Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут»

ЗВІТ

з лабораторної роботи № 2

Тема: Реалізація алгоритмів генерації ключів гібридних криптосистем.

з дисципліни «Методи реалізації криптографічних механізмів»

Виконали:
Студенти ФБ-41мн
Зубко Д.Е. та Тивонюк В. І.
«29» жовтня 2025 р

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 гененерації ключів асиметричних криптосистем.

Хід роботи

Теоретичні відомості

Сучасні криптографічні системи значною мірою покладаються на непередбачуваність, яку забезпечують генератори випадкових чисел та ключів. Безпека таких алгоритмів, як RSA, AES, та протоколів на кшталт TLS, напряму залежить від якості ентропії (випадковості), що використовується для створення секретних параметрів.

1. Генератори псевдовипадкових послідовностей (ГПВП)

На відміну від "справжніх" випадкових чисел, які генеруються фізичними процесами, псевдовипадкові послідовності є детермінованими. Вони створюються за допомогою математичного алгоритму, який, починаючи з початкового значення (**"seed"**), генерує послідовність, що виглядає випадковою. Для криптографічних потреб використовуються **криптографічно стійкі ГПВП (CSPRNG)**, які мають дві ключові властивості:

- **Статистична випадковість:** Послідовність проходить стандартні статистичні тести на випадковість (наприклад, тести NIST).
- **Непередбачуваність:** Знаючи попередні біти послідовності, неможливо передбачити наступні.

Найкращою практикою є використання ГПВП, вбудованих в операційну систему, оскільки вони збирають ентропію з багатьох непередбачуваних джерел: рухів миші, натискань клавіш, мережевої активності, апаратних шумів тощо.

2. Генерація криптографічних ключів

Генерація ключів — це процес створення секретних даних для криптографічного алгоритму. Наприклад, для симетричного шифрування (AES) ключ є простою послідовністю випадкових бітів. Для асиметричних алгоритмів (RSA) процес набагато складніший і включає:

- Генерацію одного або кількох великих простих чисел.

- Виконання математичних операцій над цими числами для отримання пари ключів (публічного та приватного).

Пошук великих простих чисел є обчислювально складною задачею. Оскільки детерміновані тести на простоту є надто повільними, на практиці використовуються **імовірнісні тести** (наприклад, тест Міллера-Рабіна). Вони не гарантують 100% простоту числа, але дають ймовірність помилки, яку можна зробити настільки малою (наприклад, меншою за 2^{-128}), що вона стає нехтувано малою.

Бібліотека PyCryptodome

PyCryptodome є сучасним, активно підтримуваним форком застарілої бібліотеки PyCrypto. Вона надає широкий набір криптографічних інструментів і дотримується найкращих практик безпеки, зокрема, покладається на системні джерела ентропії для генерації випадкових даних.

Але ми обираємо **OpenSSL** адже це йде найефективніший варіант.

OpenSSL — це потужна, повнофункціональна криптографічна бібліотека з відкритим кодом, яка є де-факто стандартом для реалізації безпечних мережевих протоколів, таких як TLS/SSL. Вона написана на мові C і надає низькорівневий доступ до широкого спектру криптографічних алгоритмів.

1. Генератор псевдовипадкових послідовностей в OpenSSL

ГПВП в OpenSSL (часто згадується як PRNG - Pseudo-Random Number Generator) є складною системою, що відповідає за надання криптографічно стійких випадкових даних.

- **Джерела ентропії:** OpenSSL автоматично збирає ентропію (випадковість) з різних джерел, специфічних для операційної системи. Для **Windows** основними джерелами:
  - **CryptGenRandom()**: Основна функція з Microsoft CryptoAPI, яка сама збирає ентропію з різних системних джерел (апаратні таймери, дії користувача, мережеві пакети).
  - **RAND_screen()**: Функція, яка (на старих системах) могла використовувати вміст екрану як джерело ентропії.
  - Час високої точності, ID процесу та потоку.
- **Пул ентропії:** Зібрані дані змішуються у внутрішньому пулі ентропії. Коли програма запитує випадкові байти, вони генеруються на основі поточного стану цього пулу за допомогою криптографічної хеш-функції (наприклад, SHA-256). Після

кожного запиту стан пулу оновлюється, що забезпечує непередбачуваність майбутніх значень.

