Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет *Інформаційних радіотехнологій і технічного захисту інформації*

Кафедра *Комп'ютерної інженерії та систем технічного захисту інформації*

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти *другий (магістерський)*

Прототипування генератора псевдовипадкових чисел для системи парольного захисту

Виконав:
студент *2* курсу, групи *СТЗІАм-21-1*
Коновченко Артем Ігорович

Спеціальність *125 «Кібербезпека»*
Тип програми *освітньо-професійна*
Освітня програма *«Системи технічного захисту інформації, автоматизація її обробки»*

Керівник *доц. Горелов Д.Ю.*

Допускається до захисту
Зав. кафедри _____ *проф. Антіпов І.Є.*
(підпис)

2022 р.

Харківський національний університет радіоелектроніки

Факультет *Інформаційних радіотехнологій і технічного захисту інформації*
Кафедра *Комп'ютерної інженерії та систем технічного захисту інформації*
Рівень вищої освіти *другий (магістерський)*
Спеціальність *125 «Кібербезпека»*
Тип програми *освітньо-професійна*
Освітня програма *«Системи технічного захисту інформації, автоматизація її обробки»*

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 20 _____ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові *Коновченку Артему Ігоровичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Прототипування генератора псевдовипадкових чисел для системи парольного захисту*

затверджена наказом по університету від « 04 » 11 2022 р. № 1446 СТ

2. Термін подання студентом роботи до екзаменаційної комісії 10 грудня 2022 р.

3. Вихідні дані до роботи *Дослідити існуючі алгоритми генерації псевдовипадкових чисел та можливості використання хеш-функцій у якості генератора псевдовипадкових чисел. Для досягнення поставленої мети необхідно розв'язати наступні задачі: 1) провести аналітичний огляд сучасних алгоритмів генерації псевдовипадкових чисел; 2) дослідити можливість використання хеш-функцій у якості генераторів псевдовипадкових чисел; 3) розробити прототип, що використовує алгоритм генерації на основі хеш-функції з новою концепцією зберігання паролів*

4. Перелік питань, що потрібно опрацювати в роботі

1. Огляд сучасних алгоритмів генерації псевдовипадкових чисел
2. Дослідження можливості використання хеш-функцій у якості генераторів псевдовипадкових чисел
3. Розробка прототипу, що використовує алгоритм генерації на основі хеш-функції з новою концепцією зберігання паролів
4. Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів)

1. Прототипування генератора псевдовипадкових чисел для систем парольного захисту. А4. Ел.ф.

2. Принцип роботи генератора псевдовипадкових чисел. А4. Ел.ф.

3. Алгоритм оцінки якості генерації алгоритмів. А4. Ел.ф.

4. Результати оцінки алгоритмів генераторів псевдовипадкових чисел. А4. Ел.ф.

5. Хеш-функції у якості генератору випадкових послідовностей. А4. Ел.ф.

6. Результати оцінки алгоритмів генераторів псевдовипадкових чисел на основі хеш функції. А4. Ел.ф.

7. Вимоги та опис розроблюваного прототипу. А4. Ел.ф.

8. Архітектура розробленого прототипу. А4. Ел.ф.

9. Зовнішній вигляд розробленого прототипу. А4. Ел.ф.

10. Зміст файлу до та після розшифрування. А4. Ел.ф.

11. Блок-схеми функціональних частин. А4. Ел.ф.

12. Переваги та недоліки розробленого прототипу. А4. Ел.ф.

13. Висновки. А4. Ел.ф.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	*Огляд сучасних алгоритмів генерації псевдовипадкових чисел*	*01.09.22 – 20.09.22*	
2	*Дослідження можливості використання хеш-функцій у якості генераторів псевдовипадкових чисел*	*21.09.22 – 10.10.22*	
3	*Розробка прототипу, що використовує алгоритм генерації на основі хеш-функції з новою концепцією зберігання паролів*	*11.10.22 – 30.11.22*	
4	*Перевірка роботи на антиплагіат*	*01.12.22 – 05.12.22*	
5	*Представлення кваліфікаційної роботи на кафедрі*	*10.12.2022*	

Дата видачі завдання *01 вересня 2022 р.*

Студент _____
(підпис)

Керівник роботи _____ *Горелов Д.Ю.*
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 93 с., 34 рис., 2 табл., , 22 джерела, 3 додатки.

ГЕНЕРАЦІЯ ПАРОЛІВ, ГЕНЕРАТОРИ ПАРОЛІВ, АЛГОРИТМИ ГЕНЕРАЦІЇ, ШИФРУВАННЯ, ЗБЕРІГАННЯ ІНФОРМАЦІЇ, ПСЕВДОВИПАДКОВА ГЕНЕРАЦІЯ.

Об’єкт дослідження – алгоритми генерації паролів та методи їх обробки і зберігання.

Предмет дослідження – системи, що використовуються для генерування та зберігання паролів їх різновиди й області використання.

Методи дослідження: аналіз та узагальнення існуючих систем та алгоритмів, моделювання алгоритмів.

Генератори випадкових паролів – це програмне забезпечення, яке приймає вхідні данні від генератора випадкових або псевдовипадкових чисел та автоматично генерує пароль за заданими параметрами.

Генератори випадкових паролів часто є частиною менеджерів паролів. Менеджер паролів – це комп’ютерна програма, яка дозволяє користувачам зберігати, генерувати та керувати паролями для локальних додатків та онлайн сервісів.

Галузь використання – системи технічного захисту інформації

ABSTRACT

Explanatory note of the qualification work: 93 p., 34 fig., 2 table, 22 sources, 3 appendices.

PASSWORD GENERATION, PASSWORD GENERATORS. GENERATION ALGORITHMS, ENCRYPTION, INFORMATION STORAGE, PSEUDO-RANDOM GENERATION.

