**Міністерство освіти і науки України**
**«Київський національний університет імені Тараса Шевченка»**

**Факультет інформаційних технологій**
**Кафедра кібербезпеки та захисту інформації**

ДОПУСТИТИ ДО ЗАХИСТУ:
завідувач кафедри кібербезпеки
та захисту інформації
_____Н.В. Лукова-Чуйко
«  » червня 2021р.

# ПОЯСНЮВАЛЬНА ЗАПИСКА
## дипломної роботи
***бакалавра .***
(назва освітнього рівня)

**галузь знань** 12 Інформаційні технології
(шифр і назва галузі знань)

**спеціальність** 125 Кібербезпека
(код і назва спеціальності)

**освітня програма** Кібербезпека
(назва освітньої програми)

**на тему:** «Розробка схеми криптографічного генератора, заснованого на комбінуванні конгруентних генераторів, з оцінкою його якості»

**Виконавець:** студент IV курсу, групи КБ-42

Іваночко Іван Михайлович
(підпис) (прізвище ім'я по-батькові)

| | **Прізвище, ініціали** | **Підпис** |
|---|---|---|
| **Керівник** | Фесенко А.О. | |

| | | |
|---|---|---|
| **Нормоконтроль** | Зюбіна Р. В. | |

**Київ 2021**

**Міністерство освіти і науки України**
**«Київський національний університет імені Тараса Шевченка»**

**Факультет інформаційних технологій**
**Кафедра кібербезпеки та захисту інформації**

**ЗАТВЕРДЖЕНО:**
завідувач кафедри кібербезпеки
та захисту інформації
_____Н.В. Лукова-Чуйко
«10» жовтня 2020 р.

# ЗАВДАННЯ
## на виконання дипломної роботи

**спеціальності** 125 Кібербезпека
(код і назва спеціальності)

**освітньої програми** Кібербезпека
(назва освітньої програми)

**Студенту** **КБ-42** **Іваночко Іван Михайлович**
(група) (прізвище ім'я по-батькові)

**Тема дипломної роботи** Розробка схеми криптографічного генератора, заснованого на комбінуванні конгруентних генераторів, з оцінкою його якості

**1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ**

Тема дипломної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №2 від 08.10.2020 р.

**2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ**
Статистичні властивості конгруентної послідовності, кращі практики при створені ГПВП

**3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ**
Проаналізувати властивості конгруентної послідовності. Роздивитися інші генератори послідовностей для знаходження їх сильних сторін. Покращити якість вихідної послідовності шляхом створення генератора псевдовипадкових чисел заснованого на комбінуванні конгруентних генераторів.

## 4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

**Практична цінність** Створений генератор псевдовипадкових послідовностей та була надана йому оцінка.

## 5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання:12 жовтня 2020 року

Завдання видав _____ А. О. Фесенко

Завдання прийняв до виконання _____ (підпис) І.М. Іваночко (ініціали, прізвище)

**КАЛЕНДАРНИЙ ПЛАН**

| № п/п | Найменування етапів робіт | Строки виконання робіт (початок-кінець) | Відмітка про виконання |
|---|---|---|---|
| 1 | Уточнення постановки задачі | 25.01.2021 – 29.01.2021 | *виконано* |
| 2 | Аналіз літератури | 29.01.2021 – 22.02.2021 | *виконано* |
| 3 | Обґрунтування вибору рішення | 22.02.2021 – 26.02.2021 | *виконано* |
| 4 | Аналіз існуючих алгоритмів генерації випадкових послідовностей | 26.02.2021 – 14.03.2021 | *виконано* |
| 5 | Сфери застосування псевдовипадкових послідовностей | 15.03.2021 – 08.04.2021 | *виконано* |
| 6 | Дослідження статистичних властивостей випадкових послідовностей | 09.04.2021 – 19.04.2021 | *виконано* |
| 7 | Розробка генератора | 20.04.2021 – 10.05.2021 | *виконано* |
| 8 | Оформлення пояснювальної записки | 11.05.2021 – 08.06.2021 | *виконано* |
| 9 | Підготовка до захисту дипломної роботи | 09.06.2021 – 21.06.2021 | *виконано* |

Завдання видав _____ (підпис) А.О. Фесенко (ініціали, прізвище)

Завдання прийняв до виконання _____ (підпис) І. М. Іваночко (ініціали, прізвище)

Термін подання дипломної роботи до ЕК 08 червня 2021 року

# РЕФЕРАТ

Дипломна робота обсягом 85 аркушів складається із вступу, трьох розділів, висновків та переліку посилань із 12 джерел. Містить 59 рисунків та 2 таблиці. Обсяг додатків складає 6 сторінок.

Метою даної роботи є створення генератора псевдовипадкових послідовностей з оцінкою його вихідних послідовностей.

В даній роботі були розглянуті властивості конгруентної послідовності, підбір правильних модифікаторів до неї, криптографічна стійкість конгруентного методу.

Крім того були описані методи запобігання компрометування таблиці значень для криптографічного генератора, розглянуті «слабкі» вхідні значення для конгруентної послідовності.

Ключові слова: криптографічний генератор, конгруентна послідовність, генерація випадкових чисел, псевдовипадкова послідовність, параметри генератора.

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

| | | |
|---|---|---|
| ПВП | – | псевдовипадкова послідовність |
| ГПВЧ | – | генератор псевдовипадкових чисел |
| ЛКП | – | лінійна конгруентна послідовність |
| ЛКМ | – | лінійний конгруентний метод |
| ГПВП | – | генератор псевдовипадкових послідовностей |

# ЗМІСТ

# ВСТУП

*Актуальність теми дослідження.* Випадкові числа грають важливу роль в криптографії та її різних додатках. Двома основними вимогами до послідовності випадкових чисел є випадковість і непередбачуваність. Розглянемо по порядку.

1. Випадковість.

Зазвичай при створенні послідовності псевдовипадкових чисел передбачається, що дана послідовність чисел повинна бути випадковою в деякому певному статистичному сенсі. Для доказу того, що дана послідовність є випадковою, використовуються 2 критерія:

- Однорідний розподіл: розподіл чисел в послідовності має бути однорідним; це означає, що частота появи кожного числа повинна бути приблизно однаковою.

- Незалежність: жодне значення в послідовності не повинно залежати від інших.

Хоча існують тести, які показують, що послідовність чисел відповідає деякому розподілу, такому як однорідний розподіл, тесту для "доказу" незалежності немає. Проте, можна підібрати набір тестів для доказу того, що послідовність є залежною. Загальна стратегія передбачає застосування набору таких тестів до тих пір, поки не буде впевненості, що незалежність існує.

2. Непередбачуваність.

При "правильній" випадковій послідовності кожне число статистично не залежить від інших чисел і, отже є непередбачуваною. Тобто неможливо, знаючи попередні елементи послідовності і алгоритм, передбачити наступний елемент.

Послідовність називається по-справжньому випадковою, якщо її не можна відтворити. Це означає, що якщо запустити генератор по-

справжньому випадкових послідовностей двічі при одному і тому ж вході, то на його виході вийдуть різні випадкові послідовності.

Джерела таких випадкових чисел знайти важко. Фізичні генератори шумів, такі як детектори подій іонізуючої радіації, газові розрядні трубки конденсатор, який протікає, можуть бути такими джерелами. Однак застосування цих пристроїв в криптографії обмежено. Проблеми також викликають грубі атаки на такі пристрої. Альтернативним рішенням є створення набору з великого числа випадкових чисел і опублікування його в деякій книзі. Проте, і такі набори забезпечують дуже обмежене джерело чисел в порівнянні з тією кількістю, яка потрібна. Більш того, хоча набори з цих книг дійсно забезпечує статистичну випадковість, вони передбачувані, так як легко можна отримати їх копію.

Таким чином, шифрувальні додатки використовують для створення випадкових чисел спеціальні алгоритми. Ці алгоритми детерміновані і, отже, створюють послідовність чисел, яка не є статистично випадковою. Проте, якщо алгоритм хороший, отримана послідовність буде проходити багато тестів на випадковість. Такі числа часто називають псевдовипадковими числами.

Створенню хороших генераторів псевдовипадкових послідовностей приділяється чимала увага в математиці. Проблема в тому, що всі генератори псевдовипадкових послідовностей за певних умов дають передбачувані результати і кореляційні залежності.

Випадкові числа та їх генератори є невід'ємними елементами сучасних криптосистем. Наведемо конкретні приклади використання випадкових чисел в криптографії:

1. сеансові і інші ключі для симетричних криптосистем, таких як DES, ГОСТ 28147-89, Blowfish.

2. стартові значення для програм генерації ряду математичних величин в асиметричних криптосистемах, наприклад RSA, ELGamal.

3. випадкові слова, комбіновані з парольний словами для порушення

"атаки вгадування" пароля криптоаналітиків.

4. вектор ініціалізації для блокових криптосистем, що працюють в режимі зворотного зв'язку.

5. випадкові значення параметрів для багатьох систем електронного цифрового підпису, наприклад DSA або СТБ 1176.2-99.

6. випадкові вибори в протоколах аутентифікації, наприклад в протоколі Цербер.

7. випадкові параметри протоколів для забезпечення унікальності різних реалізацій одного і того ж протоколу, наприклад в протоколах SET і SSL

*Метою роботи* є створення криптографічного генератора та підвищення якості псевдовипадкових послідовностей шляхом покращення їх статистичних властивостей.

*Об'єктом дослідження* є процес формування конгруентних послідовностей псевдовипадкових чисел.

*Предметом дослідження* є методи та засоби формування конгруентних послідовностей псевдовипадкових чисел.

*Методи досліджень*. Для вирішення задачі розробки комбінаційного методу формування ПВП перестановками чисел у якості первинних генераторів використовувались методи: статистичного аналізу, комп'ютерної криптографії.

# РОЗДІЛ 1
# АНАЛІЗ ІСНУЮЧИХ АЛГОРИТМІВ ГЕНЕРУВАННЯ ПОСЛІДОВНОСТЕЙ ТА ЇХ ТЕСТУВАННЯ

## 1.1 Сфери застосування випадкових та псевдовипадкових послідовностей

Історично випадкові числа почали використовуватися для проведення вибіркових спостережень, замість безперервних. Випадкові числа застосовуються при вирішенні складних обчислювальних задач і реалізації обчислювальних методів (наприклад, метод Монте-Карло). Розвиток електронно обчислювальної техніки, з одного боку, розширив коло завдань, що використовують випадкові числа, а з іншого - пред'явило високі вимоги до якості їх генерації. Згодом випадкові числа стали відігравати важливу роль в інформатиці, розподілених обчисленнях, криптографії та інших областях.

Послідовність чисел називається випадковою, якщо відтворити її, знаючи алгоритм і всі вхідні дані, не представляється можливим (двічі запустивши генератор в тих же умовах, ми отримаємо різні послідовності). Але комп'ютерні системи детерміновані, тобто для них характерний строго певний набір станів (кількість таких станів може бути досить великою, але кінцевою). Це призводить до того, що послідовності, які ними генеруються, будуть періодичні і відтворювані - такі послідовності називаються псевдовипадковими. Як відомо, все періодичне є в тій чи іншій мірі передбачуваним, тобто невипадковим. Отримання істинно випадкових послідовностей є досить трудомістким. До того ж для їх створення підходить далеко не кожен фізичний або інформаційний процес.

Випадкові та псевдовипадкові послідовності знаходять застосування в самих різних сферах людської діяльності. Нижче наведено перелік найбільш

відомих напрямків, в яких послідовності застосовуються найбільш інтенсивно.

1. *Криптографія*. Криптографічні методи є базовими в забезпеченні інформаційної безпеки. У криптографії випадкові послідовності відіграють визначну роль. Вони використовуються, зокрема, для отримання ключової послідовності використовуваного алгоритму шифрування, для генерації гами поточних шифрів, а також для вироблення векторів ініціалізації (блокових шифрів).

2. *Інші напрямки захисту інформації.* Випадкові послідовності незамінні при формуванні паролів і користувацьких ключів (надійний пароль це - послідовність випадкових символів). Крім цього, вони можуть використовуватися для внесення невизначеності в результати роботи різних алгоритмів захисту інформації. Вони також необхідні при формуванні випадкових запитів, при аутентифікації і вирішенні багатьох інших задач.

3. *Тестування алгоритмів.* Важливим завданням є перевірка правильності роботи програм. Тестування - досить довгий і трудомісткий процес. Для його здійснення потрібен великий обсяг вхідних даних. Використання генераторів випадкових чисел підвищує ефективність тестування і дозволяє економити час.

4. *Мережеві протоколи.* Випадкові послідовності можуть використовуватися, наприклад, в якості сесійних ключів, а також для вироблення випадкових параметрів протоколу, що забезпечує унікальність його різних реалізацій.

5. *Математичне і імітаційне моделювання.* При моделюванні складних фізичних, технологічних і соціально-економічних систем і процесів, обійтися без застосування джерел випадковості не представляється можливим.

6. *Математична статистика.* Математична статистика вивчає наближені методи збору та аналізу даних за результатами експерименту для виявлення існуючих закономірностей, тобто знаходження законів розподілу випадкових величин і їх числових характеристик. Необхідною складовою

вибіркових методів є формування представницьких вибірок з генеральної сукупності з використанням випадкових чисел.

Також випадкові послідовності часто використовуються в теорії чисел; статистичної фізики; прогнозуванні; методах обчислень (в тому числі метод Монте-Карло); теорії управління; інформаційних технологіях для банківських, платіжних, торговельних систем; завадостійкого кодування; автономному і вбудованому діагностуванні компонентів комп'ютерних систем; модуляції радіосигналів; в контролі ходу виконання програм з використанням сторожових процесорів; індустрії ігор і лотерей.

## 1.2 Застосування випадкових та псевдовипадкових послідовностей для захисту інформації

Однією зі сфер діяльності, в яких випадкові послідовності відіграють важливу роль, є захист інформації.

Випадкові послідовності, що застосовуються в різних аспектах захисту інформації, використовуються для вирішення наступних завдань:

- генерація гаммуючих послідовностей при потоковому шифруванні інформації за схемою, найбільш близькою до схеми абсолютно стійкого шифру;

- формування векторів ініціалізації для блокових шифрів, що працюють в режимі зворотного зв'язку;

- отримання початкових значень для програм генерації деяких параметрів в асиметричних криптосистемах;

- формування користувацьких ключів і паролів;

- формування ключової інформації, на секретності і якості якої ґрунтується стійкість більшості криптоалгоритмів;

- формування випадкових запитів при реалізації великого числа криптографічних протоколів, наприклад, протоколів вироблення загального

секретного ключа, поділу секрету, прив'язки до біту, аутентифікації, електронного підпису та ін.;

- внесення невизначеності в роботу засобів захисту, наприклад, при реалізації концепції імовірнісного шифрування, при якому одному і тому ж початковому тексту, при одному і тому ж ключі відповідає безліч шифротекстів;

- виконання статистичного тестування;

- формування затемняючих множників при сліпому шифруванні (протокол сліпого підпису).

Особливе місце займає використання випадкових чисел в криптографії - одному з найпотужніших і найефективніших методів захисту інформації. У криптографії ключову роль відіграють послідовності бітів - двійкові послідовності, що складаються з випадково розділених значень «0» і «1».

Для ряду криптографічних перетворень використовують випадкові первинні стани або цілі послідовності. Отже, стійкість криптоалгоритму, що використовує такі стани або послідовності, безпосередньо залежить від алгоритму генерації випадкових чисел і послідовностей, точніше від ступеня випадковості вихідних послідовностей.

Широта і важливість областей застосування випадкових послідовностей, їх визначальна роль в забезпеченні високого рівня захисту інформації обумовлюють актуальність їх вивчення.

Процес генерації випадкових чисел є основною частиною багатьох криптографічних операцій. Наприклад, криптографічні ключі повинні вибиратися настільки випадково, наскільки це можливо, щоб на практиці не можна було відтворити їх значення. Криптографічно стійкі генератори випадкових чисел повинні видавати дані, які неможливо передбачити з вірогідністю вище 0,5; це означає, що будь-який метод передбачення чергового вихідного біта не повинен діяти ефективніше, ніж просто випадкове вгадування.

## 1.3 Методи отримання псевдовипадкових послідовностей

В даний час існує велика кількість способів генерації послідовностей, що володіють тим або іншим ступенем випадковості [1-4]. Однак на практиці більшість із таких генераторів виробляють послідовності, властивості яких не задовольняють вимогам випадковості. Один із найпоширеніших прикладів цього - генератори псевдовипадкових чисел, вбудовані в стандартні бібліотеки багатьох мов програмування (такі, як, наприклад, функція стандартної бібліотеки мови C rand ()). Часто в числах, згенерованих за допомогою подібних функцій, простежуються явні закономірності. Наприклад, отримані числа в одному і тому ж сеансі з плином часу монотонно зростають, що прямо суперечить вимогам, що пред'являються до властивостей випадкових (і псевдовипадкових) послідовностей.