1.1. Внутрішня реалізація ГПВП в OpenSSL

Хоча функція RAND_bytes() є простим інтерфейсом, "під капотом" OpenSSL реалізовано складний механізм, що відповідає стандарту **NIST SP 800-90A**. Він складається з кількох рівнів:

1. **Джерела ентропії (Entropy Sources):** Як було зазначено, OpenSSL збирає "сиру" ентропію з різних джерел. На Windows CryptGenRandom() є основним, але OpenSSL також може додавати власні джерела, такі як RDSEED/RDRAND (якщо підтримується процесором) або таймери високої точності.
2. **Пул ентропії (Entropy Pool):** Це внутрішній буфер, де зібрані випадкові дані постійно "перемішуються" за допомогою криптографічної хеш-функції (наприклад, SHA-256). Це робиться для того, щоб навіть якщо одне з джерел ентропії буде скомпрометоване, його вплив на фінальну послідовність був мінімальним.
3. **DRBG (Deterministic Random Bit Generator):** Це сам алгоритм, який генерує псевдовипадкову послідовність. OpenSSL використовує реалізацію, що базується на хеш-функціях (**Hash_DRBG**) або блочних шифрах (**CTR_DRBG**).

2. Генерація криптографічних ключів в OpenSSL

OpenSSL надає функції для генерації ключів для багатьох асиметричних алгоритмів. Для **RSA** цей процес включає:

- **Пошук простих чисел:** Використовується функція BN_generate_prime_ex, яка генерує великі прості числа. Вона використовує дані з ГПВП OpenSSL та виконує імовірнісний тест на простоту Міллера-Рабіна з великою кількістю ітерацій, щоб мінімізувати ймовірність помилки (тобто що згенероване число насправді є складеним).
- **Створення структури ключа:** Після знаходження двох великих простих чисел p та q, OpenSSL обчислює модуль n, публічну експоненту e та приватну d і зберігає їх у спеціальній структурі RSA.

2.1. Внутрішня реалізація генерації простих чисел для RSA

Ключовим етапом генерації ключів RSA є пошук двох великих простих чисел. OpenSSL використовує для цього функцію BN_generate_prime_ex, яка працює за наступним алгоритмом:

1. **Генерація кандидата:** За допомогою RAND_bytes() генерується випадкове непарне число n заданої довжини (наприклад, 1024 біт для 2048-бітного ключа).

2. **Просіювання (Trial Division):** Кандидат n ділиться на невеликі прості числа (зазвичай, перші кілька тисяч). Якщо він ділиться націло хоча б на одне з них, він відкидається, і генерується новий кандидат. Це швидкий спосіб відсіяти більшість складених чисел.
3. **Імовірнісний тест Міллера-Рабіна:** Якщо число пройшло просіювання, воно піддається серії імовірнісних тестів Міллера-Рабіна.

ПРАКТИЧНЕ ЗАВДАННЯ

Для виконання завдання було написано програму на мові C++, яка використовує бібліотеку OpenSSL для демонстрації роботи двох основних функцій: RAND_bytes для генерації псевдовипадкових послідовностей та RSA_generate_key_ex для генерації ключів RSA.

```
// 1. Аналіз функції генерації ПВП (RAND_bytes)
void test_rand_bytes() {
    std::cout << "--- 1. Аналіз функції генерації ПВП (RAND_bytes) ---" << std::endl;

    const int buffer_size = 16;
    std::vector<unsigned char> buffer(buffer_size);

    int ret = RAND_bytes(buffer.data(), buffer_size);

    std::cout << "Виклик RAND_bytes(" << buffer_size << ")..." << std::endl;
    if (ret == 1) {
        std::cout << "Код повернення: " << ret << " (Успіх)" << std::endl;
        std::cout << "Згенерована послідовність (16 байт): ";
        print_hex(buffer.data(), buffer_size);
    } else {
        std::cerr << "Помилка генерації випадкових байтів! Код: " << ret << std::endl;
    }
}
```

```
// 2. Аналіз функції генерації ключів RSA
void test_rsa_generate() {
    std::cout << "\n--- 2. Аналіз функції генерації ключів RSA ---" << std::endl;

    int key_lengths[] = {1024, 2048, 4096};

    for (int bits : key_lengths) {
        std::cout << "\n--- Генерація RSA ключа довжиною " << bits << " біт ---" << std::endl;

        auto start = std::chrono::high_resolution_clock::now();

        RSA* rsa_key = RSA_new();
        BIGNUM* bne = BN_new();
        BN_set_word(bne, RSA_F4);

        int ret = RSA_generate_key_ex(rsa_key, bits, bne, NULL);

        auto end = std::chrono::high_resolution_clock::now();
        std::chrono::duration<double> duration = end - start;
```

Програма була скомпільована на платформі Windows з використанням Visual Studio 2022 Build Tools (x64 Native Tools Command Prompt) та бібліотеки OpenSSL-Win64.