The object of research is password generation algorithms and methods of their processing and storage.

The subject of research is the systems used to generate and store passwords of their variety and area of use.

Research methods: analysis and generalization of existing systems and algorithms, modeling of algorithms.

Random password generators are software that take input from a random or pseudo-random number generator and automatically generate a password based on given parameters.

Random password generators are often part of password managers. A password manager is a computer program that allows users to store, generate and manage passwords for local applications and online services.

Field of use – technical information protection systems

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА ТЕРМІНІВ

ГПВЧ – генератор псевдовипадкових чисел

ГВЧ – генератор випадкових чисел

ПЗ – програмне забезпечення

ОС – операційна система

Хеш – рядок біт, що є вихідним результатом хеш – функції.

Хеш-функція – функція, що перетворює ряди біт у ряди біт фіксованої довжини та відповідає наступним умовам:

- за даним значенням функції складно обчислити вхідні дані, що відображені в це значення;
- для заданих вхідних даних важко знайти інші вхідні дані, що відображаються з тим самим результатом;
- важко знайти якусь пару вхідних даних із однаковим значенням хеш-функції.

ВСТУП

Пароль – умовне слово або комбінація символів, що складається з букв, цифр та інших символів, призначений для підтвердження особи або повноважень. Якщо використовуються тільки цифри, то таку комбінацію називають ПІН – кодом. Паролі використовують для захисту інформації від несанкціонованого доступу. У більшості обчислювальних систем комбінація "ім'я користувача - пароль" використовується для посвідчення користувача.

Паролі зазвичай є найпоширенішим і доступним засобом аутентифікації. Для користувачів це один із найзручніших варіантів, який, до того ж, не вимагає наявності будь-якого додаткового обладнання чи спеціальних навичок. Але автентифікацію за паролем далеко не завжди можна назвати вдалим вибором. Як і багато технічних рішень, вона страждає від декількох чинників - людського фактору та технічної недосконалості. Багато людей не можуть запам'ятовувати стійкі паролі, оскільки це об'єктивно складно. Тому вони використовують прості, нестійкі паролі, які легко зламати. Технічна недосконалість є результатом помилок під час проєктування й реалізації ПЗ, що використовується під час перевірки паролів.

Щоб уникнути цих проблеми були розроблені системи зберігання та генерації паролів. Основою системи генерації паролів є алгоритми генерування псевдовипадкових чисел або алгоритми генерування випадкових чисел за заданими користувачем параметрами. Системи зберігання паролів зазвичай використовують бази даних у яких зберігаються хеші паролів або вони шифруються за допомогою інших алгоритмів.

У даній роботі розглянуто використання таких систем та алгоритмів, які вони використовують, їх недоліки та переваги, проведено порівняння різних алгоритмів. Встановлено доцільність використання тих чи інших алгоритмів та систем на основі власних досліджень та досліджень інших

людей.

Об’єкт дослідження – алгоритми генерації паролів та методи їх обробки і зберігання.

Предмет дослідження – системи, що використовуються для генерування та зберігання паролів їх різновиди й області використання.

Методи дослідження: аналіз та узагальнення існуючих систем та алгоритмів, моделювання алгоритмів.

Галузь використання – системи технічного захисту інформації.

1 АЛГОРИТМИ ГЕНЕРАЦІЇ ПАРОЛІВ

У більшості випадків для генерації паролів використовують генератори псевдовипадкових чисел за допомогою якого робиться вибірка символів з заданого масиву символів. Генератор псевдовипадкових чисел (ГПВЧ, англ. pseudorandom number generator, PRNG, також відомий як детермінований генератор випадкових бітів DRBG) — це алгоритм для генерації послідовності чисел, властивості яких наближено до властивостей послідовностей випадкових чисел. Послідовність, згенерована PRNG, не є справді випадковою, оскільки вона повністю визначається початковим значенням, яке називається початковим значенням PRNG (яке може містити справді випадкові значення). [2]

До генераторів псевдовипадкових чисел застосовуються наступні якісні вимоги [3]:

досить довгий період, що гарантує відсутність зациклювання послідовності в межах розв'язуваної задачі. Довжина періоду має бути математично доведена;

ефективність - швидкість роботи алгоритму та малі витрати пам'яті;

відтворюваність - можливість заново відтворити раніше згенеровану послідовність чисел будь-яку кількість разів;

портованість - однакове функціонування на різному устаткуванні та операційних системах;

швидкість отримання X_{n+i} елемента послідовності чисел, при заданні X_n елемента, для i будь-якої величини; це дозволяє розділяти послідовність на декілька потоків (послідовностей чисел).

1.1 Зерно генерації

Для того, щоб ініціалізувати генератор потрібно початкове значення, яке називається зерно. Зерно – це основа генерації. Воно представляє собою число або вектор чисел, який потрібно відправити при ініціалізації генератора. Схема ініціалізації зображена на рис. 1.1.

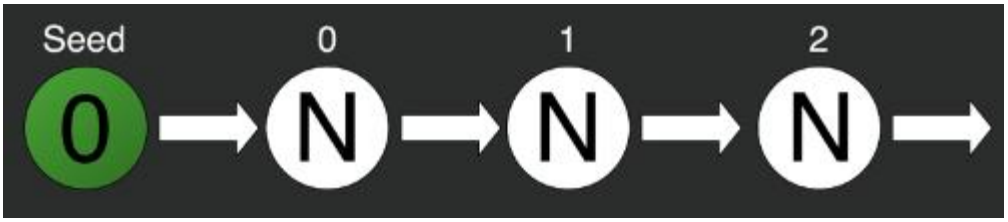

Рисунок 1.1 – Схема ініціалізації генератора

На цій схемі зображений звичайний порядок генерації чисел. Тобто, задається зерно й далі, послідовно генеруємо ряд чисел, де попереднє число є зерном для наступного. Оскільки генерація завжди послідовна, то виникає проблема. Неможливо отримати одразу i – й елемент послідовності. Для отримання другого елементу необхідно спочатку задати зерно, потім вирахувати нульовий елемент, за ним перший, потім другий і т.д. Вирішити дану проблему можливо за допомогою розділення одного генератора на декілька окремих (рис. 1.2).