Більшість криптографічних додатків використовують генератори випадкових чисел для створення ключів, за допомогою яких шифрується і розшифровується потрібна інформація. Однак часто саме застосовувані в них генератора є найслабшим місцем в системах шифрування. Справа в тому, що програмні генератори повністю детерміновані. Зазвичай вони використовують різні складні функції для обчислення псевдовипадкових чисел. Відповідно, послідовності, отримані в результаті роботи таких генераторів, є в тій чи іншій мірі передбачуваними і відтворюваними і не підходять, наприклад, для використання в криптографічних додатках. Необхідно відзначити, що в деяких випадках можливість відтворити випадкову послідовність є корисною (наприклад, при тестуванні алгоритмів розробником). Проте, послідовність не повинна мати властивості, які дозволили б криптоаналітику відновити її в процесі аналізу роботи захищеного додатка або протокола.

Існує табличний спосіб генерації випадкових послідовностей. Він полягає втому, що випадкові числа оформлені у вигляді таблиці, паперової або електронної, яка зберігається в оперативній пам'яті або на зовнішньому

носії. Один з варіантів таблиці випадкових чисел і способу їх вибору описаний в ГОСТ Р ІСО 24153-2012 (Статистичні методи. Процедури рандомізації та відбору випадкової вибірки). Основний плюс цього методу полягає в тому, що з його допомогою можна відтворювати неодноразово одну і ту ж послідовність псевдовипадкових чисел. Однак серйозним недоліком, фактично не допускаючий застосування таких генераторів при вирішенні практичних завдань, є те, що запас вільних номерів обмежений. Також при такому підході можливо неефективне використання обчислювальних ресурсів комп'ютера (наприклад, через необхідність зберігати таблицю або її частини в оперативній пам'яті або звертатися до зовнішньої пам'яті). В даний час такий спосіб генерації використовується досить рідко.

Серед генераторів псевдовипадкових послідовностей, що набули широкого поширення і застосування при вирішенні завдань з серйозними вимогами до якості згенерованої послідовності, розрізняють апаратні, програмні та програмно-апаратні (змішані).

Апаратний генератор випадкових чисел це пристрій, який генерує послідовність випадкових чисел на основі вимірюваних, хаотично змінюючихся параметрів протікаючого фізичного процесу. При апаратному способі генерації, випадкові числа є прямим або побічним продуктом вимірювань деякої фізичної величини, яка є надійним джерелом ентропії. Зазвичай це процеси, що протікають в неживій природі. Теоретично такі процеси абсолютно непередбачувані, однак на практиці отримані таким чином випадкові числа доводиться піддавати перевірці за допомогою спеціальних статистичних тестів. Незважаючи на кращі статистичні властивості і, відповідно, більш високу ступінь випадковості, апаратним генераторам притаманні такі недоліки:

- потенційно високі тимчасові і матеріальні витрати на конструювання, встановлення та налаштування в порівнянні з програмними ГПВЧ;

- більш низька швидкість генерації випадкових чисел, ніж при програмній реалізації ГПВЧ;
- неможливість відтворення раніше згенерованої послідовності (що в деяких випадках є небажаним) .

Програмні (алгоритмічні) генератори (генератори псевдовипадкових послідовностей) засновані на детермінованих алгоритмах. У отриманих таким чином послідовностей завжди існує період (нехай іноді і дуже великий), а також спостерігаються і інші відхилення від випадковості. Будь який ГПВЧ з обмеженими ресурсами рано чи пізно зациклюється - починає повторювати одну і ту ж послідовність чисел. Період ГПВЧ залежить від типу генератора і його параметрів. Якщо породжувана послідовність ГПВЧ має занадто короткий період, то такий ГПВЧ стає непридатним для багатьох практичних додатків.

Більшість простих арифметичних генераторів хоча і володіють великою швидкістю, але страждають від багатьох серйозних недоліків:

- досить короткий період;
- послідовні значення не є незалежними;
- де-які біти «менш випадкові», ніж інші;
- нерівномірний розподіл;
- оборотність.

Фактично, результат роботи таких генераторів не є випадковою послідовністю. Проте, до послідовностей, виробленим програмними генераторами, пред'являються певні вимоги, оскільки вони повинні в якійсь мірі імітувати випадкові послідовності. Зокрема, період таких послідовностей повинен бути досить великим, щоб при генерації послідовності необхідної довжини не виникало повторень. На відміну від апаратних генераторів, програмні генератори здатні відтворити раніше згенеровану послідовність, що в деяких випадках є безперечною перевагою.

Програмно-апаратний генератор може формувати потік випадкових шумів, які потім перетворюються в числа. Також можливий варіант, коли

«зерно» (тобто якісь вхідні дані алгоритму шифрування) генерується за допомогою апаратного генератора (оскільки її розмір досить невеликий, і, відповідно, її отримання не вимагає великих витрат часу і ресурсів), а підсумкова послідовність - за допомогою программного.

До програмно-апаратних генераторів випадкових чисел можуть відноситися, наприклад, пристрої комп'ютера. Зокрема, джерелом випадкової послідовності можуть бути шуми пристроїв комп'ютера (наприклад, процесора), системний час, тимчасові інтервали між натисканнями клавіш, руху миші і так далі. Як правило, послідовності, отримані в результаті таких процесів, потребують постобробці. До того ж швидкість їх отримання є досить низькою (особливо при генерації послідовностей досить великих обсягів). До таких генераторів можна віднести, зокрема, псевдопристрої /dev/random та /dev/urandom ОС Linux.

Послідовність називається істинно випадковою (ІВП), якщо її не можна відтворити. Це означає, що якщо запустити генератор істинно випадкових послідовностей двічі при одному і тому ж вході, то на його виході вийдуть різні випадкові послідовності. Основна складність полягає в тому, щоб зуміти відрізнити випадкову послідовність від невипадкової.

Однак на практиці далеко не завжди можна безпосередньо використовувати вихідні дані джерел істинно випадкових чисел. Тому зазвичай доводиться використовувати так звані псевдовипадкові послідовності. Псевдовипадкова послідовність - це послідовність, що складається з псевдовипадкових двійкових чисел, одержуваних за допомогою заданого детермінірованного алгоритму, але застосовуються в якості випадкових. При цьому зазвичай алгоритми отримання ПВП використовують спеціальне випадкове початкове значення, або «зерно» (seed). Для того щоб ПВП мали змогу використовуватися в якості випадкових послідовностей, вони повинні по статистичних властивостях бути близькі до ІВП. В таблиці 1.1 наведено порівняння основних характеристик обох типів випадкових послідовностей.

*Таблиця 1.1*

Характеристики ІВП та ПВП

| Характеристика | Випадкові послідовності | Псевдовипадкові послідовності |
|---|---|---|
| Відсутність періодичності | Так | Ні |
| Непередбачуваність | Так | Умовна |
| Незалежність значень | Так | Умовна |
| Рівень криптостійкості | Високий | Умовний |
| Швидкість генерації | Низька | Висока |
| Відтворюваність | Ні | Так |
| Простота генерації | Ні | Так |
| Ціна генерації | Висока | Низька |

Ми знаємо, що на мікрорівні випадковість існує (квантова механіка), але невідомо, чи зберігається ця випадковість при переході на макрорівень. Додаткова властивість випадкової послідовності полягає в тому, що випадкова послідовність не може бути стиснута [4].

Вимоги до якісного генератору випадкових чисел [3]:

- Непередбачуваність результатів роботи: при невідомому ключі / початковому стані генератора на основі відомої кінцевої частини ПВП неможливо визначити як її наступний елемент (пряма непередбачуваність, або непередбачуваність вправо), так і попередній (зворотна непередбачуваність , непередбачуваність вліво);
- Великий період послідовності.
- Можливість ефективної апаратної та програмної реалізації.

На практиці домогтися виконання всіх цих умов, як правило, не представляється можливим. Більш того, часто ці умови є взаємовиключними.

Тому доводиться шукати баланс між ними і в першу чергу прагнути до виконання того, що є найбільш важливим в контексті розв'язуваної задачі.

Розглянемо декілька генераторів псевдовипадкових послідовностей таких як: алгоритм середини квадрата, лінійний конгруентний метод, адитивні ГПВП та генератор на регістрах зсуву. Проаналізуємо ці генератори для знайдення «сильних» сторін кожного, та подальшого використання у нашому генераторі.

### 1.3.1 Алгоритм середини квадрата

Історично одним з перших ГПВП був запропонований в 1946 р Дж. Фон Нейманом алгоритм «середини квадрата», що складається з наступних кроків:

1. Вибирається n-розрядне початкове випадкове раціональне десяткове число $x_i$ (його джерелом може бути ГВП).
2. На кожному наступному кроці обчислюється квадрат $y_i$ цього числа (2n-розрядне число).
3. В якості чергового випадкового n-розрядного числа $x_{i+1}$ вибирається n середніх розрядів квадрата попереднього числа $y_i$.

Однако отримані таким способом числа виявляються пов'язаними між собою. Крім того, послідовність цих чисел має малий період.

### 1.3.2 Лінійний конгруентний метод

В генераторах такого типу формування ПВП виконується за допомогою наступного рекурентного алгоритму:

$$x_{n+1} = (a * x_n + c) mod\, m. \quad (1.4)$$

тобто черговий елемент ПВП вираховується застосуванням операції ділення по модулю n до лінійно перетвореного попереднього елемента ПВП $x_n$. Параметрами лінійного конгруентного генератора (ЛКГ) є:

- модуль m;
- множник (мультиплікатор) a;
- приріст (інкремент) c;
- початкове значення (seed) x0.

Параметри a, c, $x_i$ - цілі числа з відрізка [0, m ]. Від їх значень істотно залежить якість отриманої ПВП. При невдалих поєднаннях параметрів можна отримати послідовність з малим періодом.

Відомо [5], що вихідні послідовності ЛКГ мають так звану «решітчату» структуру, тобто є легко передбачуваними і не можуть застосовуватися в криптографічних цілях.

Доказано [5], що формула (1.4) лінійної конгруентної послідовності має максимальну довжину періоду m тоді і тільки тоді, коли:

- c та m є взаємно простими числами;
- число (a-1) кратно деякому простому дільнику m;
- число (a-1) кратно 4, якщо число m кратно 4.

Слід помітити, що виконання зазначених умов забезпечує максимальний період ПВП, але не гарантує її якість.

Для відкидання ЛКГ, що породжують свідомо невипадкові послідовності, використовують поняття потенціалу s лінійної конгруентної послідовності з максимальним періодом - найменшого цілого числа, такого, що $(a-1)^s = 0 \; mod \, m$ [4]. При s = {1; 2} наступні елементи отриманої ПВП лінійно залежать від попередніх. Такі послідовності позбавлені необхідних властивостей ПВП. У разі s = {3; 4} послідовність більш схожа на ПВП, але будь-який її елемент як і раніше істотно залежить від сусідніх з ним. Вважається, що починаючи з s = 5 послідовності ЛКГ можуть мати достатній ступінь випадковості (що вимагає підтвердження за допомогою різних статистичних тестів, тому що велике значення потенціалу це тільки необхідна, але не достатня умова випадковості ПВП).

В [4, 5] наводяться набори параметрів для побудови ЛКГ максимального періода. Якщо інкремент c = 0, формула (1.4) набуває виду:

$$x_{n+1} = a * x_n \, mod \, m. \quad (1.5)$$

Приватним випадком такого ЛКГ є генератор Парка-Міллера, параметри якого a, c, m відповідно дорівнюють 75, 0 і $2^{31} - 1$ = 2147483647.

Розвитком лінійного конгруентного генератора є поліноміальні генератори (квадратичний, кубічний, довільного ступеня). Для таких генераторів у формулі (1.4) лінійний вираз замінюється поліномом потрібного ступеня, і, відповідно, збільшується кількість параметрів.

Лінійні конгруентні генератори мають простий алгоритм і високу швидкість роботи. Вони непридатні для задач криптографії, але можуть використовуватися в різних додатках, які не потребують криптостійкості ПВП (моделювання, лотереї, комп'ютерні та азартні ігри, індустрія розваг).

### 1.3.3 Адитивні генератори псевдовипадкових послідовностей

Подальший розвиток ідеї ЛКГ полягає у зв'язуванні чергового елемента вихідної ПВП не з одним, а з двома попередніми елементами.

Найбільш відомим прикладом послідовності, в якій черговий елемент залежить від двох попередніх, є послідовність Фібоначчі. Адитивний генератор Фібоначчі використовує формулу

$$x_{i+1} = x_i \; + \; x_{i-1} \, mod \, n. \quad (1.6)$$

Період такого генератора, як правило, більше, ніж у ЛКГ з тим же значенням модуля n. Проте, якість отриманих таким чином ПДП виявляється недостатньою, для використання їх в тих випадках, коли потрібна висока ступінь випадковості.

Для подолання недоліків генератора Фібоначчі був запропонований його варіант з наступною формулою загального члена послідовності:

$$x_{i+1} = x_{i-j} \; + \; x_{i-k} \, mod \, n. \quad (1.7)$$

Вхідні індекси формули (1.7) значення k и j, для яких має виконуватися умова $k > j \geq 1$, називаються запізнюваннями, а відповідний генератор називається генератором Фібоначчі з запізненням. Для його максимально

можливого періоду існує оцінка [4], зазвичай формулюється у вигляді такої теореми:

Якщо многочлен $x^j + x^k + 1$ є примітивним многочленом над кінцевим полем (полем Галуа) другого порядку GF (2), то період відповідного генератором Фібоначчі з запізненням дорівнює числу

$$2^{\log_2 n-1}(2^k - 1). \quad (1.8)$$

Один з генераторів цього сімейства (Дж.Ж. Мітчел, Д.Ф. Мур) визначається формулою

$$x_i = x_{i-24} + x_{i-55} \, mod \, n, i \geq 55 , \quad (1.9)$$

де n-парне число, x0, ..., x54 -довільні цілі числа.

Відомий мультиплікативний варіант генератора Фібоначчі (Дж. марсали), для якого у формулі (1.7) замість складання використовується операція множення. Максимально можливий період такого генератора, який реалізується при вже запропонованих умовах на j і k, дорівнює

$$2^{\log_2 n-3}(2^k - 1). \quad (1.10)$$

Використовувані на практиці величини запізнень наводяться, наприклад, в [4]. Слід мати на увазі, що з приростом цих величин при програмній реалізації генератора збільшується потреба в пам'яті, що знижує ефективність алгоритма.

Крім цього, при тестуванні вихідної послідовності мультиплікативного генератора Фібоначчі (як і ЛКГ) проявляється її решітчата структура [4].

Таку ж особливість структури зберігають послідовності, згенеровані за допомогою більш складно організованого алгоритму, в якому черговий елемент залежить від довільного кількості попередніх елементів:

$$x_i = a_1 x_{i-1} + \ldots + a_{k-1} x_{i-k+1} + a_k \mod n \quad (1.11)$$

В деяких умовах коефіцієнти лінійної комбінації в правій частині формули (1.11) можуть бути визначені таким чином, щоб період послідовності був максимально можливим. Спосіб знаходження коефіцієнтів

реалізується засобами теорії кінцевих полів і являє собою складну математичну задачу.

Адитивні генератори мають високу продуктивність, але не є крипостійкими. Вони використовуються як допоміжні блоки в складі стійких ГПВП. Також вони служать основою деяких криптоалгоритмів (Fish, Pike, Mush) [3].

### 1.3.4 Генератор на регістрах зсуву з лінійним зворотним зв’язком

Генератори на регістрах зсуву з лінійним зворотним зв'язком (РЗЛЗЗ) грають дуже важливу роль як в генерації ПВП, так і в різних аспектах захисту даних, таких як контроль цілісності даних при їх ненавмисному спотворенні (CRC-коди), для самотестування інтегральних схем, при розробці криптоалгоритмів [2-4].

Алгоритм РЗЛЗЗ використовує двійкове подання числа. Лінійний регістр зсуву з зворотним зв'язком являє собою сукупність регістра зсуву і функції зворотного зв'язку. Регістр зсуву це упорядкований набір бітів довжини n, для якого визначена операція зсуву бітів на одну і ту ж величину вліво або вправо. При необхідності витягти біт вміст регістра зміщується на одну позицію вправо (рис.1.1).

Рисунок 1.1 – Схема роботи зсуву з зворотним зв’язком

При генерації чергового біта частина заздалегідь визначених осередків, званих «відводами» (tapped cells), пропускаються через функцію зворотного зв'язку.

У найбільш простому для практичної реалізації випадку функція зворотного зв'язку лінійна (відповідно лінійним є і регістр зсуву). Будемо вважати, що в якості такої функції використовується операція XOR (виключне АБО).

