

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

Звіт з виконання лабораторної роботи  
**РЕАЛІЗАЦІЯ АЛГОРИТМІВ  
ГЕНЕРАЦІЇ КЛЮЧІВ ГІБРИДНИХ  
КРИПТОСИСТЕМ**

Виконали студенти  
групи ФІ-52мн  
Волошин Ігор  
Геращенко Володимир  
Дорошенко Юрій

**Варіант 2Б**

# 1 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 генерации ключів асиметричних криптосистем. Аналіз генерації у бібліотеці PyCrypto під Linux платформу.

## 2 Теоретичні відомості

### 2.1 Генератор псевдовипадкових послідовностей в PyCrypto (Linux)

ГПВП в бібліотеці PyCrypto (та її сучасному форку PyCryptodome) на платформі Linux реалізовано як криптографічно стійкий генератор (CSPRNG), який пріоритетно спирається на механізми операційної системи, але має власний алгоритм для випадків, коли потрібен об'єктний доступ.

- **Джерела ентропії:** У середовищі Linux PyCrypto делегує збір «сирої» ентропії ядру операційної системи. Основні механізми включають:
  - *Системний виклик `getrandom()` (Linux 3.17+)*: Сучасний метод, який дозволяє отримувати випадкові байти безпосередньо з ядра, не використовуючи файлові дескриптори.
  - *Пристрій `/dev/urandom`*: Традиційний інтерфейс ядра Linux. Ядро збирає ентропію з апаратних переривань (натискання клавіш, рух миші), дискових операцій (Block Device IO), мережних пакетів та, за наявності, інструкцій процесора (RDRAND).
  - *Джиптер таймерів (Clock Jitter)*: У рідкісних випадках неможливості використання системних джерел, бібліотека може використовувати варіації часу виконання коду як додаткове джерело ентропії.
- **Пул ентропії:**
  - *На рівні ОС*: Linux підтримує власний ентропійний пул (зазвичай 4096 біт), який постійно переміщується. У сучасних ядрах Linux (5.x+) для цього використовується потоковий шифр ChaCha20.
  - *На рівні PyCrypto*: Якщо використовується клас `Crypto.Random.Fortuna`, бібліотека створює власні 32 пули накопичення. Ентропія розподіляється між ними циклічно, що захищає від передчасного використання недостатньо випадкових даних.

#### 2.1.1 Внутрішня реалізація ГПВП в PyCrypto (Алгоритм Fortuna)

Хоча функція `get_random_bytes()` зазвичай є просто обгорткою над `/dev/urandom`, внутрішня архітектура генератора (клас `Random.new()`) базується на алгоритмі **Fortuna**, розробленому Брюсом Шнайєром та Нільсом Фергюсоном. Ця реалізація складається з таких компонентів:

1. **Акумулятор (Accumulator):** Цей компонент відповідає за збір ентропії. Він містить 32 пули (Pools). Джерела ентропії додають події в ці пули за принципом round-robin. Це зроблено для того, щоб протидіяти атакам, коли джерело ентропії може бути частково контрольованим злоумисником. Пересів (reseeding) генератора відбувається тільки тоді, коли в пулах накопичиться достатньо даних.

2. **Генератор (Generator):** Це частина, що безпосередньо видає псевдовипадкові байти. У PyCrypto він реалізований на основі блочного шифру **AES-256**.

- *AES у режимі лічильника (CTR):* Генератор використовує AES для шифрування лічильника (Counter). Ключ для AES постійно оновлюється (re-keying) після кожного запиту на генерацію даних або після генерації 1 МБ даних. Це гарантує, що навіть якщо поточний стан буде скомпрометовано, зломисник не зможе відновити попередні випадкові числа (Backtracking resistance).

3. **Захист від розгалуження (Fork Safety):** На Linux критично важливою є обробка системного виклику `fork()`. PyCrypto перевіряє PID процесу перед кожною генерацією. Якщо виявлено, що процес був клонований (PID змінився), генератор примусово перезапускається з новим зерном (seed), щоб батьківський і дочірній процеси не генерували однакові ключі.

## 3 Опис досліджуваних функцій

### 3.1 Генерація псевдовипадкових послідовностей (ПВП)

Для генерації криптографічно стійких випадкових чисел у бібліотеці використовується функція `get_random_bytes`.

- **Функція:** `Crypto.Random.get_random_bytes(N)`
- **Алгоритм:** У середовищі Linux функція виступає обгорткою над системним пристроєм `/dev/urandom`. Ядро Linux використовує ентропійний пул, що наповнюється шумами апаратних драйверів, переривань та мережевої активності. Якщо бібліотека не може звернутися до ОС, вона використовує власний PRNG на базі алгоритму Fortuna.
- **Вхідні дані:**
  - `N (int)` — кількість байтів, яку необхідно згенерувати.
- **Вихідні дані:**
  - Рядок байтів (`bytes`) довжиною `N`.
- **Коди повернення / Помилки:** Функція повертає сирі дані або викликає виключення `NotImplementedError`, якщо джерело ентропії недоступне.