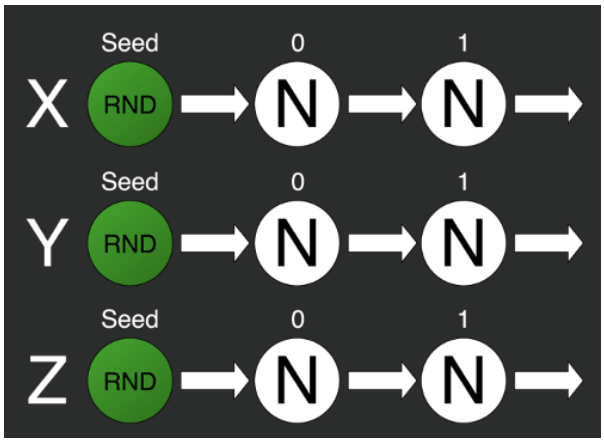

Рисунок 1.2 – Схема ініціалізаці та розділення генератора

Тобто беремо декілька генераторів та задаємо їм різні зерна. Але тут виникає друга проблема: неможливо гарантувати випадковість i – тих

елементів різних послідовностей з різними зернами (рис. 1.3)

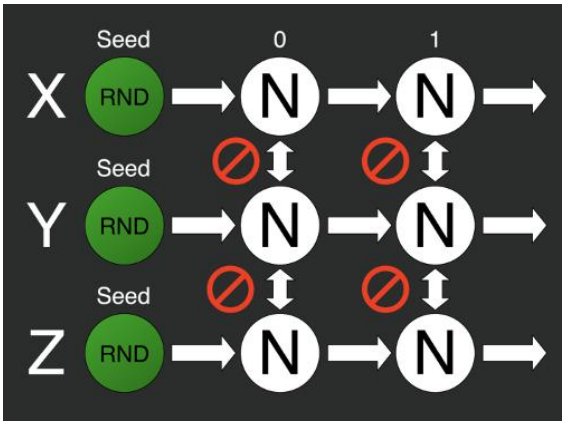

Рисунок 1.3 – Проблема розділення генератору

1.2 Оцінка якості генерації

Для перевірки якості генераторів застосовують різні тести. У світі немає жодного «офіційного» набору критеріїв, який би оцінював, наскільки дані випадкові послідовності біт застосовні саме для конкретної галузі застосування. Існують різні тести, які оцінюють, наскільки досліджувана послідовність біт «схожа» чи «не схожа» на справді випадкову. Методи оцінки якості генераторів випадкових та псевдовипадкових послідовностей можна розділити на дві групи:

1) Графічні тести. Властивості послідовностей відображаються у вигляді графічних залежностей, за виглядом яких роблять висновки про властивості досліджуваної послідовності. До цієї категорії можна віднести такі тести: гістограма розподілу елементів послідовності, розподіл на площині, перевірка на монотонність і т.д.

2) Статистичні випробування. Статистичні властивості послідовностей визначаються числовими характеристиками. На основі оціночних критеріїв робляться висновки про ступінь близькості властивостей аналізованої та випадкової послідовностей. На відміну від графічних тестів, де результати інтерпретуються користувачами, унаслідок чого можливі розбіжності у трактуванні результатів, статистичні тести характеризуються тим, що вони

видають чисельну характеристику, що дозволяє однозначно сказати, пройдено тест чи ні. [20]

Оцінка якості алгоритмів буде здійснюватися на основі графічних тестів двох видів, що згенеровані за алгоритмом, який наведений у додатку А. Перший тип – це згенерована послідовність яку візуалізовано за допомогою перших трьох байтів отриманого числа, які конвертовано в RGB – представлення (рис. 1.4).

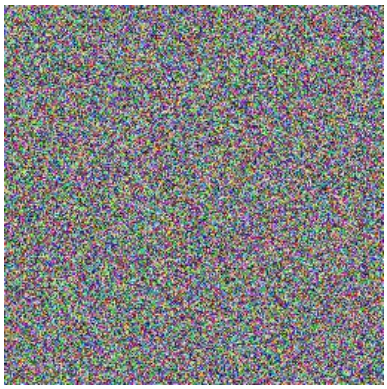

Рисунок 1.4 – Зображення першого типу

Другий тип зображення – це просторова інтерпретація згенерованої послідовності. Необхідно взяти перші два біти числа (X та Y), потім потрібно порахувати кількість потраплянь у задані точки та під час візуалізації віднімаємо з одиниці відношення кількості потраплянь в будь – який інший піксель. Чорні пікселі – це точка, де відбулося найбільше потраплянь, білі пікселі – це куди або не було потраплянь, або було дуже мало потраплянь (рис. 1.5).