Нехай конфігурація відвідної послідовності задається послідовністю бітів $[c_1 \dots c_l]$, в якій відводам відповідають одиниці, а іншим осередкам - нулі. Вміст регістра зсуву позначимо $[s_{l-1} \dots s_0]$. Шаг генерації чергового біта за допомогою регістра зсуву довжини l складається з наступних операцій (рис. 1.2):

1. Вираховується значення функції зворотного зв'язку (наприклад, XOR осередків регістра з коефіцієнтами $c_1 \dots c_l$) .
2. Це ж значення записується в самий лівий осередок регістру $s_{l-1}$.
3. Склад всіх бітів регістра зсувається на одну позицію вправо.
4. Склад крайнього правого осередку регістру (s0) утворює вихідне значення генератора.

$$s_j = c_1 * s_{j-1} \oplus c_2 * s_{j-2} \oplus \dots \oplus c_l * s_{j-l}.$$

Рисунок 1.2 – Схема роботи регістра зсуву з лінійно зворотним зв’зком

Отримана послідовність обов'язково матиме період. Максимальне значення різних внутрішніх станів регістра, а, значить, період ПВП дорівнює $2^{l-1}$. Послідовність з таким періодом називається М-послідовністю (послідовністю максимального періоду).

Математичною основою РЗЛЗЗ є теорія лінійних послідовних машин і теорія полів Галуа. У відповідність описаним вище РЗЛЗЗ ставиться деякий многочлен, званий асоційованим, або створеним, многочленом. Його ступінь дорівнює довжині (розрядності) регістра, а коефіцієнти рівні елементам відвідної послідовності регістру $[c_1 \dots c_l]$. Асоційований многочлен визначає деякі важливі властивості вихідної ПВП [2, 4].

Основні плюси РЗЛЗЗ генераторів [4]:

1. Висока швидкість генерації.
2. Проста реалізація як в програмному, так і в апаратному варіантах.
3. Гарна якість вихідний ПВП з точки зору статистичних властивостей.

4. Зручність використання в якості допоміжних елементів, корисних при вирішенні деяких спеціальних завдань захисту даних, наприклад, отримання ПВП з певними характеристиками (довжина, період, закон розподілу).

Криптостійкість генераторів ПВП на базі РЗЛЗЗ через передбачуваність недостатня для їх використання при захисті даних. Однак нелінійні комбінації таких генераторів є складовими частинами систем поточного шифрування [2]. Вони також знаходять застосування в задачах контролю та діагностики засобів обчислювальної техніки (синтез генераторів псевдовипадкових тестових послідовностей і сигнатурних аналізаторів), в системах телекомунікацій (апаратна реалізація схем завадостійкого кодування, схем скремблювання) та інших.

Існує спеціальний вид ГПВП на основі регістрів зсуву - з нелінійної зворотним зв'язком [4].

### 1.4 Криптостійкі генератори псевдовипадкових чисел

Використовувані в криптографії та криптоаналітиці генератори псевдовипадкових послідовностей повинні відповідати більш жорстким вимогам, ніж, наприклад, генератори для задач моделювання або методів обчислень. Такі генератори, звані далі приптографічно стійкими (КСГПВП), повинні відповідати більш суворим критеріям:

1. Непередбачуваність результатів роботи: при невідомому ключі / початковому стані генератора на основі відомої кінцевої частини ПВП неможливо визначити як її наступний елемент, так і попередній.

2. Нерозрізненість статистичних властивостей згенерованої ПВП від аналогічних властивостей істинно випадкової послідовності.

3. Великий період послідовності.

4. Ефективна програмна реалізація.

Непередбачуваність генератора передбачає обчислювальну нерозв'язність наступних завдань:

- визначення попереднього елемента послідовності на основі відомого фрагмента ПВП кінцевої довжини (непередбачуваність вліво);
- визначення подальшого елемента послідовності на основі відомого фрагмента ПВП кінцевої довжини (непередбачуваність вправо);
- визначення ключа за відомим фрагментом ПВП кінцевої довжини.

Прийнято рахувати, що непередбачуваний вліво генер є крипостійким [8].

У роботі [7] пропонується наступний підхід до оцінки придатності ГПВП для задач криптології. Якісний (інакше - статистично безпечний) ГПВП також повинен мати:

- статистичні властивості, близькі до властивостей ІВП;
- нелінейний алгоритм перетворення ключа, що забезпечує розмноження внесених у вхідний текст спотворень;
- статистично незалежні результуючі ПВП при різних випадкових початкових значеннях генератора.

Для багатьох ГПВЧ (конгруентних генераторів, генераторів на основі регістрів зсуву та інших) зазначені критерії не виконуються: при привабливих статистичних властивостях результуючої ПВП алгоритми можуть бути уразливі для криптоаналізу, і в разі компрометації початкового стану генератора стає можливим як отримувати нові результати його роботи, так і відновлювати колишні.

### 1.4.1 Генератори на основі стійких криптоалгоритмів

Відповідно до наведеної вище класифікації з точки зору використовуваного в них перетворення ГПВП можна розділити на некриптографічні і криптографічні. Перетворенням при цьому є функція

зашифровування. Властивість криптостійкості цієї функції успадковує генератор.

До криптографічних генераторів відносяться генератори на базі блокових і потокових криптоалгоритмів, а також хеш-функцій.

Блокові генератори. Сильною стороною блокових ГПВП є їх непередбачуваність, яка забезпечується нелінійним перетворенням вхідних даних, реалізованим у вигляді багатораундової структури. Алгоритми стійких блокових шифрів засновані на перетвореннях підстановки (substitution) і перестановки (permutation). Багаторазові повтори цих операцій призводять до розсіювання (diffusion) і перемішування (confusion) бітів відкритого текста. В результаті розсіювання вплив вхідних бітів і бітів ключа поширюється на велике число бітів шифротекста. Тим самим ховаються можливі статистичні залежності між бітами відкритого тексту. Перемішування за допомогою підстановок ускладнює залежність між ключем і шифротекстом, ускладнюючи вилучення інформації про використаний ключ. В ідеальному випадку, якби до кожного елементу відкритого тексту застосовувалася унікальна підстановка, був би побудований абсолютно криптостійкий шифр. На практиці розсіювання і перемішування застосовуються спільно, що призводить до появи лавинного ефекту і забезпечує високу криптостійкість шифру. В якості відповідного алгоритму шифрування можна використовувати криптостійкі блокові шифри, такі як 3DES, AES, ГОСТ 28147-89 (Магма) і ГОСТ 34.12-2015 ( коник), RC6 та інші.

Поточні генератори. Перевага поточних генераторів полягає у високій швидкості роботи при достатньому ступені непередбачуваності. Поточні алгоритми шифрування, як правило, використовують в якості ключів заздалегідь згенеровані ПВП. Їх криптостійкість, а також секретність ключової послідовності, і визначають надійність шифру. Прикладами таких шифрів можуть бути Salsa20, HC-256, Cha-Cha, RC4 і т.д.

Генератори на основі хеш-функцій. Такі генератори мають найбільш суворо обгрунтовану непередбачуваність, оскільки вона базується на обчислювальній складності розв'язання математичних задач (розкладання великих чисел на прості множники або дискретного логарифмування). Однак генератори цього типу помітно програють по продуктивності. Крім того, використання хеш-функцій передбачає попереднє формування вхідної послідовності, що не завжди зручно при реалізації генератора

### 1.4.2 Генератори засновані на обчислювально складних математичних завданнях

Існують особливі види КСПВП, наприклад, засновані на обчислювально складних математичних завданнях. Алгоритм Блюма-Мікалі заснований на завданні дискретного логарифма, алгоритм Блюм-Блюма-Шуба - на передбачуваній складності факторизації цілих чисел. Останній алгоритм має доведену високу криптосктійкість, але відрізняється дуже низькою швидкістю роботи і не для всіх апаратних платформ допускає ефективну реалізацію. Також складним вважаються робота зі степенями числа: возведення у степінь, або знаходження кореня з числа [10].

### 1.4.3 Спеціальні реалізації

Прикладами генераторів, що використовують спеціальні реалізації, можуть служити: алгоритм Ярроу, в якому робиться спроба визначити ентропію вхідних даних; псевдопристроїв /dev/random та /dev/urandom, реалізовані в більшості POSIX-сумісних ОС; функція CryptGenRandom у складі CryptoAPI компанії Microsoft.

Таким чином, можна стверджувати, що стійкі алгоритми блокового і потокового шифрування можна використовувати як генератори псевдовипадкових послідовностей.

Непередбачуваність багатьох криптографічних генераторів неможливо довести строго, так як вона базується на оцінках недостатності різних ресурсів противника для розкриття використовуваного генератором криптоалгоритма. Це призводить до необхідності емпіричного дослідження якості ГПВП.

Існує інший погляд на якісний (щодо криптографії) ГПВП: побудова криптографічно стійкого генератора можна звести до побудови статистично безпечного генератора, який повинен відповідати таким вимогам [4]:

- по результатам проходження статистичних тестів отримана ПВП не відрізняється від ІСП;
- для побудови генератора використовується таке нелінійне перетворення ключа, котре забезпечує розмноження спотворень;
- при різних випадкових початкових значеннях генератор породжує статистично незалежні ПВП.

### 1.4.4 Використання криптостійких генераторів псевдовипадкових послідовностей

Зупинимося докладніше на декількох, найбільш істотних для цілей шифрування, варіантах застосування ГПВП.

У відповідності до відомих принципів Керкгоффса [9] надійність шифрування визначається тільки секретністю ключа, але не секретністю криптоалгоритмів. При асиметричному шифруванні ключ повинен відповідати певним вимогам, пов'язаним з математичною основою алгоритму. Для симетричного шифрування секретні ключі є найважливішою складовою надійності відповідних алгоритмів. Ідеальним ключем є випадкова послідовність бітів. На практиці замість такого ключа часто використовується якісна ПВП, що складається з рівноймовірних бітів, які не мають статистичних закономірностей [11].

У асиметричних криптосистемах функція генератора ПДП полягає в тому, щоб, використовуючи короткий секретний ключ k як «зерно» (seed), сформувати довгу псевдовипадкову послідовність (наприклад, в криптосистемах RSA, ElGamal, ГОСТ 3410).

Довжина ключа, що забезпечує надійність криптоалгоритма, визначається системою шифрування і становить не менше 128 бітів для симетричних систем і не менше 2304 бітів для систем з відкритим ключем [11].

### 1.4.5 Формування ключа для симетричних криптосистем

Якісний ключ, призначений для використання в рамках симетричної криптосистеми, являє собою випадковий двійковий набір. Якщо потрібно ключ розрядністю n, в процесі його генерації з однаковою ймовірністю повинен виходити будь-який з 2n можливих кодів. Генерація ключів для асиметричних криптосистем - процедура складніша: так, ключі, застосовувані в таких системах, повинні володіти певними математичними властивостями. Наприклад, в разі системи RSA модуль шифрування є добуток двох великих простих чисел.

Для генерації ключової інформації, призначеної для використання в рамках симетричної криптосистеми, використовуються наступні методи (в порядку зростання якості):

1. Програмна генерація, яка передбачає обчислення чергового псевдовипадкового числа як функції поточного часу, послідовності символів, введених користувачем, особливостей його клавіатурного почерку і т.п.
2. Програмна генерація, заснована на моделюванні якісного генератора ПВП з рівномірним законом розподілення.
3. Апаратна генерація з використанням якісного генератора ПВП.

4. Апаратна генерація з використанням генераторів випадкових послідовностей, побудованих на основі фізичних генераторів шуму і якісних генераторів ПВП.

Використання якісного генератора ПВП дозволяє при реалізації симетричних блокових шифрів зменшити число раундів шифрування, а значить, збільшити швидкодію криптоалгоритма.

### 1.4.6 Генерація гамми для синхронних поточних шифрів

У синхронному потоковому шифрі послідовність, яка зашифровується, генерується незалежно від потоку відкритого тексту і потоку шифротекста. Функціонування генератора гами при потоковому шифруванні ілюструє схема на рис. 1.2[7]. Початковий стан може бути функцією від ключа k та, можливо, від деякої рандомізують змінної. Мета генератора гами - розгорнути короткий випадковий ключ k у довгу псевдовипадкову послідовність.

Рисунок 1.2 – Роль ГПВП у процесі шифрування: *а* – абсолютно стійкий шифр; *б* – гамування(синхронний потоковий шифр); G - ГПВП; F - лінійна(XOR або *mod p*) або нелінійна функція.

### 1.4.5 Генерація гамми для самосинхронізованих поточних шифрів

Так звані самосинхронізовані або асинхронні потокові шифри, навпаки, мають здатність продовжувати правильне розшифрування в тому випадку, коли шифропослідовність, що генерується приймальним шифратором (дешифратором), випадає з синхронізації з гамою шифратора передавального. Для таких потокових шифрів функція, яка визначає

наступний стан криптосистеми, бере в якості входу фрагмент шифротексту, згенерованого до цього.

В самосинхронізованих поточних шифрах (рис. 1.3) символи відкритого тексту шифруються з урахуванням обмеженого числа попередніх n-символів, які беруть участь у формуванні ключової послідовності [7]. При цьому секретним ключем $Z$ є функція зворотного зв'язку генератора ПВП.

Рисунок 1.3 – Гамування зі зворотним зв’язком(самосинхронізоване потокове шифрування)($FB$ – функція зворотнього зв’зку; $Q$ – елементи пам’яті ГПВП)

## 1.5 Тестування генераторів псевдовипадкових послідовностей

Генератори псевдовипадкових послідовностей можуть використовуватися для виконання різних функцій. Крім того, вони можуть використовуватися для вирішення важливих завдань, в тому числі для захисту даних. Тому необхідна надійна перевірка властивостей генераторів, таких, як близькість вихідної послідовності до істинно випадкової з точки зору її статистичних властивостей і непередбачуваність вихідних значень.

Існують два основних напрямки аналізу псевдовипадкових послідовностей:

- *Криптографічні.* Ціль цього напрямку - пошук таких закономірностей досліджуваної послідовності, щоб по її частині можна було відновити всю послідовність.
- *Статистичні.* Цей напрямок орієнтований на пошук відхилень статистичних властивостей псевдовипадкової послідовності, на основі яких можна прогнозувати наступні і попередні значення членів послідовності з ймовірністю, більшою за 0,5.

Надійний доказ непередбачуваності генераторів псевдовипадкових послідовностей - складна проблема, яка до сих пір не вирішена. У разі криптографічно стійких генераторів псевдовипадкових послідовностей зазвичай вважається, що для визначення наступного біта вироблюваної послідовності з імовірністю більше 0,5 у супротивника не буде достатньо ресурсів (таких, як часу, обчислювальних ресурсів, а також матеріальних засобів).

Існують теоретичні методи дослідження псевдовипадкових послідовностей, які використовують різні підходи до визначення випадковості. До них відносяться, наприклад, частотний підхід (вперше запропонований фон Мізесом), алгоритмічний підхід Мартін-Лефа, сложностной підхід Колмогорова [7]. Однак в даний час такі теоретичні розробки не дають застосованих на практиці інструментів оцінки ступеня випадковості послідовності.

Існує і інший підхід, який передбачає, що певний набір статистичних властивостей псевдовипадкової послідовності повинен відповідати аналогічним властивостям істинно випадкової послідовності. Простіше кажучи, псевдовипадкова послідовність при проходженні статистичних тестів повинна поводитися як істинно випадкова послідовність. При цьому жоден тест не повинен виявляти в досліджуваній псевдовипадковій послідовності будь-які закономірності, які б відрізняли її від істинно випадкової послідовності.

На практиці підтвердження ступеня випадковості результату роботи ГПВП означає перевірку відсутності в отриманій послідовності статистичних закономірностей і кореляції між послідовностями.

Історично перші необхідні (але не достатні) умови статистичної подібності періодичної двійковій ПВП і ІВП були сформульовані в 1967 р. Вони відомі як постулати Голомба:

1. Число одиниць і число нулів в кожному періоді ПВП повинні відрізнятися не більше ніж на одиницю.

2. В кожному періоді ПВП половина серій, тобто послідовностей, що складаються з однакових символів, повинна мати довжину один, одна чверть - мати довжину два і т.д .; до того ж для кожної з цих довжин повинна бути майже однакова кількість серій нулів і одиниць.

3. Автокореляційна функція, що є мірою подібності послідовності і її зсуву на будь-яке число бітів *t*, нерівний періоду, приймає різні значення в міру того, як *t* проходить всі допустимі значення.

Сенс останнього постулату можна трактувати наступним чином: знання попереднього значення послідовності не дозволяє зробити припущення про поточне її значення.