Команда для компіляції:

```
cl open_ssl2.cpp /EHsc /utf-8 /I "C:\Program Files\OpenSSL-Win64\include" /link /LIBPATH:"C:\Program Files\OpenSSL-Win64\lib\VC\x64\MD" libcrypto.lib libssl.lib Ws2_32.lib Gdi32.lib
```

Компіляція коду:

```
    test_rand_bytes();
    test_rsa_generate();
    return 0;
}
```

```
C:\Users\Volodymyr\Desktop\Assignments\6 year\Крипта>cl open_ssl2.cpp /EHsc /I "C:\Program Files\OpenSSL-Win64\include" /link /LIBPATH:"C:\Program Files\OpenSSL-Win64\lib\VC\x64\MD" libcrypto.lib l
32.lib Gdi32.lib
Microsoft (R) C/C++ Optimizing Compiler Version 19.43.34810 for x64
Copyright (C) Microsoft Corporation.  All rights reserved.

open_ssl2.cpp
Microsoft (R) Incremental Linker Version 14.43.34810.0
Copyright (C) Microsoft Corporation.  All rights reserved.

/out:open_ssl2.exe
"/LIBPATH:C:\Program Files\OpenSSL-Win64\lib\VC\x64\MD"
libcrypto.lib
libssl.lib
Ws2_32.lib
Gdi32.lib
open_ssl2.obj
```

1. **Генератор ПВП:** Функція RAND_bytes успішно повернула 16 випадкових байтів, що підтверджується кодом повернення 1. Це свідчить про коректну роботу з системним джерелом ентропії на платформі Windows.

```
C:\Users\Volodymyr\Desktop\Assignments\6 year\Крипта>open_ssl2.exe
--- 1. Аналіз функції генерації ПВП (RAND_bytes) ---
Виклик RAND_bytes(4096)...
Код повернення: 1 (Успіх)
Згенерована послідовність (4096 байт): fde577adc3b1030e90d84f0e5e59e42cc31899e0d5a0f676
3c6bb0fe69409de96ae30512d14ee4084443109c05ccd49e918eb3f76c9c5b52d7f5bf9311ba2fddf8f2e9e
8469a7867b9ed2751499d88abfeec6a3e52d1a2472817a053aa8036ea09d56cb36b3d318f231fa19139929a
4b821292ecbbac2b3f5d431025042b431fd237a51424ac127a1ef6d241ade0a14813c4b55e945471a24e803
17ae6b68104939918b734a4ef864961c056259fad84b178f5d0662c727be24b19ea4c1fe04e09c819cf6b47
3bfe8af4ae02e03d65510a77862b91e6331d03473fe38b878867fa25d17d2643c7ad670c777d52afe1f8669
24dca27d391fe1091d86aa33a076678bdb0352031b8cfb366cae6123b1625070f11f53adbca3afbe8305cf7
a1999a48af7cbd29a1a0e3eb264daa418fece530b676990002c9f772266560edeacee8bc843cdcaa490df5e
```

- **Ефективність за часом:** Заміри часу показали значне, нелінійне зростання обчислювальної складності зі збільшенням довжини ключа. Якщо 1024-бітний ключ генерується майже миттєво (~8 мс), то для 2048 біт час зростає до ~35 мс (в 4 рази), а для 4096 біт — до ~550 мс (ще в 15 разів). Це наочно демонструє, що подвоєння довжини ключа веде до значно більших витрат часу на пошук відповідних простих чисел