Лістинг 1: Скрипт для тестування функцій PyCrypto

```
from Crypto.Random import get_random_bytes
```

```
size=20000
// get 'size' random bytes
data = get_random_bytes(size)
```

## 3.2 Генерація ключів RSA

Для створення пари ключів використовується метод `generate` класу `RSA`.

– **Функція:** `Crypto.PublicKey.RSA.generate(bits, e=65537)`

– **Алгоритм:**

1. Генерація двох великих простих чисел  $p$  та  $q$  заданої бітової довжини.
2. Для перевірки простоти використовується поєднання методу пробних ділень (Trial Division) та ймовірного тесту Міллера-Рабіна.
3. Обчислення модуля  $n = p \cdot q$  та функції Ейлера  $\phi(n)$ .
4. Обчислення секретної експоненти  $d$  такої, що  $d \cdot e \equiv 1 \pmod{\phi(n)}$  за допомогою розширеного алгоритму Евкліда.

– **Вхідні дані:**

- `bits` (int) — бажана довжина модуля ключа в бітах (наприклад, 1024, 2048, 4096). Повинна бути кратною 256.
- `e` (int, необов'язковий) — публічна експонента (за замовчуванням 65537).

– **Вихідні дані:**

- Об'єкт класу `RSAPublicKey`, що містить атрибути  $n, e, d, p, q, u$ .

– **Коди повернення:** Повертає об'єкт ключа або викликає виключення `ValueError`, якщо довжина ключа замала ( $<1024$ ).

## 4 Контрольний приклад роботи

Нижче наведено код програми для статистичного аналізу ПВП та вимірювання часу генерації ключів RSA.

Лістинг 2: Скрипт для тестування функцій PyCrypto

```
from Crypto.PublicKey import RSA
from Crypto.Random import get_random_bytes

bits = 2048
// Generate keypair with size of 'bits'
RSA.generate(bits)
```

## 5 Результати експериментів для швидкодії

В ході лабораторної роботи було проведено серію зі 100 запусків `get_random_bytes` для розмірів 4 кілобайт, 1 мегабайт та 10 мегабайт, які можна побачити у таблиці 1. Також було проведено 50 запусків для кожного розміру ключа. Результати вимірювань ефективності наведено у таблиці 2.

Табл. 1: Результати тестування швидкодії генератора псевдовипадкових послідовностей (Crypto.Random)

Byte size	Average time (s)	Speed (MB/s)
4096	0.00001	287.01
1,048,576	0.00296	338.19
10,485,760	0.02556	391.26

Табл. 2: Часові характеристики та варіативність генерації ключів RSA

Bits	Average (c)	Min (s)	Max (s)	Coefficient of variation (%)
1024 bit	0.1779	0.0528	0.4396	52.75
2048 bit	0.5859	0.1150	2.3475	76.17
4096 bit	5.0829	0.5950	15.2132	71.12

## 5.1 Аналіз якості ПВП

Зі збільшенням розміру вибірок, значення статистики хі-квадрат поводити себе випадково, але не перевищувала порогового значення, а значення ентропії Шенонна наближалась до 'ідеалу', який дорівнює 8. Тому, можемо вважати що генератор ПВП у PyCrypto є надійним генератором.

## 5.2 Аналіз результатів генерації ключів RSA

Можна побачити з результатів, що час генерації зростає експоненційно відносно довжини ключа. При подвоєнні довжини з 2048 до 4096 біт середній час виконання збільшився у 8,7 разів (з 0,59 с до 5,08 с).

Значна різниця між мінімальним і максимальним часом виконання (особливо для 4096 біт: 0,6 с проти 15,2 с) пояснюються тим, що час роботи залежить від кількості спроб, необхідних для знаходження випадкового простого числа. Це є характерною ознакою ймовірнісних тестів простоти.

Ключі довжиною 4096 біт забезпечують вищий рівень безпеки, проте їх генерація може займати значний час (до 15 секунд у пікових випадках). Це робить їх менш придатними для частоті зміни сеансових ключів у реальному часі, але прийнятними для довгострокового зберігання секретів.

## 6 Висновки

У ході роботи досліджено бібліотеку PyCryptoDome. Встановлено, що:

1. Функція `get_random_bytes` забезпечує високу ентропію завдяки використанню системного виклику `/dev/urandom`.
2. Час генерації ключів RSA зростає нелінійно: перехід від 2048 до 4096 біт збільшує час генерації приблизно у 8 разів.
3. Високий коефіцієнт варіації ( $\approx 70\%$ ) є характерною ознакою ймовірнісних алгоритмів пошуку простих чисел, що ускладнює проведення атак по часу.