Але виконання цих постулатів не гарантує випадковий характер даної послідовності. Для аналізу рівня випадковості послідовності розроблені і розробляються спеціальні статистичні тести. В даний час такі тести стали основним інструментом оцінки якості генераторів псевдовипадкових послідовностей.

Такі тести дозволяють:

- оцінювати можливість використання ПВП в криптографічних додатках;
- виявляти генератори, що виробляють послідовності, що істотно відрізняються від ІВП;
- перевіряти правильність реалізації генераторів ПВП;
- розробляти нові генератори ПВП.

Статистичні тести використовуються також для оцінки якості криптографічних примітивів (шифрів, хеш-функцій) шляхом перевірки криптографічних властивостей вироблюваних ними послідовностей (лавинного ефекту зміни вихідних даних при викривлені елементів вхідних даних, кореляції проміжних і вихідних послідовностей). Вперше це було зроблено в ході відомого конкурсу з вибору нового криптографічного стандарту США Advanced Encryption Standard (AES).

Існують різні методики тестування генераторів псевдовипадкових послідовностей. Так чи інакше, всі вони базуються на порівнянні властивостей вироблюваних ними послідовностей з властивостями істинно еталонної випадкової послідовності. За характером оцінювання та подання тести, використовувані для дослідження властивостей послідовностей, можна розділити на два основних типи: статистичні тести і графічні тести. Найчастіше статистичні тести використовують для тестування двійкових послідовностей. Якщо ж послідовність отримувана генератором не двійкова – використовують графічні тести.

### 1.5.1 Статистичні тести

Статистичні тести це емпіричні тести для оцінки якості ГПВП і виявлення їх «слабких місць» шляхом розрахунку статистичних характеристик ПВП і порівняння їх з аналогічними характеристиками ІВП.

Вирішення таких завдань засноване на перевірці деяких гіпотез щодо властивостей ПВП, вироблених генераторами. В якості статистичної гіпотези може використовуватися довільне припущення про характер розподілу і властивості випадкової величини. Істинність або хибність такого припущення підтверджується або відхиляється за допомогою методів математичної статистики.

Загальний механізм перевірки статистичних гіпотез полягає в наступному. Висуваються дві гіпотези: нульова (H0) та альтернативна (H1). Припустимо, нульова гіпотеза полягає в тому, що тестована ПВП істинно випадкова (з точки зору конкретного тесту), а альтернативна - що ПВП не випадкова [7].

Для кожного тесту вибирається деяка обчислювана функція (статистика), яка зводить властивість випадковості протестованих даних до одного числового значення (що спостерігається зі статистикою). Статистика тесту представляється як реалізація випадкової величини. Для того щоб

виконати оцінку проходження тесту, необхідно знати розподіл тестової статистики в припущенні, що нульова гіпотеза вірна. Часто цю роль грають нормальний розподіл і розподіл $x^2$ (Пірсона) [8].

Статистичні тести поєднують в собі обчислювальні процедури для знаходження статистики досліджуваної послідовності і вирішальне правило перевірки, за допомогою якого за значеннями статистики визначають, прийняти або відкинути нульову гіпотезу:

- якщо вибіркове значення статистики належить критичній області, то нульова гіпотеза $H_0$ відкидається, так як при одноразовому випробуванні відбулася подія, ймовірність якого мала і дорівнює α;
- якщо вибіркове значення статистики потрапляє в допустиму область, то робиться висновок, що дані випробування не суперечать висунутій нульовій гіпотезі $H_0$ і вона приймається.

Таким чином, алгоритм перевірки статистичних гіпотез складається з наступних кроків (рис. 1.4):

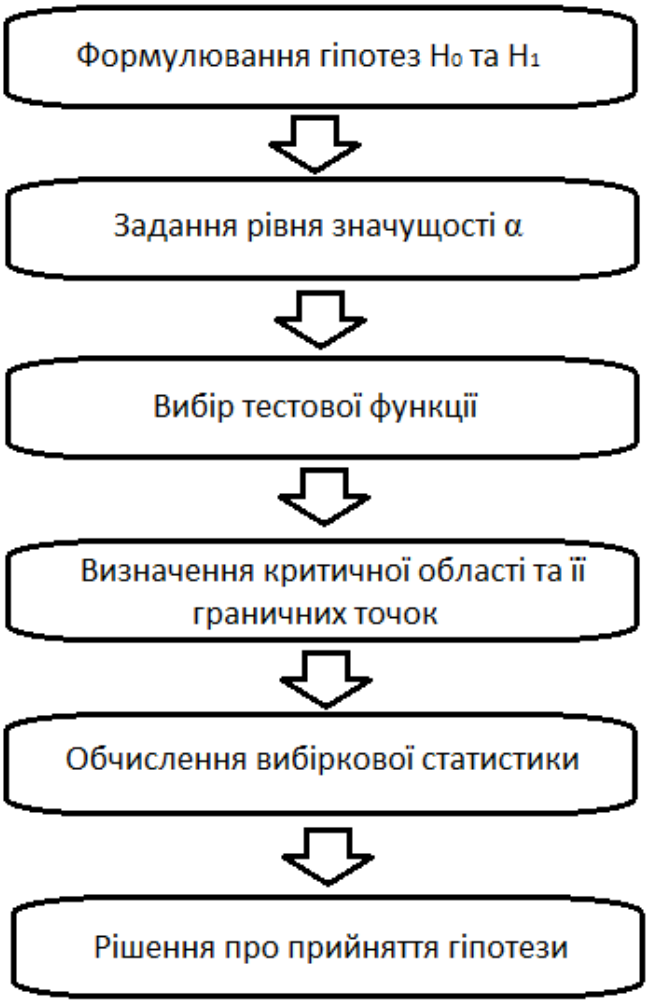



Рисунок 1.4 – Алгоритм перевірки статистичних гіпотез

Для того щоб зробити висновок про проходження тесту, перевірка цих гіпотез виконується за допомогою різних статистичних критеріїв - правил, відповідно до яких приймається або відхиляється нульова гіпотеза. Нижче перераховані різні типи таких критеріїв в порядку наростання їх надійності:

1. Порогове значення. Величина обчисленої статистики порівнюється з деяким граничним значенням, і якщо статистика, наприклад, перевищує його, тест вважається пройденним.

2. Довірчий інтервал. Тест вважається пройденим, якщо величина статистики тесту потрапляє в певний довірчий інтервал, що залежить від прийнятого рівня значущості.

3. Імовірнісний підхід. Набір значень статистики тесту вважається набором значень випадкової величини з заданим законом розподілення.

Останній варіант зарекомендував себе найбільш ефективним і надійним. Саме він використовується в багатьох пакетах статистичних тестів.

При ухваленні рішення про те, чи був пройдений тест, можливі два типи помилок. Виникнення помилки першого роду означає, що тестована псевдовипадкова послідовність насправді є випадковою, але вірна нульова гіпотеза $H_0$ відкидається. Імовірність такої помилки дорівнює рівню значущості α, який задається до початку тестування. Рівень значущості α - це ймовірність того, що тестування покаже невипадковість послідовності, тоді як фактично вона є випадковою. Відповідно, ймовірність прийняття правильного рішення становить (1-α). Зазвичай в практичних завданнях прийнятним вважається значення рівня значущості 0,05. Однак для цілей криптографії використовують більш суворі значення α (як правило, з інтервалу [0,001; 0,01]).

Помилка другого роду (β) означає прийняття гіпотези про випадковість розглянутої послідовності, коли послідовність в дійсності не випадкова. З точки зору криптографії така помилка більш критична. Величина помилки другого роду визначає потужність критерію - ймовірність того, що нульова гіпотеза буде відхилена при вірній альтернативній гіпотезі. Між двома цими видами помилок існує взаємозалежність: чим менше α, тим більше β, і навпаки.

Статистика тесту побудована так, що її менші значення відповідають дефектам псевдовипадкової послідовності - відхилення від істинної випадковості.

Зазвичай для зручності сприйняття результатів тестування обчислення за допомогою еталонного розподілу ймовірностей, тестова статистика перетворюється в так зване значення p-value. Це значення трактується як ймовірність того, при заданому рівні значущості ідеальний генератор випадкових послідовностей може призвести послідовність, менш випадкову, ніж досліджувана. Така подія тим менш імовірна, чим менше значення p-value. Виконання умови p-value >= α означає успішне проходження теста.

Важливим є той факт, що для будь-яких статистичних тестів, які відповідають нульовій гіпотезі, значення p-value рівномірно розподілені на інтервалі [0; 1). Це означає, що тестована довільним тестом послідовність повинна бути рівномірно розподілена на інтервалі [0; 1).

Перерахуємо кілька найбільш відомих інструментів статистичного тестування псевдовипадкових послідовностей [7]:

- добірка Кендалла і Бабінгтон-Сміта;
- тести Д. Кнута;
- тести DIEHARD (Дж. Марсали);
- пакет тестів NIST (A. Rukhin і ін.);
- пакет TestU01 (П. Л'Екуйе);
- Crypt-XS (Helen Gustafson);
- John Walker (Autodesk, Inc.), ENT;
- Dieharder (Robert G. Brown).

Крім пакетів, що містять набори тестів для всебічного вивчення статистичних властивостей ГПВП, існують окремі тести, які спрямовані на більш точний і повний аналіз ПВП.

Слід розуміти, що тестування не може замінити криптоаналіз. Проте, воно є обов'язковим етапом аналізу стійкості криптографічного генератора. В умовах існування великої кількості різних статистичних тестів, як широко і давно поширених, так і нових, важливий обгрунтований вибір, пов'язаний зі специфікою вирішуваних завдань захисту інформації.

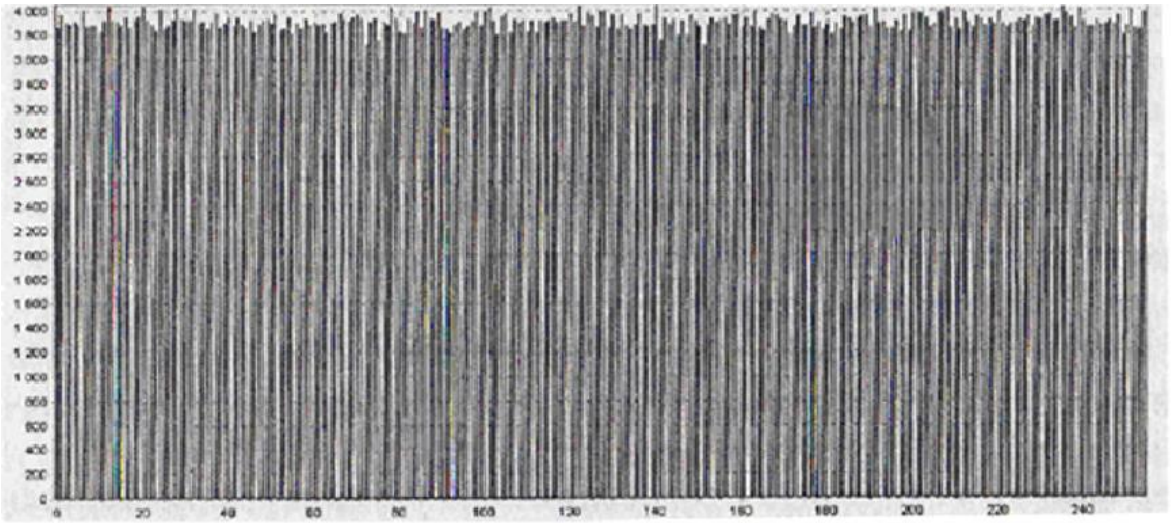
### 1.5.2 Графічні тести

Графічні тести використовують гістограму або діаграму розподілу на площині елементів послідовності (рис.1.5), перевірку серій, перевірку на монотонність, автокореляційну функцію, профіль лінійної складності і дискретне перетворення Фур'є [4]. Графічні тести мають ту саму математичну основу, що і статистичні, але поступаються статистичним

тестам у точності, оскільки орієнтовані на пошук явних відхилень псевдовипадкової послідовності від еталону. Такі відхилення повинні бути добре помітні «на око» при візуальному сприйнятті. Графічні тести дають наближене візуальне уявлення певних статистичних властивостей досліджуваної послідовності у вигляді тих чи інших графіків і гістограм. При використанні графічних тестів працюють не з бітами, а з числами. З цієї причини послідовність, яку тестують, часто представляють у вигляді набору, наприклад, 32-бітних чисел.

Рисунок 1.5 – візуалізація залежностей між елементами послідовності

Тест «Гістограма розподілу елементів» дозволяє оцінити рівномірність розподілу символів в досліджуваній послідовності і визначити частоту появи конкретного символу. Для побудови гістограми в згенерованій послідовності підраховується, скільки разів зустрічається кожен елемент, після чого будується графік залежності числа появ елементів від їх чисельного представ-лення (найчастіше використовується таблиця ASCII, або ж її еквівалент – проміжок чисел [1, 255]). Вважається, що послідовність задовільняє вимогам випадковості, якщо в ній присутні всі можливі елементи даної розрядності, при цьому розкид частот появи символів прагне до нуля. В іншому випадку, вона не є випадковою [1].



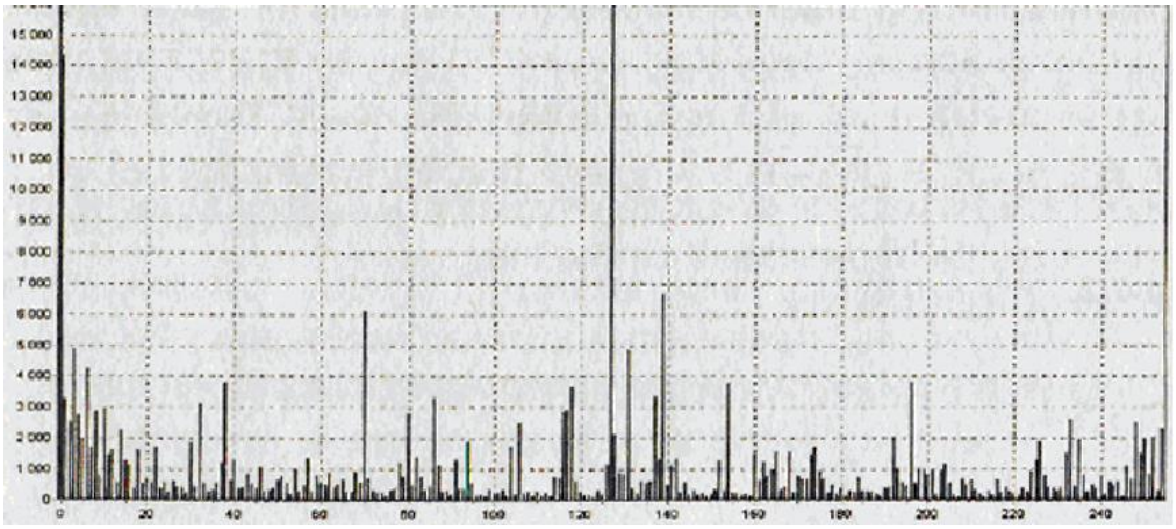


Рисунок 1.6 а – позитивний результат проходження тесту, б – негативний результат проходження тесту

Даний тест може принести користь також в тих випадках, коли оцінюється якість послідовності з законом розподілу, відмінним від рівномірного, або послідовності, в якій деякі символи взагалі відсутні(рис 1.7)

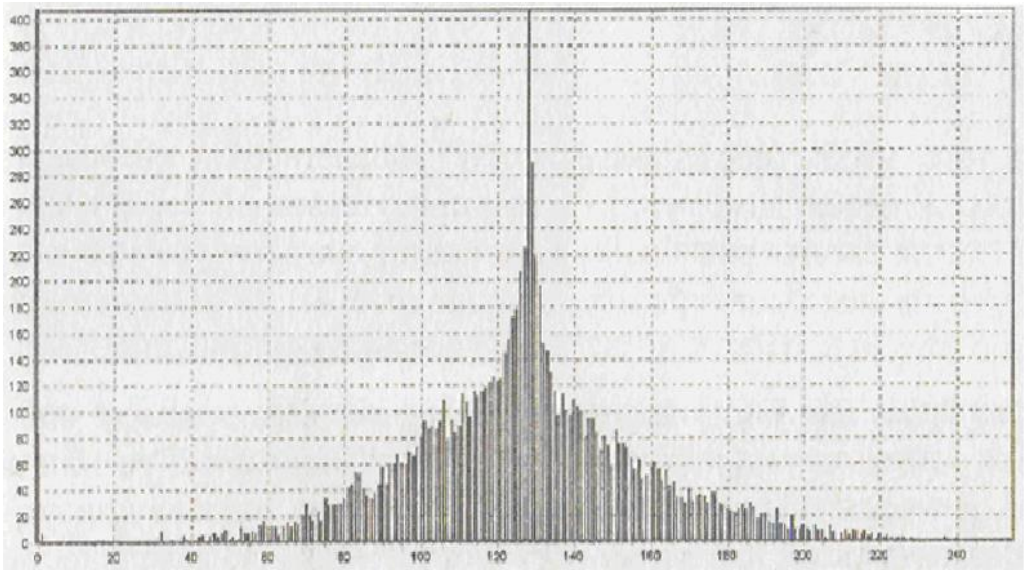


Рисунок 1.7 – Приклад гістограми послідовності з законом розподілення близьким до нормального

Тест «Розподіл на площині» використовується для визначення залежностей між елементами досліджуваної послідовності. Для цього на поле

розміром R – кількість елементів досліджуваної послідовності, наносяться точки з координатами (R; $x_r$), де $x_r$ - елементи досліджуваної послідовності x. Далі аналізується вид отриманої картинки - якщо точки на полі розташовані хаотично, то вважається, що між елементами послідовності відсутні залежності. Якщо ж на полі присутні залежності, тобто отримані якісь візерунки, то послідовність не є випадковою(рис. 1.9).