- **Структура ключів:** Було продемонстровано стандартний формат PEM для публічних та приватних ключів. Довжина текстового представлення ключів також суттєво зростає, що є важливим фактором при їх зберіганні та передачі.

```
--- 2. Аналіз функції генерації ключів RSA ---

--- Генерація RSA ключа довжиною 1024 біт ---
Код повернення: 1 (Успіх)
Час генерації: 0.0078881 с

Публічний ключ (формат PEM):
-----BEGIN RSA PUBLIC KEY-----
MIGJAoGBAJlLnRcCAaaCTw5U8WZzK6HhDmqtJbYOxmAOnJip0Hdh8mCFe5nhgxRe
mqDRTnZnnK/K9IF8qhoQX2qRFFbrlEXOQWR1lZFnSdCWWlVRQCqN06FyL3J72KOW
wCWNe/eg/NGlY9nGEPoZxdLpmsWNQ0DiaRxEke8AakVobbDWfpcHAgMBAAE=
-----END RSA PUBLIC KEY-----

Приватний ключ (перші 120 символів):
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQCZS50XAgGmgk8OVPFmcyuh4Q5qrSW2DsZgDpyYqdB3YfJghXuZ
4YMUXpqg0U52Z5yvyvSBfKo...
```

```
--- Генерація RSA ключа довжиною 2048 біт ---
Код повернення: 1 (Успіх)
Час генерації: 0.0840776 с

Публічний ключ (формат PEM):
-----BEGIN RSA PUBLIC KEY-----
MIIBCgKCAQEAxDWkddgSUzvFdUYJOwRbsvhzhmrsBZI6m9lg8NJ5ekMXzE2q48P2
Bv3iFviyMDFr8fJbFDr+BxwQzolOQ5rqfY8ZKuxt9yVXrxm1KNyrCCaVuNTp5MQg
vAvLPfRSIR3Bq8lBAGEr2qmSnyInZ7rtjZWkrTmUVbyll8EEWecJbgrlxaUo0hIy
EFMBm21vMPNlwIKSrpOYQ8ZR6ai8ldbTS+BOxDiRgLgTZUq5tg//u8PqQqb6uaF/
unipUKBYeAuo3dFw22x+fDM4DCrI2E3zwUOS/2P0qsbrnpp6mkfmTRRzB8jCu4up
lDm3L4RKK5GyppcwJrUe4cI3DY6ftkDTSwIDAQAB
-----END RSA PUBLIC KEY-----

Приватний ключ (перші 120 символів):
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAxDWkddgSUzvFdUYJOwRbsvhzhmrsBZI6m9lg8NJ5ekMXzE2q
48P2Bv3iFviyMDFr8fJbFDr...

--- Генерація RSA ключа довжиною 4096 біт ---
Код повернення: 1 (Успіх)
Час генерації: 0.401162 с

Публічний ключ (формат PEM):
-----BEGIN RSA PUBLIC KEY-----
MIICCgKCAgEAz9tdQ0Np3HAoOsj3lw6d1+Sfuj+r+XiNO5DlUZ3bdJu4H14/c2yb
zk5MDztOiNJyCpOzXneEtnjCGPg8fcpS00VzK4LIl6abxHs3IzCrG+i2iHHF17Ns
BYDFbf9mk0cumUvy+IiXIkzc9QMO2r2bV506ygqXx9eIFG7XMwiNQKARQ0jbMhTe
L0GQztYRIkoY4DAwHVZA7wELC5/wTQG1mGqqukDHKtgGXCRb82gVfLFx3EZhVANr
DjvJr3Q749feT/JnvWluMR8wRkX9W9vpBiGfyQsNjnWlHwyF8UHHpHGfmyjzklot
N1f/BhJ7S4mjDb1Gh/WTon1o0tfyif6rZVbb7wSbusM/iySzzvMmNEtPUWwOvOzi
6aIlO7+GlecqrfJ3ocLPoYAG/AAU+gVnPl1/NYNHQyDtRv9/VOWX84P0lcu5KJ4s
V6GFXgrR42gsOvaOitURrXL46WQepdfVC9Dx8Iy6PJU/bVu7BmvOsS4ZpVDTLeuX
maC9j02pr65Nf2eR/WtZvaUtQz02wIB+oGMFtFm1lhFcld2VqfNvqyWXH66ZQUmC
NEUFpeJ6H/YJ7ygqhWmvgEpIZ4fTBnpImVLuLN+PQX7mHJe/W0FArnZmczYCGXX8
xNmdO8qEUDCL4CzABjOSJybSlUj11McLvqblOBUe3a/0TyY1qwayK7MCAwEAAQ==
-----END RSA PUBLIC KEY-----

Приватний ключ (перші 120 символів):
-----BEGIN RSA PRIVATE KEY-----
MIIJJwIBAAKCAgEAz9tdQ0Np3HAoOsj3lw6d1+Sfuj+r+XiNO5DlUZ3bdJu4H14/
c2ybzk5MDztOiNJyCpOzXne...
```

