch}(E,F,G) = (E \wedge F) \oplus (\neg E \wedge G)$$
$$\mathrm{Ma}(A,B,C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$$
$$\Sigma_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$$
$$\Sigma_1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$$

The bitwise rotation uses different constants for SHA-512. The given numbers are for SHA-256.

The red ⊞ is addition modulo 2^{32} for SHA-256, or 2^{64} for SHA-512.

Figure 4: Схема стискаючої функції `SHA-2`.

А схема самого генератора виглядає наступним чином:

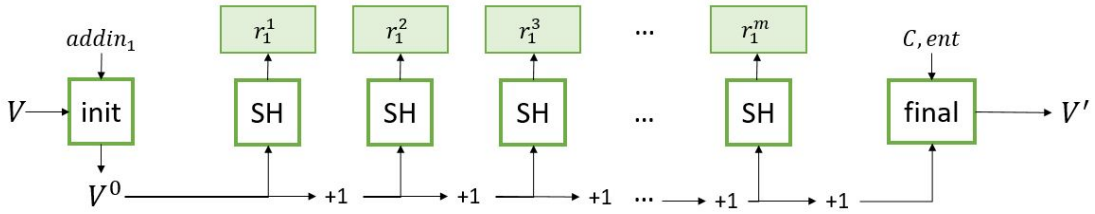


Figure 5: Схема генератора `SHA256_DRBG`.

Тому генератор `SHA256_DRBG` добре підійде для пристроїв з дуже обмеженими обчислювальними ресурсами.