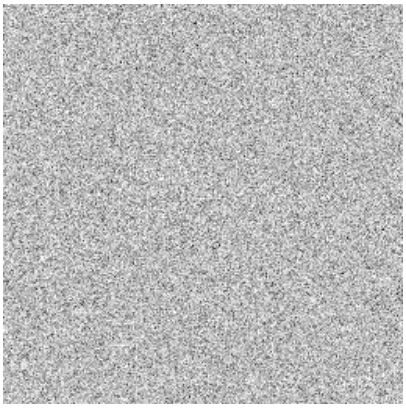

Рисунок 1.5 – Зображення другого типу

1.3 Лінійний конгруентний генератор

Лінійний конгруентний генератор (LCG) — це алгоритм, запропонований Д. Г. Лемером у 1949 році який дає послідовність псевдорандомізованих чисел, обчислених за допомогою розривного кусково-лінійного рівняння. Метод являє собою один із найстаріших і найвідоміших алгоритмів генератора псевдовипадкових чисел. Теорію, що лежить в їх основі, відносно легко зрозуміти, і вони легко і швидко впроваджуються, особливо на комп'ютерному обладнанні, яке може забезпечити модульну арифметику шляхом скорочення бітів пам'яті.

Генератор визначається рекурентним співвідношенням:

$$X_{n+1} = (aX_n + c)\ mod\ m \qquad (1.1)$$

де m – модуль (натуральне число, відносно якого розраховують залишок від ділення, $m \geq 2$), a – множник $(0 \leq a < m)$, c – приріст $(0 \leq c < m)$, X_0 – початкове значення $(0 \leq X_0 < m)$.

Лінійна конгруентна послідовність, визначена числами m, a, c та X_0 періодична з періодом, не більшим за m. При цьому довжина періоду рівна m тоді і тільки тоді, коли [4]:

числа c і m взаємно прості;

$b = a - 1$кратно p для кожного простого p, що є дільником m;

b кратно чотирьом, якщо m кратно чотирьом.

Наявність цієї властивості для випадку $m = 2^3$ де e — число бітів в машинному слові, було доведено М. Грінбергом (англ. M. Greenberg).[5] Наявність цієї властивості для загального випадку і достатність умов були доведені Т. Е. Халлом (англ. T. E. Hull) і А. Р. Добеллом (англ. A. R. Dobell) [6].

Метод генерації лінійної конгруентної послідовності при $c = 0$ називають мультиплікативним конгруентним методом, а при $c \neq 0$ — змішаним конгруентним методом. При $c = 0$ згенеровані числа будуть мати менший період, ніж при $c \neq 0$, але при певних умовах можна отримати

період довжиною $m - 1$, якщо m — просте число. Той факт, що умова $c \neq 0$ може призводити до появи більш довгих періодів, був встановлений В. Е. Томсоном (англ. W. T. Thomson) і незалежно від нього А. Ротенбергом (англ. A. Rotenberg).[4] Щоб гарантувати максимальність циклу повторення послідовності при $c = 0$, необхідно в якості значення параметра m обирати просте число. Найвідомішим генератором подібного типу є так званий мінімальний стандартний генератор випадкових чисел, запропонований Стівеном Парком (англ. Stephen Park) і Кейтом Міллером (англ. Keith Miller) в 1988 році. Для нього $a = 16807$, а $m = 2147483647$ [7].

При дослідженні алгоритму виявилося, що під час генерування чисел без розділення генератора патерн не утворюється (рис. 1.6).

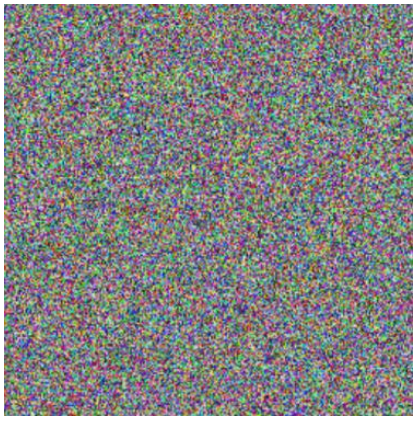

Рисунок 1.6 – Зображення першого типу для ЛКГ без розділення

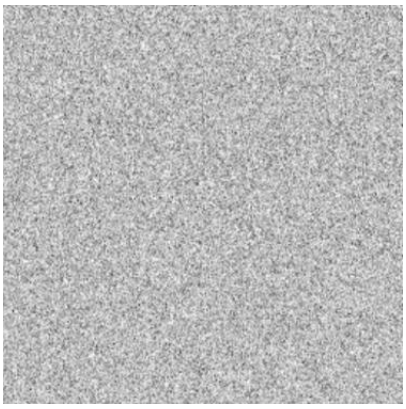

Рисунок 1.7 – Зображення другого типу для ЛКГ без розділення

Але при використанні і-тих елементів в послідовностях з різними зернами патерн починає прослідковуватися. При чому його вигляд буде

залежати виключно від коефіцієнтів, які були підібрані для генератора. Приклад наведено на рис. 1.8 та 1.9.

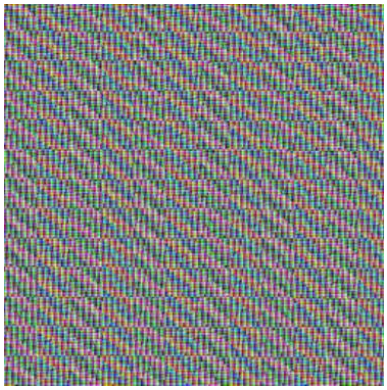

Рисунок 1.8 – Зображення першого типу для ЛКГ з розділенням

Рисунок 1.9 – Зображення другого типу для ЛКГ з розділенням

Вихор Мерсенна

1.4 Вихор Мерсенна

Вихор Мерсенна (англ. Mersenne twister, MT) — генератор псевдовипадкових чисел (ГПВЧ), розроблений 1997 року японськими вченими Макото Мацумото і Такудзі Нісімурою. Вихор Мерсенна ґрунтується на властивостях простих чисел Мерсенна і забезпечує швидке генерування високоякісних чисел за критерієм випадковості псевдовипадкових чисел.