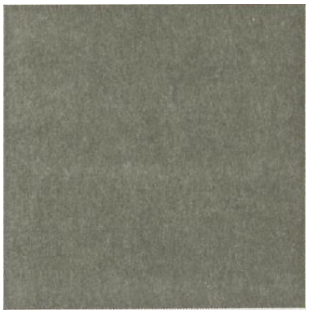


Рисунок 1.8 – позитивний результат послідовності великої довжини

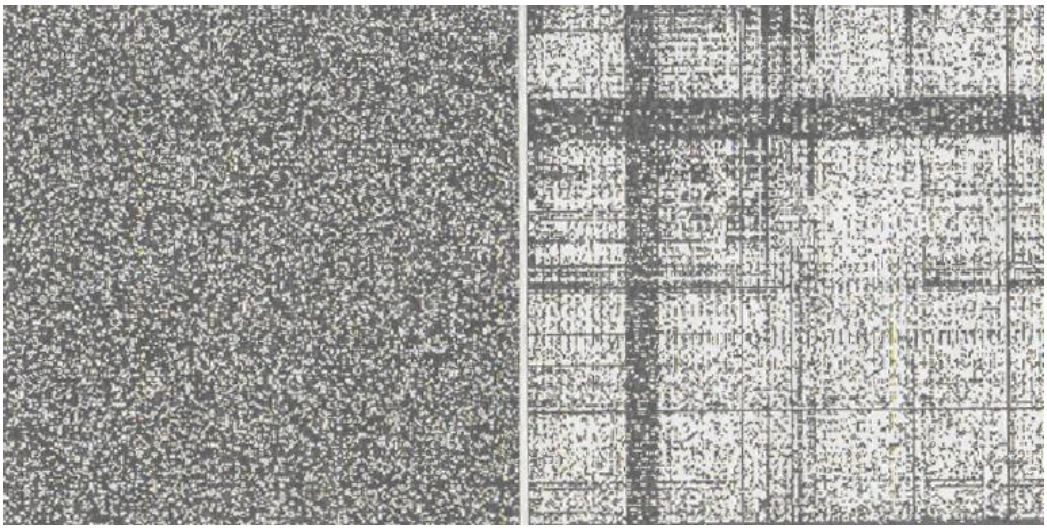


Рисунок 1.9 а – позитивний результат проходження тесту, б – негативний результат проходження тесту

Для послідовності великої довжини гарним результатом є абсолютно чорне поле(рис 1.8).

Тест «Перевірка на монотонність» дозволяє оцінити рівномірність розподілу символів в досліджуваній послідовності на основі аналізу довжин ділянок зростання і спаду елементів послідовності.

Побудова проводиться таким чином. Досліджувана послідовність ε графічно представляється у вигляді наступних один за одним непересічних ділянок зростання та спаду елементів послідовності.

У послідовності, чиї статистичні властивості близькі до властивостей істинно випадкової послідовності, ймовірність появи ділянки зростання (спаду) певного розміру залежить від його довжини: чим більше довжина, тим менше ймовірність (рис. 1.11, а). В іншому випадку послідовність не є випадковою (рис. 1.11, б).

Наприклад, маємо наступну послідовність:

ε= 1 2 3 4 5 5 4 4 3 2 2 2 1 2 3 4 5.

Розбиваємо послідовність на ділянки зросту та спаду:

ε= 1234555 4432221 2345.

Отримуємо ділянку зростання довжиною 6, далі ділянка спаду довжиною 7 і знову ділянко зростання довжиною 4(рис. 1.10)

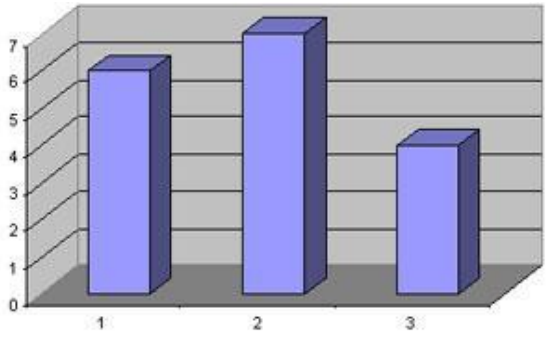



Рисунок 1.10 – приклад тесту перевірки на монотонність

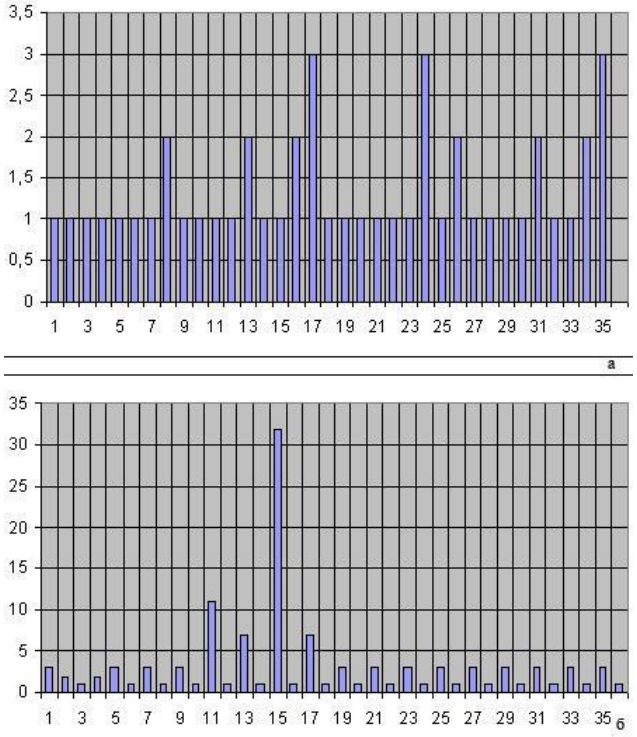



Рисунок 1.11 а – позитивний результат проходження тесту, б – негативний результат проходження тесту

### 1.6 Висновки за розділом 1

Проаналізувавши попередньо описані методи генерування ПВП, дійшов до висновку, що основні параметри які повинен мати генератор:

- ефективна апаратна реалізація;
- швидкодія;
- неможливість відтворення послідовності без знання ключа($\boldsymbol{x_0}$);
- встановлення чіткого періоду між якими власне і відбувається генерування послідовності.

Швидкодії буде набувати наш ГПВЧ, завдяки ітеративному підходу конгруентного методу. Ефективної апаратної реалізації додаток набуде завдяки розробці на об’єктно-орієнтованому язику програмування. Реалізуємо також зсув кожного значення на число, яке також згенерується конгруентною послідовністю, що не дасть можливості відтворити послідовність без знання ключа.

Виявивши властивості, які повинен мати ефективний ГПВЧ, перейдемо до розробки його схеми, та реалізації у нашій програмі.

# РОЗДІЛ 2
# РОЗРОБКА ГЕНЕРАТОРА

## 2.1 Розробка схеми ГПВЧ

Розглянувши властивості ефективного генератору, спроектуємо власний ГПВЧ. У схемі має бути присутній власне початковий генератор конгруентної послідовності, який буде генерувати послідовність від якої ми будемо опиратися. Також, маємо генерувати зсув для кожного елементу, так як зсув не може бути постійним числом, бо даний підхід фактично лиш змінить одну з констант – c, а також при умові що $m - x_n > c$, отримаємо вихід за межі періоду. А оскільки наш ГПВЧ має комбінувати конгруентні генератори, то для кожного елементу будемо формувати зсув із сум двох

інших конгруентних генераторів, період яких може бути, навіть, більшим за період початкового генератору, при умові

*Визначення 2.1* $x_n < 0$, тоді $x_n = x_n + m$, або $x_n > m$, тоді $x_n = x_n - m$ ,

де m – період першого генератора, $x_n$-значення елементу на даній ітерації.

Також новий генератор має містити у собі обчислювально складну функцію, для даної реалізації, я обрав такі функції як возведення у степінь, та взяття кореня із числа.

У загальному весь алгоритм виглядає наступним чином: формуються 3 конгруентні послідовності за заданими параметрами, доцільність вибору яких описана у наступних пунктах, для кожного елементу першої послідовності формується зсув, який описаний вище, потім перевіряється чи досі елемент знаходиться у рамках [0, m], згідно визначення 2.1, після цього елемент зводиться у третю степінь, та з нього береться корінь квадратний, заключним етапом є повторна перевірка входження елементу до діапазону [0, m].

Схематично ГПВЧ виглядає наступним чином(рис. 2.1), виконуючи умови описані вище.

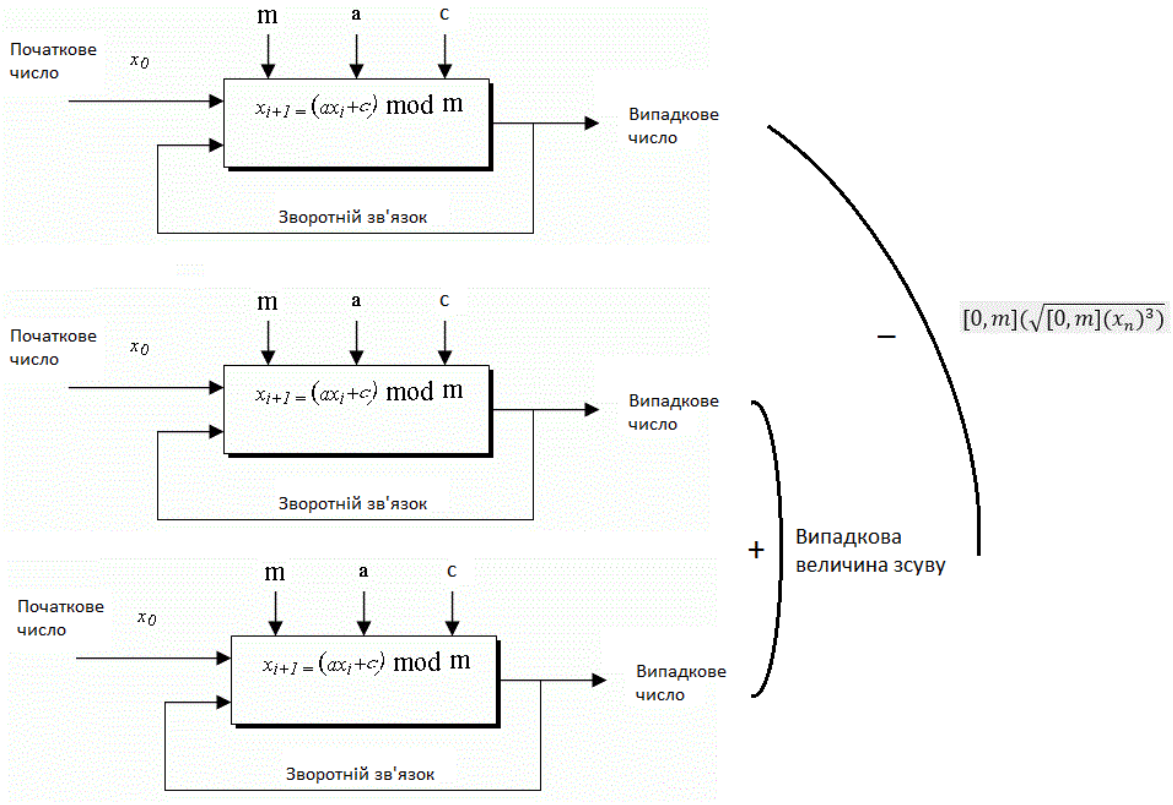



Рисунок 2.1 – Схема ГПВЧ складеного з трьох конгруентних генераторів

## 2.2 Оцінка параметрів конгруентного генератора

Згадаємо формулу конгруентної послідовності, описану вище(1.5). В даній формулі важливий правильний підбір параметрів. Наприклад, для $\boldsymbol{x_0}$ = 7, a = 8, c = 9, m = 10 отримаємо послідовність

7, 5, 9, 1, 7, 5, 9, 1

Тобто послідовність взагалі не виглядає «випадковою». На даному прикладі було проілюстровано, що конгруентна послідовність завжди зациклюється. Більше того, далі буде показано, що ця властивість існує у всіх послідовностей виду $x_{n+1} = f(x_n)$. На рисунках 2.1-2.2 наведені графіки ЛКП довжиною 1000 чисел. Як бачимо з рисунка, навіть невелика зміна параметра функції приводить до появи короткого періоду.(Всі графіки були створені в профільному додатку – Excel)

Рисунок 2.1 – Графік ЛКП для $\boldsymbol{x_0 = 7}$, a = 106, c = 1283, m = 6075

Крім того, у ЛКП є ще один явний недолік – наявна «решітчаста» структура послідовності чисел зображена на рисунку 2.3. Даний недолік обумовлений властивостями операцій у кінцевому полі, використаних у формулі 1.5.

Роздивимось задачу правильного вибору числа m. По-перше, m має бути досить великим. Наприклад, для m = 2, послідовність в кращому випадку буде мати вигляд 0, 1, 0, 1 ….

По-друге, значення m, має сенс вибирати таким, що дорівнює $2^q$, де q – число бітів у машинному слові, оскільки це дозволяє не застосовувати ділення по модулю.

Рисунок 2.2 – Графік ЛКП для $\boldsymbol{x_0 = 7}$, a = 105, c = 1283, m = 6075

Рисунок 2.3 – Графік ЛКП для $\boldsymbol{x_0 = 7}$, a = 106, c = 1284, m = 6075

## 2.3 Підбір значень до кожного конгруентного генератора

Нижче наведена таблиця констант для лінійних конгруентних генераторів, взята з [5].

*Таблиця 2.1*

Параметри ЛКГ для формули 1.5

| $(a, c, m)$ | $(a, c, m)$ | $(a, c, m)$ |
|---|---|---|
| (106,1283,6075) | (625,6571,31104) | (1277,24749,117128) |
| (211,1663,7875) | (1541,2957,14000) | (2041,25673,121500) |
| (421,1663,7875) | (1741,2731,12960) | (2311,25367,120050) |
| (430,2531,11979) | (1291,4621,21870) | (1597,51749,244944) |
| (936,1399,6655) | (205,29573,139968) | (2661,36979,175000) |
| (1366,1283,6075) | (421, 17117,81000) | (4081,25673,121500) |
| (171,11213,53125) | (1255,6173,29282) | (3661,30809,145800) |
| (859,2531,11979) | (281,28411,134456) | (3613,45289,214326) |
| (419,6173,29282) | (1093,18257,86436) | (1366,150889,714025) |
| (967,3041,14406) | (421,54773,259200) | (8121,28411,134456) |
| (141,28411,134456) | (1021,24631,116640) | (4561,51349,243000) |

Вибирати значення періоду кожного генератору будемо незалежно, оскільки, вони не взаємозв'язані, за означенням 2.1, у той же час значення періоду не має перевищувати $2^{64}$ для забезпечення швидкодії. Разом з цим використанні значення не будемо записувати константами у коді, а організуємо роботу з файлами. Це робиться з двох причин:

1) При заволодінні зловмисником коду, він не матиме змогу відтворити послідовність;

2) Для зручності роботи з програмою.

Оскільки, ключовий простір має бути досить великим, для унеможливлення перебору всіх значень, а з визначення 2.1 випливає $0 \leq x_0 < m$, де m період першого генератору, то значення з наведеної вище таблиці для першого генератору маємо взяти з найбільшим значенням m(a=421, c=54773, m=259200).

Значення для двох інших генераторів маємо вибрати з наступної логіки, що випливає з теореми 2.2, сума максимально можливих значень другого та третього генераторів має дорівнювати періоду першого генератора. Наприклад, для другого генератору виберемо значення(a=205, c=29573, m=139968), а третього(a=2311, c=25367, m=120050).