Інша ітерація тестування, з замірами середнього часу:

```
1.1. Тест на рівномірність розподілу байтів
Очікувана частота для кожного байта: 78.125
Статистика Хі-квадрат: 270.822 (для 255 ступенів свободи, очікується ~255)
Результат: Розподіл виглядає рівномірним (ТЕСТ ПРОЙДЕНО).

1.2. Базовий тест на повторювані патерни (диграми)
Максимальна частота повторення пари байтів: 4
Результат: Очевидні патерни не виявлено (ТЕСТ ПРОЙДЕНО).

--- 2. Порівняння продуктивності генерації ключів RSA ---

--- Тестування ключа 1024 біт (10 запусків) ---
Середній час генерації: 0.00882694 с
Стандартне відхилення: 0.00248702 с

--- Тестування ключа 2048 біт (10 запусків) ---
Середній час генерації: 0.0424839 с
Стандартне відхилення: 0.0265722 с

Приклад згенерованого ключа (2048 біт):
-----BEGIN RSA PUBLIC KEY-----
MIIBCgKCAQEAsK6brAwaQpO1DxrFOqvP3rfptClsFmEQ6nRzdOIUjxYkBv7ipkqg
Yofk+igNzN6URU8WDxJoBLJ9+dm5H1aY6f7P6Cu1fp8SUNM9qhZTUGlRnD9qvr0z
P1zFAAZItBll8RzapeGEtfStx0+BDRCeU5XpT+TbIDE0SxblcyaMDDXEQI0l2aqx
p98NWfDtCnZBk1s758pT3PCqNBU6ue4krFlsOk4o9Q564lrSsl41jnabv+cosuJL
w0a/EZemcPBY1FZdcC9i7E+/keZIhJMVpuiWXQA/x5W7ubrn+kqQhaJJO+BUWwyc
xyH9gW6c7EPI+Y+zhsMrZB6DsRQKSp6CCQIDAQAB
-----END RSA PUBLIC KEY-----


--- Тестування ключа 4096 біт (10 запусків) ---
Середній час генерації: 0.478124 с
Стандартне відхилення: 0.147881 с
```

Вивід статистики:

```
  Генерація RSA ключа довжиною 2048 біт

  Запуск 1: 0.070064 с
  Запуск 2: 0.022851 с
  Запуск 3: 0.019257 с
  Запуск 4: 0.052814 с
  Запуск 5: 0.037763 с

--- Статистика ---
Середній час: 0.040550 с
Стандартне відхилення: 0.018952 с
Мінімальний час: 0.019257 с
Максимальний час: 0.070064 с
Коефіцієнт варіації: 46.737594%
```

Висновки

У ході лабораторної роботи було успішно досліджено основні функції генерації випадкових даних та ключів RSA в бібліотеці OpenSSL для платформи Windows

Практичні приклади підтвердили, що OpenSSL є потужним та ефективним інструментом для реалізації криптографічних механізмів, проте вибір параметрів безпеки (зокрема, довжини ключа) вимагає врахування компромісу між криптостійкістю та продуктивністю системи.

Аналіз стійкості показав, що ГПВП RAND_bytes генерує якісні випадкові послідовності, що було підтверджено базовими статистичними тестами. Дослідження ефективності генерації ключів RSA наочно продемонструвало експоненційну залежність часу виконання від довжини ключа. Зокрема, було встановлено, що подвоєння довжини ключа з 2048 до 4096 біт збільшує час генерації більш ніж в 11 разів. Це підкреслює фундаментальний компроміс у криптографії: вищий рівень безпеки вимагає значно більших обчислювальних ресурсів та призводить до менш передбачуваного часу виконання.

Однак процес генерації ключів RSA, хоч і безпечний, є обчислювально дорогим та, що більш важливо, **нестабільним за часом виконання**. Ця непередбачуваність продуктивності є критичним фактором, який необхідно враховувати при проектуванні систем, де час генерації ключів може впливати на доступність сервісу (наприклад, при встановленні великої кількості TLS-з'єднань). Для таких систем може знадобитися попереднє генерування пулу ключів або вибір менш ресурсоємних асиметричних алгоритмів.