Вихор Мерсенна позбавлений багатьох недоліків, властивих іншим

ГПВЧ, таких як малий період, передбачуваність, легко відшукувані статистичні закономірності. Проте, цей генератор не є криптостійким, що обмежує його використання в криптографії. Існують принаймні два загальних варіанти алгоритму, що відрізняються тільки величиною використовуваного простого числа Мерсенна, найпоширенішим з яких є алгоритм MT19937, період якого становить $2^{19937} — 1$ (приблизно $4{,}3 \cdot 10^{6001}$) [8].

Для w-розрядної довжини слова Вихор Мерсенна генерує цілі числа в діапазоні $[0.2^w - 1]$.

Вихор Мерсенна базується на лінійній рекуррентній матриці над скінченним бінарним полем F_2. x позначатимемо w-вимірні вектори над полем $F_2 = \{0,1\}$, відповідні машинним словам розміру w. Вихор Мерсенна генерує послідовність векторів, які є псевдовипадковими цілими з діапазону від 0 до $2^w — 1$. Діленням на $2^w — 1$ можна також отримати псевдовипадкове дійсне число з діапазону [0,1].

Також є обмеження, що 2^{nw-r} є простим числом Мерсенна. Цей вибір спрощує перевірку примітивності та перевірку k-розподілу, які необхідні для пошуку параметрів.

Ряд x визначається як ряд w -бітових величин із рекурентним відношенням:

$$x_{k+n} := x_{k+m} \oplus ((x_k{}^u \mid x_{k+1}{}^l)A) \qquad k = 0,1 \ldots \qquad (1.2)$$

n — ціле, яке позначає порядок рекурентності;

m — ціле, $1 \leq m < n$;

A — матриця розміру w×w, з елементами з F_2;

| — побітове АБО (OR);

$\oplus$ — додавання за модулем два (XOR).

У правій частині $x_k{}^u$ позначає «старші $w - r$ біти» x_k і $x_{k+1}{}^l$ «молодші r біт» x_{k+1}. Вектор $(x_k{}^u \mid x_{k+1}{}^l)$ є конкатенацією старших $w - r$ біт x_k і молодших r біт x_{k+1}. Взявши $(x_0, x_1, \ldots, x_{n-1})$ для початкового заповнення.

Тоді генератор обчислить x_n за рекурентним виразом при $k = 0$. Для $k = 1,2,\dots$, генератор обчислить $x_{n+1}, x_{n+2}, \dots$ Форму матриці А обрано з розрахунку швидкості виконання множення на А (рис 1.10).

$$A = \begin{pmatrix} 0 & 1 & & & & \\ 0 & 0 & 1 & & & \\ 0 & \cdots & \cdots & \ddots & & \\ & & & & \ddots & \\ & & & & & 1 \\ a_{w-1} & a_{w-2} & \cdots & \cdots & \cdots & a_0 \end{pmatrix}$$

Рисунок 1.10 – Матриця А

І обчислення xA зводиться до побітових операцій:

$$xA = \begin{cases} x \gg 1 & x_0 = 0 \\ (x \gg 1) \oplus a & x_0 = 1 \end{cases} \quad (1.3)$$

де

$$x := ({x_k}^u \,|{x_{k+1}}^l) \quad k = 0, 1, \dots$$
$$a := (a_{\omega-1}, a_{\omega-2}, \dots, a_0) (1.4.3)$$
$$x := (x_{\omega-1}, x_{\omega-2}, \dots, x_0)$$

Необроблені послідовності, що генеруються рекурсією (1.4.1), мають поганий рівномірний розподіл на великих розмірностях. Щоб це виправити, використовується метод загартування (англ. tempering), на виході якого отримують кінцеву псевдовипадкову послідовність. Метод полягає в тому, що кожне згенероване слово множиться праворуч на спеціальну оборотну матрицю Т розміром $w \times w$. Для матриці $T: x \to z = x\,T$, вибрано такі послідовні перетворення:

$$\mathrm{y} := \mathrm{x} \oplus (\mathrm{x} >> u)$$
$$\mathrm{y} := :\mathrm{y} \oplus ((\mathrm{y} << s) \,\&\, b)$$
$$\mathrm{y} := :\mathrm{y} \oplus ((\mathrm{y} << t) \,\&\, c)$$
$$\mathrm{z} := \mathrm{y} \oplus (\mathrm{y} >> l) \quad (1.4)$$

де l, s, t і u — цілі, а b і c — спеціально підібрані бітові маски розміру

слова, і $(x \gg u)$ позначає побітову операцію зсуву вправо на u бітів [9].

Приклад патернів генерування під час одиничного генерування наведено на рис 1.10 та 1.11

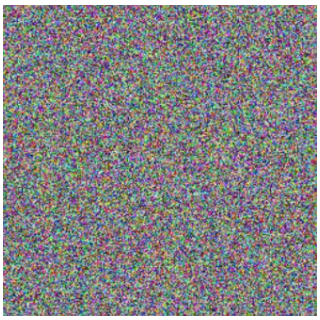

Рисунок 1.10 – Зображення першого типу для Вихору Мерсенна без розділення

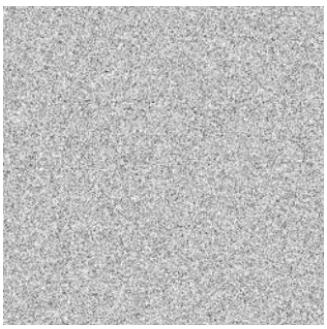

Рисунок 1.11 – Зображення другого типу для Вихору Мерсенна без розділення

Для згенерованих рядів з використанням розділення зображення патерну наведені на рис. 1.12 та рис. 1.13.