## 2.4 Доцільність вибору мови програмування

У дипломному проекті була поставлена задача, реалізація якої може бути проведена за допомогою різних мов програмування. Але я вибрав мову C #, що входить в Visual Studio .Net (v.7), JetBrains Rider. так як він має ряд переваг, які спрощують процес створення додатків. Важливу роль у виборі мови зіграло те, що C # - краща .NET-мова, так як він був спеціально спроектований для .NET Framework.

C # створювався Microsoft, як основна мова для .NET Framework. Microsoft спроектувала C # таким чином, що б C, C ++ і Java-програмісти змогли легко перейти на нього. C # має коріння в мовах C, C ++ і Java, тому такий перехід не повинен викликати труднощів.

Синтаксис у C # не такий складний як у C ++, тому його вивчення проходить набагато легше. Більшість операцій, які ви можете робити на C ++, можна зробити і на C #, за винятком операцій доступу до низькорівневих функцій (це все-таки можна зробити за допомогою некерованого коду).

C # - перша мова, підтримувана версіями .NET Framework для інших платформ.

Мова, яка використовує бібліотеку класів .NET Framework (FCL) і управляється загальномовним середовищем виконання (CLR) є .NET-сумісною мовою (мова, яка підтримує платформа .NET). Серед таких мов: Microsoft Visual Basic .NET (VB .NET), Microsoft Visual C ++ .NET, а також COBOL, Eiffel, Jscript, RPG і інші. Як я згадував раніше, C # був

спроектований спеціально для .NET Framework і містить деяку функціональність, яку ви не знайдете в інших мовах.

Програміст, який звик до платформи Майкрософт, може уявити собі C # як проміжний варіант між C ++ і Visual Basic, якщо розглядати складність мовних конструкцій і можливостей мови.

C # має C стиль синтаксису (для керуючих конструкцій, блоків коду, опису сигнатури методів і ін.), Багато спільного з Java (відсутність множинного спадкоємства і шаблонів, наявність збирача сміття) і Дельфі (орієнтованість на створення компонент), в той же час має і свій колорит.

При створенні мови в основу дизайну лягла легкість використання, домінуюча над потужністю мови і швидкістю виконання. [6] Звідси і збирач сміття з керованими об'єктними посиланнями, який автоматично звільняє за вас пам'ять, відбираючи при цьому процесорний час. Ви також отримуєте безпеку роботи з типами, а це, на думку багатьох, є другим найважливішим фактором уникнення помилок.

C # об'єктно-орієнтована мова, як і вся платформа .NET [6].

При створенні мови розглядалася не тільки простота створення програмного забезпечення, але і їх підтримки - в зв'язку, з чим в мову включили підтримку XML коментарів і контролю версій. Справжній подарунок для програмістів.

У C # представлена концепція просторів імен, аналогічна пакетам в Java. Це дозволяє ієрархічно структурувати вашу систему типів, роблячи код набагато більш зрозумілим і дозволяючи уникнути проблем з ім'ям. Ви можете розглядати простору імен як директорії, а мовні типи як файли в цих директоріях.

Реалізація структур як типів, робота з якими йде за значенням, разом з можливістю використовувати не тільки вкладені масиви (як в Java), але і багатовимірні дозволяє оптимізувати продуктивність додатків.

З огляду на дуже зручний об'єктно-орієнтований дизайн, C # є гарним вибором для швидкого конструювання різних компонентів - від

високорівневої бізнес логіки до системних додатків, що використовують низькорівневий код [6]. Також слід зазначити, що C # є і Web орієнтованим - використовуючи прості вбудовані конструкції мови ваші компоненти можуть бути легко перетворені в Web сервіси, до яких можна буде звертатися з Internet за допомогою будь-якої мови на будь-якій операційній системі. Додаткові можливості та переваги перед іншими мовами приносить в C # використання передових Web технологій, таких як: XML (Extensible Markup Language) і SOAP (Simple Object Access Protocol). Середовище розробки Web сервісів дозволяє програмісту дивитися на існуючі сьогодні Web додатки, як на рідні C # об'єкти, що дає можливість розробникам співвіднести наявні Web сервіси з їх знаннями в об'єктно-орієнтованому програмуванні [6].

Як було показано вище, вибрана мова програмування найбільш зручна для виконання поставленого завдання.

### 2.5 Програмна реалізація ГПВЧ

Полем називається член-змінна, що містить деяке значення. В ООП поля іноді називають даними об'єкта.

У власній реалізації я використовую одне поле, значення котрому я буду надавати через консоль

```
string X0 = "";
Console.WriteLine("Enter X0: ");
X0 = Console.ReadLine();
```

Рисунок 2.4 – присвоєння змінній значення через консоль

У власній програмі роботу побудував навколо методів, основні з них:

Метод, який формує конгруентну послідовність, приймаючи необхідні значення(рис. 2.5)

```
static List<int> LinearGongruent(int X0, int a,int c,int m)
{
    List<int> resultSeqence = new List<int>();
    for (int i = 0; i < lengthSequence; i++)
    {
        X0 = (a * X0) + c;
        while (X0 > m)
        {
            X0 = X0 - m;
        }
        resultSeqence.Add(X0);
    }

    //Повертає отриману послідовність з методу
    return resultSeqence;
}
```

Рисунок 2.5 – метод, що формує конгруентну послідовність

Метод, який буде віднімати від першої послідовності дві інших(рис. 2.6)

```
static List<long> MinusList(List<int> firstList, List<int> secondList, List<int> thirdList, int module)
{
    List<long> result = new List<long>();
    for (int i = 0; i < firstList.Count; i++)
    {
        long state;
        int temp = firstList[i] - secondList[i] - thirdList[i];
        while (temp < 0)
        {
            temp += module;
        }

        state = Convert.ToInt64(Math.Pow(temp, 3));
        state = Convert.ToInt64(Math.Sqrt(state));
        result.Add(state);
    }

    return result;
}
```

Рисунок 2.6 – метод, що формує зсув, та проводить роботу зі степенями

Метод, який перевірятиме елементи на входження в період зображений на рисунку 2.7.

Оскільки, раніше ми встановили доцільність роботи через файл, треба описати метод, який буде це реалізовувати(рис 2.8).

```
static List<int> CheckForMinus(List<long> minusList, int m)
{
    List<int> result = new List<int>();
    for (int i = 0; i < minusList.Count; i++)
    {
        while (minusList[i] < 0)
        {
            minusList[i] += m;
        }

        while (minusList[i] > m)
        {
            minusList[i] -= m;
        }
        result.Add(Convert.ToInt32(minusList[i]));
    }

    //Повернення результуючого словника
    return result;
}
```

Рисунок 2.7 – метод, що перевіряє входження елементу до діапазону

```
static int[,] ReadValues()
{
    StreamReader sr = new StreamReader(@"C:\study\diplom\data.txt");
    string text = sr.ReadToEnd();
    string[] values = text.Split(new char[] {'-'});
    List<string> valuesString = values.ToList();

    int[,] result = new int[3,3];
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            result[i, j] = Convert.ToInt32(valuesString[0]);
            valuesString.RemoveAt(0);
        }
    }

    //Повертаємо результуючий масив
    return result;
}
```

Рисунок 2.8 – метод, що організовує роботу з файлами.

За описаним алгоритмом, данні у файлі для зчитування має виглядати саме так:

data.txt – Блокнот

Файл Правка Формат Вид Справка

```
421-54773-259200-
205-29573-139968-
2311-25367-120050
```

Де перша, друга та третя строки будуть відповідно значеннями для першої, другої та третьої послідовності. Значення розташовані у наступному порядку a, c, m.

Метод, який комбінує у собі попередньо описані(рис 2.9).

```
static List<int> Generator(int X0)
{
    List<long> temp = new List<long>();
    List<int> result = new List<int>();
    int[,] values = ReadValues();

    List<int> firstList = LinearGongruent(X0,values[0,0],values[0,1],values[0,2]);
    List<int> secondList = LinearGongruent(X0, values[1, 0], values[1, 1], values[1, 2]);
    List<int> thirdList = LinearGongruent(X0, values[2, 0], values[2,1],values[2,2]);

    temp = MinusList(firstList, secondList, thirdList, values[0, 2]);
    result = CheckForMinus(temp, values[0, 2]);

    return result;
}
```

Рисунок 2.9 – метод, який і є генератором

Метод Main – точка входу компілятора, те місце, звідки починається робота коду(рис. 2.10).

```
public static void Main(string[] args)
{
    //Визначення місця в пам'яті у якому зберігається змінна
    string X0 = "";
    Console.WriteLine("Enter X0: ");
    X0 = Console.ReadLine();
    //Визов методу ГПВЧ та вивід отриманного на єкран
    foreach (var VARIABLE in Generator(Convert.ToInt32(X0)))
    {
        Console.Write($"{VARIABLE}, ");
    }
    Console.ReadKey();
}
```

Рисунок 2.10 – точка входу програми

## 2.6 Висновки за розділом 2

Згідно виявлених означень, які має містити у собі криптографічний генератор, у першому розділі створили власний ГПВЧ у додатку на мові програмування C#. На протязі всього розділу та моменту праці над додатком, намагався досягнути всіх властивостей, яким має відповідати ГПВЧ, а саме:

- ефективна апаратна реалізація;
- швидкодія
- неможливість відтворення послідовності без знання ключа($\boldsymbol{x_0}$);
- встановлення чіткого періоду між якими власне і відбувається генерування послідовності.

Набув додаткових знань про взаємозв'язок методів і полів, розробляючи додаток на мові C#, поглибив знання про роботу з консоллю, та набув додаткових навичок у розробці алгоритму генерування чисел.

# РОЗДІЛ 3
# ТЕСТУВАННЯ ГЕНЕРАТОРА

## 3.1 Оцінка захищеності ГПВЧ

Оскільки, значення для послідовностей обиралися за описаними вище властивостями, та кожна послідовність має п’ятий потенціал, можна вважати, що алгоритм в цілому має п’ятий потенціал, що є необхідною умовою для криптостійкого генератора. Множина ключів рівна значенню першого періоду і дорівнює 259200, при умові, що значення для конгруентних послідовностей строго визначені. Для злому звичайної конгруентної послідовності зловмиснику достатньо двох значень послідовності, якщо ж алгоритм отримання послідовності даного ГПВЧ, йому невідомий, то він повинен заволодіти всіма 9-ма значеннями. Навіть, якщо зловмисник знатиме і алгоритм і значення для послідовностей, то для перебору всіх варіантів, йому знадобиться 9 днів. З розрахунку, що для одного запуску програми необхідно 3 секунди, як показано на рисунку (3.1)

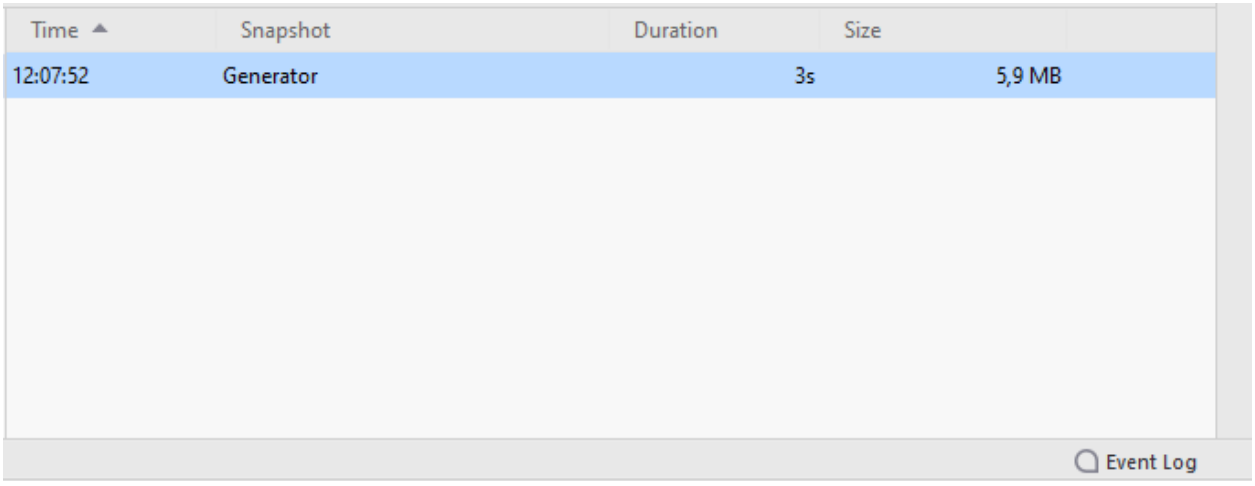



Рисунок 3.1 – Необхідна кількість часу, для одного запуску програми

### 3.2 Період зациклювання генератора

Як було описано у першому розділі, основний недолік конгруентної послідовності – зациклювання. Тобто після генерування послідовності довжиною, яка дорівнює періоду, генератор починає відтворювати одну і туж послідовність. Данна реалізація ГПВЧ була, також, напрямлена на збільшення цього самого періоду зациклювання. І на даний момент вона дорівнює не періоду однієї з послідовностей, а найменшому спільному кратному періодів всіх конгруентних генераторів. На прикладі обраних мною значень трьох конгруентних генераторів для даної реалізації, вирахуємо період зациклювання. Розкладемо період трьох генераторів на прості множники:

$259200 = 2^7 * 3^4 * 5^2$,

$139968 = 2^6 * 3^7$,

$120050 = 2 * 5^2 * 7^4$,

та вирахуємо найменше спільне кратне

$НСК_{(259200,\ 139968,\ 120050)} = 2^7 * 3^7 * 5^2 * 7^4 = 16803158400.$

Тобто, тільки після генерування 16803158400 символів ГПВЧ почне зациклюватись і відтворювати ту ж саму послідовність. Звісно, значення періоду зациклювання залежить від підбору параметрів до конгруентної послідовності. Тому при необхідності, генерувати послідовність більшу за 16млрд чисел, можна підбором параметрів ще збільшити це значення.

### 3.3 Розподілення на площині

Для побудови графів, організуємо запис отриманих значень генератора у файл:

```
foreach (var VARIABLE in Generator(Convert.ToInt32(X0)))
{
    File.AppendAllText(@"C:\учёба\диплом\result.txt", VARIABLE + Environment.NewLine);
}
```

Рисунок 3.2 – запис послідовності у файл

Після запуску коду, та генеруванню значень послідовності, файл виглядає наступним чином(рис 3.2). Після генерування послідовності переходимо до додатку Excel, та вставляємо отримані значення у одну із колонок. Далі, вибираємо вкладку “Insert”, за цим натискаємо “Insert scatter”, та в якості даних для побудови графу вибираємо колонку, яку ми заздалегідь заповнили значеннями, отриманими з генератора. В цілому робоча поверхня в додатку Excel для побудови графів виглядає наступним чином(рис. 3.3).

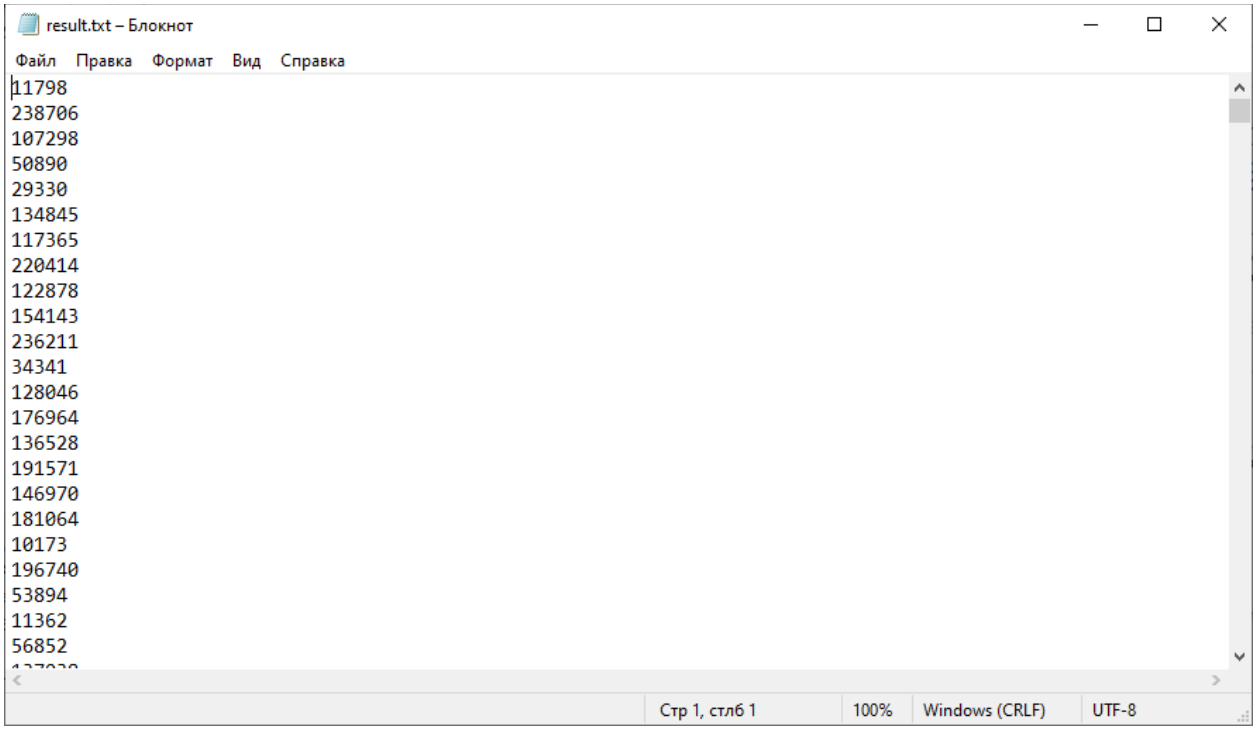



Рисунок 3.3 – Фрагмент файлу result.txt

Подивимось на розподілення чисел на графіках різної довжини та різного підбору параметра $\boldsymbol{x_0}$ (рис. 3.5 - 3.16).