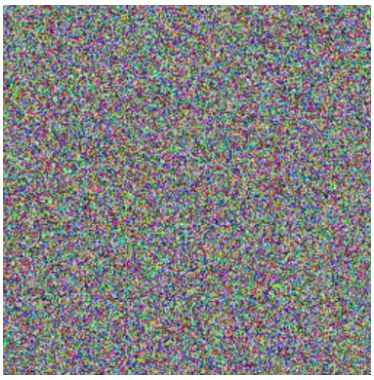

Рисунок 1.12 – Зображення першого типу для Вихору Мерсенна з розділенням

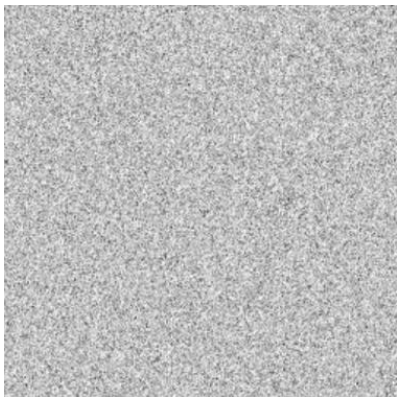

Рисунок 1.13 – Зображення другого типу для Вихору Мерсенна з розділенням

1.5 XorShift

Xorshift – клас генераторів псевдовипадкових чисел, відкритих Джорджем Марсал'я [10]. Генератори такого типу є підмножиною регістрів зсуву з лінійним зворотним зв'язком (LFSR), що дозволяє ефективно реалізувати їх без надмірного використання розріджених многочленів[11]. Генерація наступного числа в послідовності відбувається шляхом багаторазового обчислення виключающого «АБО» поточного числа та його бітового зсуву, що робить xorshift надзвичайно швидкими на сучасних комп'ютерних архітектурах. Як і всі LFSR, xorshift вимагають ретельного підбору початкових параметрів для отримання більш довгих періодичних послідовностей [12].

Генератори Xorshift є одними з найшвидших криптографічно нестійких генераторів випадкових чисел, а їх реалізація не передбачає великих обсягів коду або стану системи, що зберігається. Хоча «в сирому вигляді» вони не проходять усі статистичні тести випадковості, цей недолік добре відомий і легко виправляється шляхом додавання до їхньої структури нелінійної функції, в результаті чого виходять такі генератори як xorshift+ або xorshift*. Реалізація генератора xorshift+ мовою C, яка проходить усі тести з набору BigCrush, зазвичай вимагає менше 10 тактів на x86 для генерації випадкового числа завдяки конвеєрній обробці команд [13] .

Скремблери, відомі як + і *, слабкі в молодших бітах [14] і призначені для генерації чисел з плаваючою комою, оскільки перетворення випадкового числа на дійсне відкидає молодші біти. У загальному випадку скремблер** (вимовляється як "starstar") дозволяє LFSR проходити тести на всіх бітах.

Оскільки прості генератори xorshift (без нелінійного етапу) не проходять кілька статистичних тестів, вони вважаються ненадійними. Код генератору наведений у додатку А.

Під час використання одиничного генератора не виявлено ніяких патернів або повторювань, але під час використання розділеного генератора чітко прослідковується патерн, що властиво для багатьох генераторів. Результати візуалізації можна побачити на наступних рисунках:

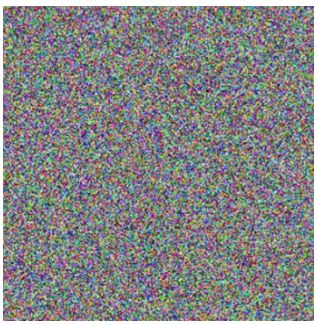

Рисунок 1.14 – Зображення першого типу для XorShift без розділення

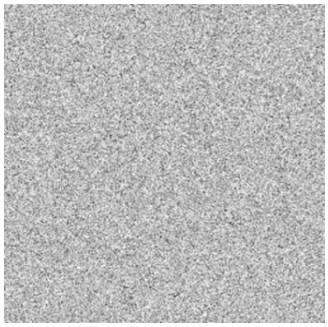

Рисунок 1.15 – Зображення другого типу для XorShift без розділення

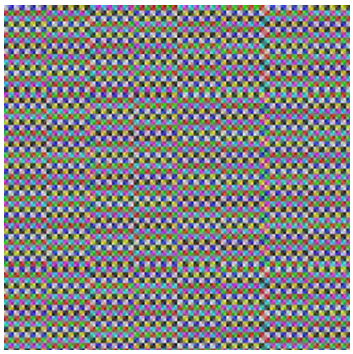

Рисунок 1.16 – Зображення першого типу для XorShift з розділенням

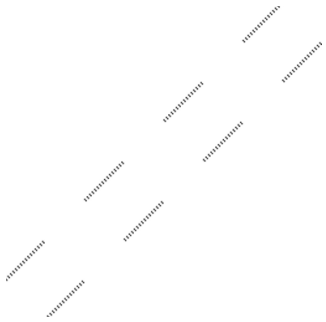

Рисунок 1.17 – Зображення другого типу для XorShift з розділенням

1.6 Тривалість послідовного генерування

Однією з важливих характеристик є триалість послідовного генерування чисел, що описує яку кількість часу генератор може генерувати послідовність чисел до першого повтору. У таблиці 1.1 наведено тривалість для алгоритмів у мілісекундах. Заміри проводились на ноутбуку Acer Aspire A517-51G.

Таблиця 1.1 – Тривалість послідовного генерування для наведених алгоритмів

	$0 \dots n$	$0\ seed\ 0 \dots n$
ЛКГ	10	28
Вихор Мерсенна	11	1870
XorShift	7	26

Вихор Мерсенна працює найдовше, але дає якісний результат. LCG (лінійний конгруентний генератор) - це вже серйозніший алгоритм, але потрібен час на підбір потрібних коефіцієнтів, щоб отримати адекватний патерн. XorShift – найшвидший алгоритм із усіх розглянутих. Його можна використовувати там, де потрібно швидко отримати випадкове значення, але необхідно пм’ятати про яскраво виражений патерн з значенням, що повторюється.

Підводячи підсумки короткого огляду сучасних алгоритмів генераторів псевдовипадкових чисел слід зазначити, що кожен з алгоритмів має свої переваги та недоліки й потрібно обирати алгоритм в залежності від потреб, адже, у деяких випадках, довша послідовність генерування переважає, тому що саме це забезпечує надійність у системах, що генерують довгі паролі. Зважаючи на те, що для звичайного користувача довжина паролю не перевищує 8 – 12 символів[15] то найкращим варіантом є алгоритми з середнім показником, адже немає необхідності генерувати довгі послідовності.

2 ХЕШ-ФУНКЦІЇ У ЯКОСТІ ГЕНЕРАТОРУ ВИПАДКОВИХ ПОСЛІДОВНОСТЕЙ

Хеш-функція (англ. hash function від hash - "перетворювати на фарш", "мішанина"), або функція згортки - функція, що здійснює перетворення масиву вхідних даних довільної довжини у вихідний бітовий рядок встановленої довжини, що виконується певним алгоритмом. Перетворення, яке виконується хеш-функцією, називається хешуванням. Вихідні дані називаються вхідним масивом, "ключем" або "повідомленням". Результат перетворення називається "хешом", "хеш-кодом", "хеш-сумою", "зведенням повідомлення".

Хеш-функції застосовуються у таких випадках:

– при побудові асоціативних масивів;

– при пошуку дублікатів у послідовності наборів даних;

– при побудові унікальних ідентифікаторів наборів даних;

– при обчисленні контрольних сум від даних (сигналу) для подальшого виявлення в них помилок (що виникли випадково або внесених навмисно), що виникають при зберіганні та/або передачі даних;

– при збереженні паролів у системах захисту у вигляді хеш-коду (для відновлення пароля по хеш-коду потрібна функція, що є зворотною по відношенню до використаної хеш-функції);

– при створенні електронного підпису (на практиці часто підписується не саме повідомлення, яке «хеш-образ»).

У загальному випадку (згідно з принципом Діріхле) немає однозначної відповідності між хеш-кодом та вихідними даними. Значення, що повертаються хеш-функцією, менш різноманітні, ніж значення вхідного масиву. Випадок, при якому хеш-функція перетворює більш ніж один масив вхідних даних в однакові зведення, називається колізією. Імовірність виникнення колізій використовується з метою оцінки якості хеш-функцій.

Існує безліч алгоритмів хешування, що відрізняються різними властивостями. Приклади властивостей:

– розрядність;

– обчислювальна складність;

– криптостійкість.

Вибір тієї чи іншої хеш-функції визначається специфікою задачі, що розв'язується. Найпростішим прикладом хеш-функції може бути «обрамлення» даних циклічним надлишковим кодом (CRC, cyclic redundancy code).

Оскільки хеш-функція – це функція перетворення, то її також можна використовувати для генерації випадкових чисел. Перевагою такого використання хеш-функції є те, що для хеш-функцій характерна рівномірність розподілу, так званий лавинний ефект [16]. Це означає, що зміна малої кількості бітів у вхідному тексті призведе до лавиноподібної та сильної зміни значень вихідного масиву бітів. Тобто, всі вихідні біти залежать від кожного вхідного біта. Також перевагою використання хеш-функцій у якості псевдогенератора випадкових чисел є те, що можливо мати доступ до будь-якого елементу послідовності просто передаючи номер елементу разом з зерном у функцію (рис. 2.1).

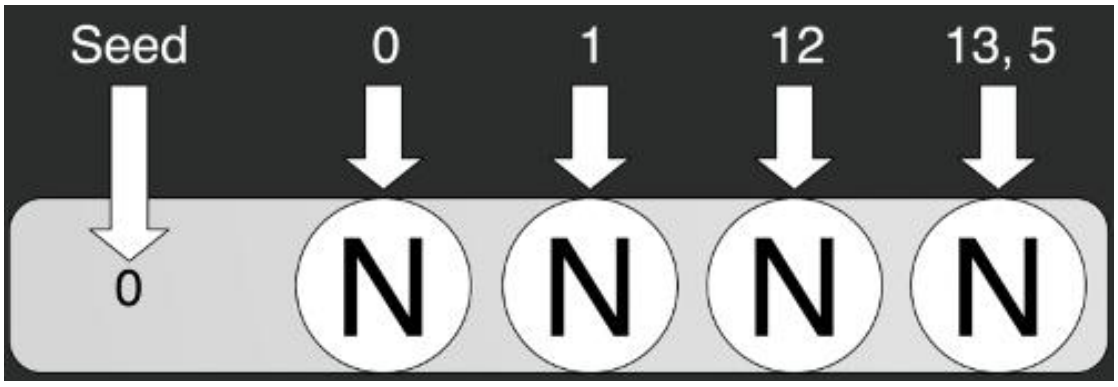

Рисунок 2.1. – Схематичне зображення доступу до елементів у хеш-функції

Вимоги до генераторів на основі хеш-функцій пред'являються ті самі, що і до простих генераторів, крім тривалості отримання послідовності. Такому генератору можна відправити на вхід одночасно зерно та необхідний

стан, тому що хеш-функції приймають на вхід масиви даних. Далі буде розглянуто декілька алгоритмів хеш-функцій, які можуть використовуватися для генерування псевдовипадкових чисел.