З аналізу графіків 3.4 – 3.15 розподілення значень на площині, можна зробити висновок, що вони позбавленні недоліків звичайної конгруентної послідовності у вигляді короткого періоду, та «решітчастої» форми. Крім цього, точки розподілені на площині не формують узори, розподілення виглядає випадковим. Зі збільшенням довжини послідовності(рис 2.17-2.20), на площині вона набуває вигляду повністю заповненого прямокутника, що є

з однією умов проходження даного тесту(див. 1 розділ). Після аналізу, враховуючи попередньо описані факти, роблю висновок, що ГПВП створений мною, пройшов графічний тест «розподілення точок на площині».

Рисунок 3.4 – Робоча поверхня додатку Excel для створення графів

Рисунок 3.5 – Графік ГПВП для $\boldsymbol{x_0 = 15}$, довжина послідовності - 1000

Рисунок 3.6 – Графік ГПВП для $\boldsymbol{x_0 = 36}$, довжина послідовності - 1000

Рисунок 3.7 – Графік ГПВП для $\boldsymbol{x_0 = 1148}$, довжина послідовності - 1000

Рисунок 3.8 – Графік ГПВП для $\boldsymbol{x_0 = 149562}$, довжина послідовності - 1000

Рисунок 3.9 – Графік ГПВП для $\boldsymbol{x_0 = 31895}$, довжина послідовності – 10000

Рисунок 3.10 – Графік ГПВП для $\boldsymbol{x_0 = 45268}$, довжина послідовності – 10000

Рисунок 3.11 – Графік ГПВП дл $\boldsymbol{x_0 = 1}$, довжина послідовності – 10000

Рисунок 3.12 – Графік ГПВП для $\boldsymbol{x_0 = 276358}$, довжина послідовності – 10000

Рисунок 3.13 – Графік ГПВП для $\boldsymbol{x_0 = 98}$, довжина послідовності – 300000

Рисунок 3.14 – Графік ГПВП для $\boldsymbol{x_0 = 3479}$, довжина послідовності – 300000

Рисунок 3.15 – Графік ГПВП для $\boldsymbol{x_0 = 678}$, довжина послідовності – 300000

Рисунок 3.16 – Графік ГПВП для $\boldsymbol{x_0 = 83278}$, довжина послідовності – 300000

### 3.4 Гістограма розподілення елементів

Як було описано у першому розділі, тест «гістограма розподілення елементів» проводиться над числами послідовності, які були взяті по модулю 255(розмір таблиці ASCII). Для цього нам треба модифікувати виклик функції CheckForMinus і тепер цей рядок у коді виглядає наступним чином:

result = CheckForMinus(result,255);

Від тепер проміжок генерування чисел буде [1; 255]. Для побудови гістограми нам знадобиться підрахунок частоти зустрічі кожного елементу, добавимо для цієї цілі ще один блок коду:

```
Dictionary<int, int> result = new Dictionary<int, int>();
//Ініціалізація словника
for (int i = 0; i < 256; i++)
{
    result.Add(i, 0);
}
//Підрахунок частоти зустрічі кожного елементу
foreach (var VARIABLE in Generator(Convert.ToInt32(X0)))
{
    result[VARIABLE]++;
}
//Запис данних у файл
foreach (var keyValuePair in result)
{
    File.AppendAllText(@"C:\study\diplom\\result.txt", keyValuePair.Value + Environment.NewLine);
}
```

Рисунок 3.17 – блок коду для побудови гістограми

Після запуску коду, та генерації послідовності файл result.txt виглядатиме наступним чином(рис 3.18).

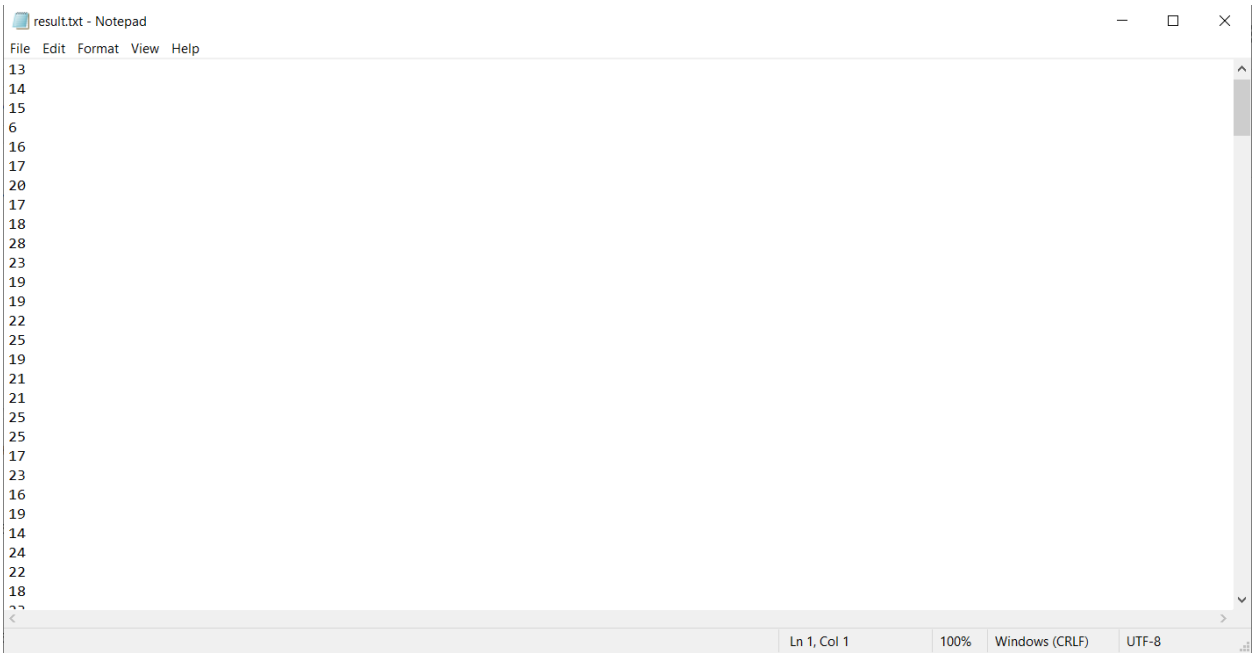



Рисунок 3.18 – Фрагмент файлу result.txt

Для побудови гістограми, за аналогією з попереднього пункту, ми маємо вставити отримані значення до однієї з колонок Excel файлу. Далі, вибираємо вкладку “Insert”, за цим натискаємо “Insert clustered column”, та в якості даних для побудови графу вибираємо колонку, яку ми заздалегідь заповнили значеннями, отриманими з генератора. В цілому робоча поверхня в додатку Excel для побудови гістограм виглядає наступним чином(рис. 3.17).

Рисунок 3.19 – Робоча поверхня для побудови гістограм в додатку Excel

Розглянемо та проаналізуємо гістограми для послідовностей різної довжини та з різними початковими значеннями $\boldsymbol{x_0}$.(рис. 3.20 – 3.27)

Рисунок 3.20 – гістограма послідовності з початковим значенням 87, довжиною 300000

Рисунок 3.21 – гістограма послідовності з початковим значенням 8596, довжиною 300000

Рисунок 3.22 – гістограма послідовності з початковим значенням 851456, довжиною 300000

Рисунок 3.23 – гістограма послідовності з початковим значенням 73568, довжиною 300000

Рисунок 3.24 – гістограма послідовності з початковим значенням 852, довжиною 1000000

Рисунок 3.25 – гістограма послідовності з початковим значенням 14253, довжиною 1000000

Рисунок 3.26 – гістограма послідовності з початковим значенням 84265, довжиною 1000000

Рисунок 3.27 – гістограма послідовності з початковим значенням 496, довжиною 1000000

З аналізу всіх отриманих гістограм, можна зробити висновок, що кожна послідовність містить у собі всі необхідні елементи з діапазону [1, 255]. При цьому розкид частот появи символів прямує до нуля. Ці умови і є необхідними для проходження тесту «гістограма розподілення елементів». З результатів цього тесту, я роблю висновок, що усі протестовані послідовності відповідали властивостям випадковості.

### 3.5 Перевірка на монотонність

Для проходження тесту на монотонність, нам знадобиться розбивати послідовність на ділянки зростання та спадання, після чого підраховувати кількість елементів у кожній ділянці. Добавимо для цієї цілі відповідний блок коду:

```
List<int> result = new List<int>();
var sequence = Generator(Convert.ToInt32(X0));
for (int i = 0; i < sequence.Count; i++)
{
    if (i == 0)
    {
        result.Add(1);
    }
    else if (i == 1)
    {
        result[0]++;
    }
    else
    {
        if (sequence[i] > sequence[i - 1] & sequence[i - 1] > sequence[i - 2])
        {
            result[result.Count - 1]++;
        }
        else if (sequence[i] < sequence[i - 1] & sequence[i - 1] < sequence[i - 2])
        {
            result[result.Count - 1]++;
        }
        else if (sequence[i] > sequence[i - 1] & sequence[i - 1] < sequence[i - 2])
        {
            result.Add(1);
        }
        else if (sequence[i] < sequence[i - 1] & sequence[i - 1] > sequence[i - 2])
        {
            result.Add(1);
        }
    }
}
foreach (var VARIABLE in result)
{
    File.AppendAllText(@"C:\study\diplom\\result.txt", VARIABLE + Environment.NewLine);
}
```

Рисунок 3.28 – блок коду для підрахунку монотонності послідовності

Логіка виконання коду наступна: спочатку генерується послідовність, далі кожний її елемент звіряється з двома попередніми, якщо тенденція росту, або спаду продовжується – до останнього елементу результуючого масиву додається одиниця, якщо ж тренд змінюється, в кінець масиву записується новий елемент - одиниця. Останнім кроком є запис результуючого масиву у файл. Створюється гістограма для перевірки монотонності послідовності точно так же, як і в попередньому пункті.

Роздивимось графіки для перевірки монотонності послідовностей з різною довжиною і різними початковими значеннями.

Рисунок 3.29 – монотонність послідовності з початковим значенням 87, довжиною 50000

Рисунок 3.30 – монотонність послідовності з початковим значенням 75321, довжиною 50000

Рисунок 3.31 – монотонність послідовності з початковим значенням 95123, довжиною 50000

Рисунок 3.32 – монотонність послідовності з початковим значенням 574, довжиною 50000

Рисунок 3.33 – монотонність послідовності з початковим значенням 4385, довжиною 100000

Рисунок 3.34 – монотонність послідовності з початковим значенням 752, довжиною 100000

Рисунок 3.35 – монотонність послідовності з початковим значенням 8342, довжиною 100000

Рисунок 3.36 – монотонність послідовності з початковим значенням 7543, довжиною 100000

З аналізу усіх графіків на дослідження монотонності послідовностей(рис.3.29 - 3.36) можна встановити чіткий період між яким послідовність зростає (спадає)[1; 8]. Також на представлених графіках відсутній різкий ріст(спад), який би виходив за встановлені межі. Всі графіки за своєю побудовою схожі на представлений приклад статистично гарної послідовності(рис. 1.1, а). З усього переліченого вище, я роблю висновок, що усі протестовані послідовності пройшли тест на монотонність.

З огляду на всі виявлені вище властивості даної функції, її можна вважати криптостійкою, при умові виконання вище описаних рекомендацій для недопущення компрометації значень конгруентних послідовностей.

### 3.6 Висновки за розділом 3

Були протестовані послідовності, які формує створений ГПВП графічними тестами, які описані у першому розділі, вдалося виявити, що властивості новоствореного генератора схожі на властивості істино випадкового генератора. Та була доказана неможливість відтворення послідовності без знання початкових значень для конгруентних

послідовностей. А при знані алгоритму, та всіх початкових значень, зловмиснику знадобиться на перебір всіх можливих варіантів, за підрахунками в останньому розділі, 9 діб. Набув додаткових знань про методи та засоби тестування псевдовипадкових послідовностей.

# ВИСНОВКИ

Аналіз публікацій, присвячених теоретичним і експериментальним дослідженням генераторів псевдовипадкових послідовностей показав, що розробка генераторів являє собою перспективний напрямок у розвитку імітаційного моделювання, шифрування даних та криптографії, а також тестуванні багатопроцесорних систем.

Однією з найважливіших завдань математичної статистики є встановлення теоретичного закону розподілу випадкової величини, що характеризує досліджувану ознаку по досвідченому (емпіричному) розподілу, який представляє варіаційний ряд.

Генерування випадкових послідовностей із заданим імовірнісним законом і перевірка їх адекватності - одні з найважливіших проблем сучасної криптології. Генератори випадкових послідовностей використовуються в

існуючих криптосистемах для генерації ключової інформації і задання ряду параметрів криптосистем.

У ході виконання дипломної роботи були розглянуті і проаналізовані основні методи генерування псевдовипадкових чисел: лінійний конгруентний метод, метод Фібоначчі, вихор Мерсенна, лінійний зсувний регістр зі зворотним зв'язком.

Апробацію керованого генератору псевдовипадкової послідовності проведено за критеріями:

- ефективна апаратна реалізація;
- швидкодія
- неможливість відтворення послідовності без знання ключа($\boldsymbol{x_0}$);
- встановлення чіткого періоду між якими власне і відбувається генерування послідовності.

Швидкодії набув мій ГПВЧ, завдяки ітеративному підходу конгруентного методу. Ефективної апаратної реалізації додаток набув завдяки розробці на об'єктно-орієнтованому язику програмування, та розведення процесів на потоки і формування послідовностей одночасно. Реалізовано також зсув кожного значення на число, яке також генерується конгруентною послідовністю, що не дає можливості відтворити послідовність без знання ключа та для кожного елементу цей зсув не є статичним. А кожний елемент залишається у межах модулю m, завдяки функції, яка контролює вихід за межі періоду, та в випадку чого – виконує додавання модулю m до елементу.

У порівнянні зі звичайним конгруентним генератором графіки розподілення значень ГПВЧ позбавленні недоліків у вигляді короткого періоду, «решітчастої» та «лінійної» форми. Було збільшено так званий період зациклювання конгруентної послідовності.

Висунуто вимоги до функціональності програми, виходячи із специфікації криптографічного генератору і сформульовані необхідні характеристики тієї мови програмування, за допомогою якої передбачалося

виконати поставлене завдання, проаналізовано переваги та недоліки різних мов і зроблений вибір на користь мови C #.

В конструкторській частині були використані нові візуальні компоненти. На основі наявних і отриманих знань про основні структури мови C # реалізовано алгоритм ГПВЧ і інтерфейс для передачі твірних параметрів і ключа.

У дипломній роботі була наведена пропозиція щодо вирішення актуальної науково-технічної задачі, яка полягає в покращенні статистичних властивостей та рівномірному розподілу у періоді елементів послідовностей псевдовипадкових чисел шляхом розробки нових та вдосконалення існуючих методів та засобів їх формування, а також у оцінюванні властивостей ПВП, графічним методом, що дає можливість виявити відсутність певних закономірностей.

Найбільш значущі результати роботи полягають у наступному:

1) Вперше було розроблено подібний метод формування псевдовипадкової послідовності на основі основного та допоміжних лінійних конгруентних генераторів, що дозволило формувати послідовність підраховуючи на кожній ітерації зсув, який не є статичний;

2) Було проведене оцінювання статистичних властивостей послідовності рівномірно розподілених чисел на основі дослідження її графічних та табличних представлень, що дозволяє виявити її статистичні відхилення;

3) Набув подальшого розвитку метод формування псевдовипадкової послідовності на основі комбінації генераторів псевдовипадкових чисел.

В результаті аналізу тестування програмної моделі генератора набори, які були отримані на виході мають наступні властивості:

- випадковість;

- рівномірний розподіл наборів в межах одного періоду;

- задане початкове значення користувачем формує нову унікальну послідовність.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Д. Є. Кнут, Мистецтво програмування, том 2 — 2001 — с. 27-49 .

2. M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator" // ACM Transactions on Modeling and Computer Simulation — т. 8 — 1998 — с. 3-30.

3. L. Blum, M. Blum, and M. Shub, "A Simple Unpredictable Pseudo Random Number Generator" // SIAM Journal on Computing, — т. 15 — 1986 — с. 364-383.

4. М.В. Лаврів і Л.Б. Петришин, "Методи і засоби генерування псевдовипадкових сигналів із рівномірним розподілом та аналіз результатів дослідження їх статистичних характеристик" // Міжнародний науково-технічний журнал “Інформаційні технології та комп ’ютерна інженерія " — № 2 — 2009 — с. 56-62.

5. Шнайер Б. 14.1 Алгоритм ГОСТ 28147-89 // Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си — 2002 — с. 373 -377.

6. Шилдг Г. Полный справочник по C# // Пер. с англ. — М. : Издательский дом "Вильямс" — 2004 — с. 752.

7. Иванов М.А., Чугунков И.В. Криптографічні методи захисту інформації в комп'ютерних системах та мережах — 2012 — 400 с.

8. Слеповичев И.И. Генераторы псевдослучайных чисел. – Саратов: СГУ — 2017 — с. 118.

9. A. Rukhin, J. Soto, J. Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications [Електронний ресурс]. –Режим доступу:
http://csrc.nist.gov/publications/nistpubs/800-22-rev1a/SP800-22rev1a.pdf

10. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. — 2002 — с. 72-81.

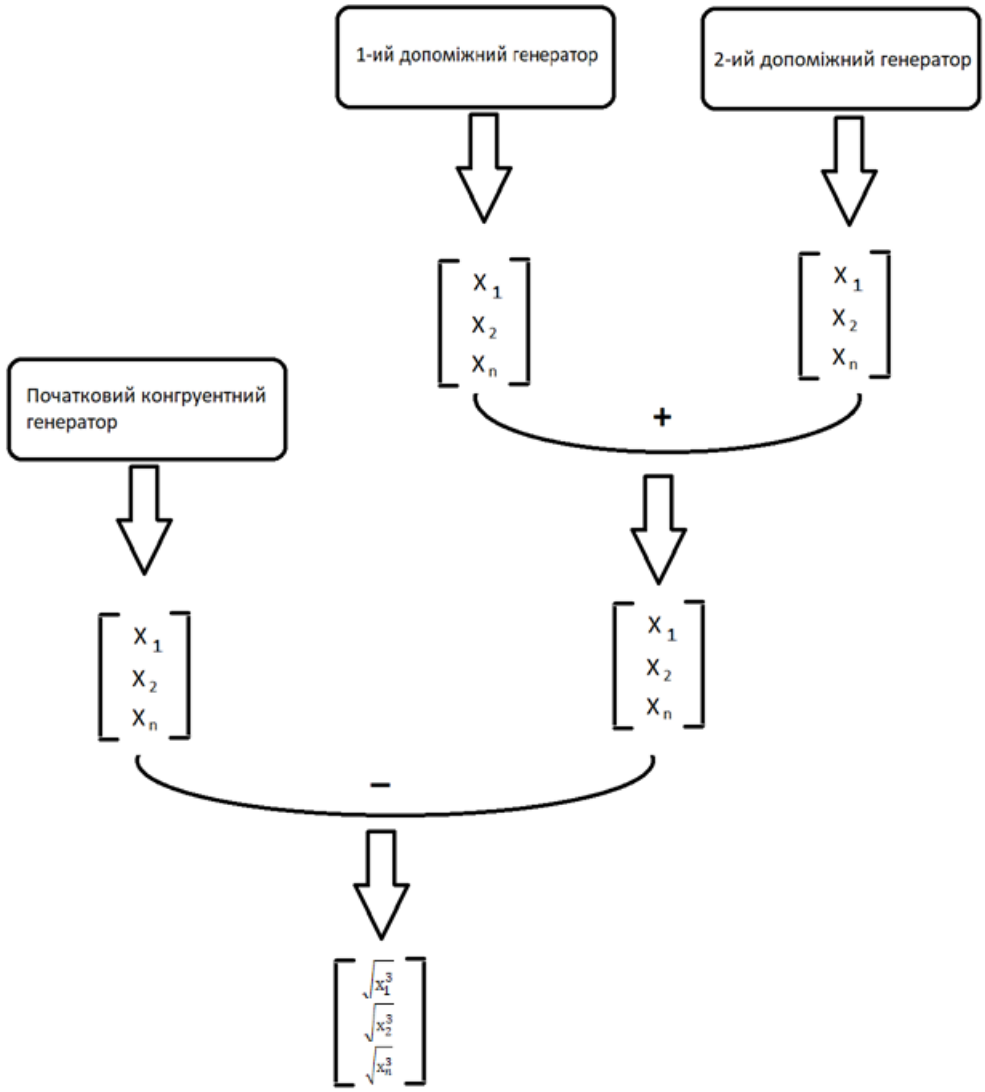
11. Жданов О.Н. Методика выбора ключевой информации для алгоритма блочного шифрования. — 2014 — с. 88

12. Дмитриев А. С. Запись и распознавание информации в одномерных динамических системах. // Радиотехника и электроника, — т.5. — 1991 — с. 101-108.

# ДОДАТКИ

**Додаток А**

Схема ГПВЧ складеного з трьох конгруентних генераторів $\sqrt{x_n^3}$




Правила підбору параметрів до конгруентних послідовностей:
• приріст c і модуль m взаємно прості;
• a - 1 кратно p для кожного простого p, що є дільником m;
• a - 1 кратно 4, якщо m кратно 4.

**Додаток Б**

Програмна реалізація ГПВЧ

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
```

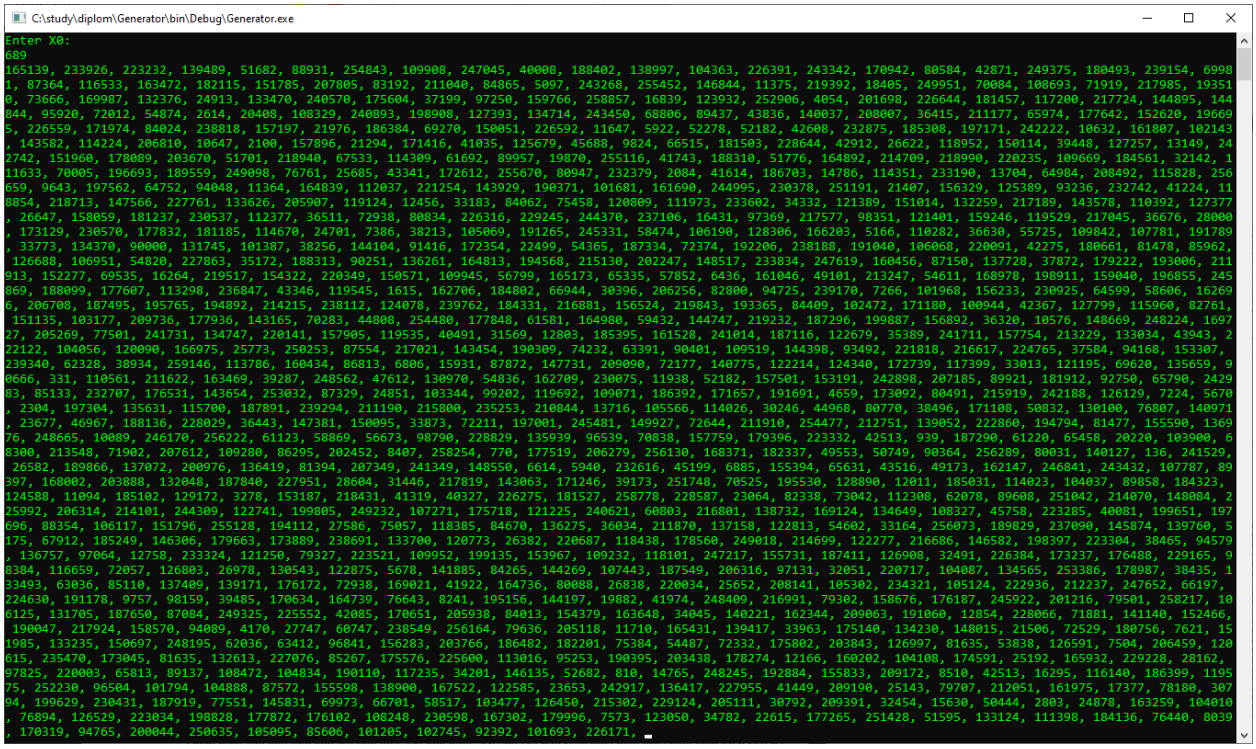
```
namespace Generator
{
    internal class Program
    {
        //Функція Main(основна точка входу для компілятора)
        public static void Main(string[] args)
        {
            //Визначення місця в пам'яті у якому зберігається змінна
            string X0 = "";
            Console.WriteLine("Введіть число X0: ");
            //Надання змінній значення через консоль
            X0 = Console.ReadLine();
            //Визов методу ГПВЧ та вивід отриманного на єкран
            foreach (var VARIABLE in Generator(Convert.ToInt32(X0)))
            {
                Console.WriteLine(VARIABLE);
            }

        }

        //Функція лінійного конгруентного методу
        static List<int> LinearGongruent(int X0, int a,int c,int m)
        {
            //Визначення місця в пам'яті для зберігання результуючої
послідовності
            List<int> resultSeqence = new List<int>();
            //Генерація циклу, де i - кількість єлементів у отриманній
послідовності
            for (int i = 0; i < 500; i++)
            {
                X0 = (a * X0) + c;
                //Додавання по модулю m
                while (X0 > m)
                {
                    X0 = X0 - m;
                }
                //Запис проміжного значення до результуючого словника
                resultSeqence.Add(X0);
            }

            //Повертає отриману послідовність з методу
            return resultSeqence;
        }

        //Функція яка приймає три конгруентні послідовності та від першої
віднімає дві інших, після чого отримане число підводиться у 3 степінь, та
береться квадратний корінь
        static List<long> MinusList(List<int> firstList, List<int>
secondList, List<int> thirdList, int module)
        {
            List<long> result = new List<long>();
            for (int i = 0; i < firstList.Count; i++)
```

```
            {
                long state;
                int temp = firstList[i] - secondList[i] - thirdList[i];
                while (temp < 0)
                {
                    temp += module;
                }

                state = Convert.ToInt64(Math.Pow(temp, 3));
                state = Convert.ToInt64(Math.Sqrt(state));
                result.Add(state);
            }

            return result;
        }

        //Функція, яка приймає послідовність отриману в попередній
функції, та значення періоду першого генератора, і перевіряє елементи
послідовності на наявність мінусу
        static List<int> CheckForMinus(List<int> minusList, int m)
        {
            //Цикл довжина якого дорівнює довжині словника
            for (int i = 0; i < minusList.Count; i++)
            {
                //Перевірка на наявність мінуса, та додавання по модулю m
                while (minusList[i] < 0)
                {
                    minusList[i] += m;
                }
            }

            //Повернення результуючого словника
            return minusList;
        }

        //Функція, яка зчитує значення послідовностей з файлу
        static int[,] ReadValues()
        {
            //Виклик будованної функції язика програмування, для
зчитування з файлу
            StreamReader sr = new StreamReader(@"C:\учёба\
диплом\data.txt");
            string text = sr.ReadToEnd();
            //Після отримання значення з файлу, розбиваємо його на числа,
які знаходяться між тире
            string[] values = text.Split(new char[] {'-'});
            //Переводимо отримані значення з массиву до словника
            List<string> valuesString = values.ToList();

            //Виділення місця в пам'яті для масиву значень
            int[,] result = new int[3,3];
            for (int i = 0; i < 3; i++)
            {
```

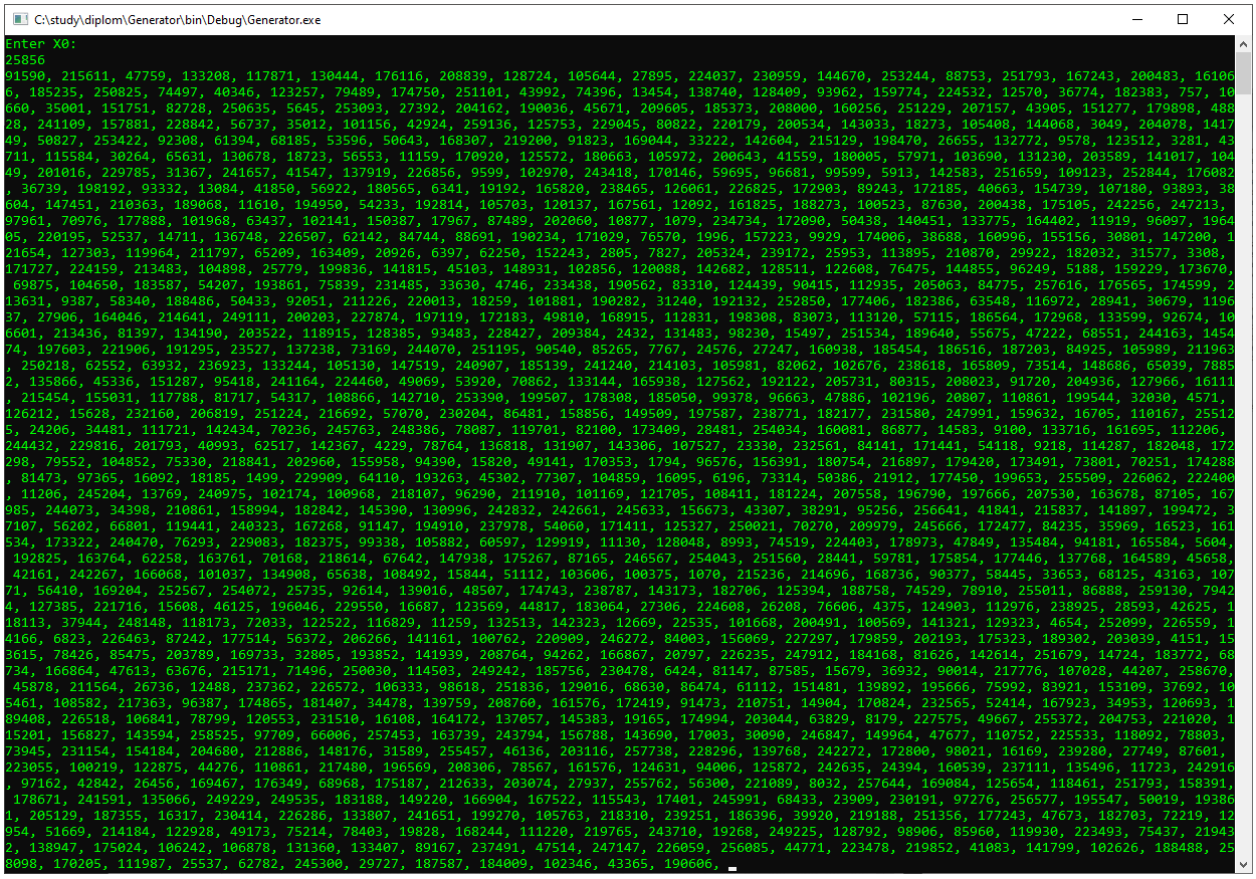
```
                for (int j = 0; j < 3; j++)
                {
                    //Записуємо значення у масив з словника
                    result[i, j] = Convert.ToInt32(valuesString[0]);
                    //Видаляємо значення з словника яке тільки що було
записане
                    valuesString.RemoveAt(0);
                }
            }

            //Повертаємо результуючий масив
            return result;
        }

        //Функція яка власне і є ГПВЧ
        static List<int> Generator(int X0)
        {
            List<int> result = new List<int>();
            int[,] values = ReadValues();
            //Викликає необхідні три конгруентні послідовності, із
значеннями отриманими з файлу
            List<int> firstList =
LinearGongruent(X0,values[0,0],values[0,1],values[0,2]);
            List<int> secondList = LinearGongruent(X0, values[1, 0],
values[1, 1], values[1, 2]);
            List<int> thirdList = LinearGongruent(X0, values[2, 0],
values[2,1],values[2,2]);

            result = MinusList(firstList, secondList, thirdList);
            result = CheckForMinus(result, values[0,2]);

            //Повертає кінцеву отриману послідовність
            return result;
        }
    }
}
```

**Додаток В**

Приклади відпрацювання програми




Послідовність ГПВЧ при $\boldsymbol{x_0 = 689}$




Послідовність ГПВЧ при $\boldsymbol{x_0 = 25856}$


Послідовність ГПВЧ при $\boldsymbol{x_0 = 20050}$


Послідовність ГПВЧ при $\boldsymbol{x_0 = 78345}$