2.1 MD5

MD5 (англ. Message Digest 5) - 128-бітний алгоритм хешування, розроблений професором Рональдом Л. Рівестом з Массачусетського технологічного інституту (Massachusetts Institute of Technology, MIT) у 1991 році. Призначений для створення «відбитків» або дайджестів повідомлення довільної довжини та подальшої перевірки їхньої автентичності. Широко застосовувався для перевірки цілісності інформації та зберігання хешів паролів. Прийшов на зміну MD4, що був недосконалим. Описаний в RFC 1321. З 2011 року відповідно RFC 6151 алгоритм вважається ненадійним.

Хоча MD5 спочатку був розроблений для використання як криптографічна хеш-функція, було виявлено, що він страждає від численних вразливостей. Його, як і раніше, можна використовувати для розрахунку контрольної суми, для перевірки цілісності даних, але тільки від ненавмисного пошкодження. Він, як і раніше, підходить для інших цілей, не пов'язаних з криптографією, наприклад для визначення розділу для певного ключа в розділеній базі даних, і може бути кращим через нижчі вимоги до обчислень, ніж до сучасних алгоритмів безпечного хешування.[17]

MD5 перероблює повідомлення змінної довжини у вихідні дані фіксованої довжини 128 біт. Вхідне повідомлення розбивається на частини 512-бітних блоків (шістнадцять 32-бітних слів); повідомлення доповнюється так, що його довжина ділиться на 512. Доповнення працює наступним чином: спочатку один біт додається до кінця повідомлення. Потім додається стільки нулів, скільки потрібно, щоб довжина повідомлення була на 64 біта менше, ніж число, кратне 512. Решта бітів заповнюються 64 бітами, що представляють довжину вихідного повідомлення за модулем 2^{64}.

Основний алгоритм MD5 працює зі 128-бітовим станом, розділеним на чотири 32-бітові слова, що позначаються A, B, C і D. Вони ініціалізуються певними фіксованими константами. Потім основний алгоритм по черзі використовує кожен 512-бітовий блок повідомлення для зміни стану. Обробка блоку повідомлень складається з чотирьох аналогічних етапів, які називаються раундами. Кожен раунд складається з 16 подібних операцій, заснованих на нелінійній функції F, модульному складанні та обертанні вліво. На рис. 2.2. показано одну операцію в раунді.

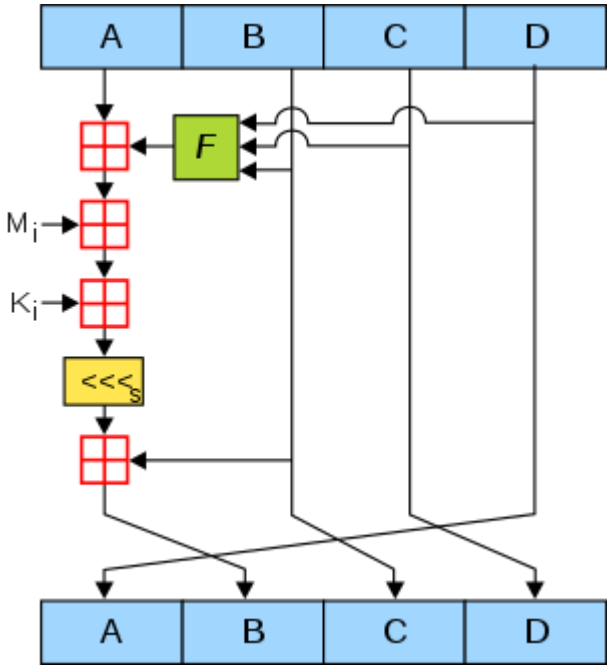

Рисунок 2.2 – Схема операції алгоритму MD5

Є чотири можливі функції. У кожному раунді використовується різна:

$$F(B,C,D) = (B \wedge C) \vee (\bar{B} \wedge D)$$

$$G(B,C,D) = (B \wedge D) \vee (C \wedge \bar{D})$$

$$H(B,C,D) = B \oplus C \oplus D$$

$$I(B,C,D) = C \oplus (B \vee \bar{D})$$

При дослідженні алгоритму не виявлено ніяких видимих патернів при генеруванні послідовності так і при досліджені частотної характеристики. (рис. 2.3, рис. 2.4.)

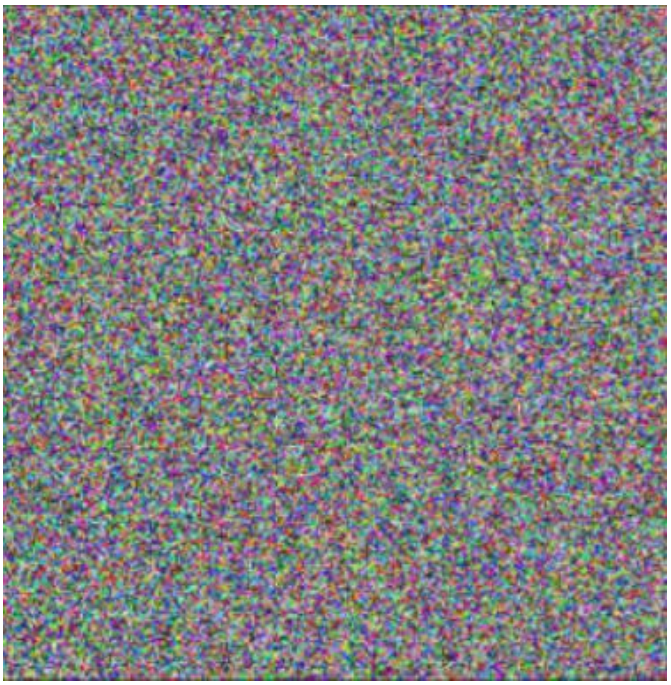

Рисунок 2.3 – Зображення першого типу

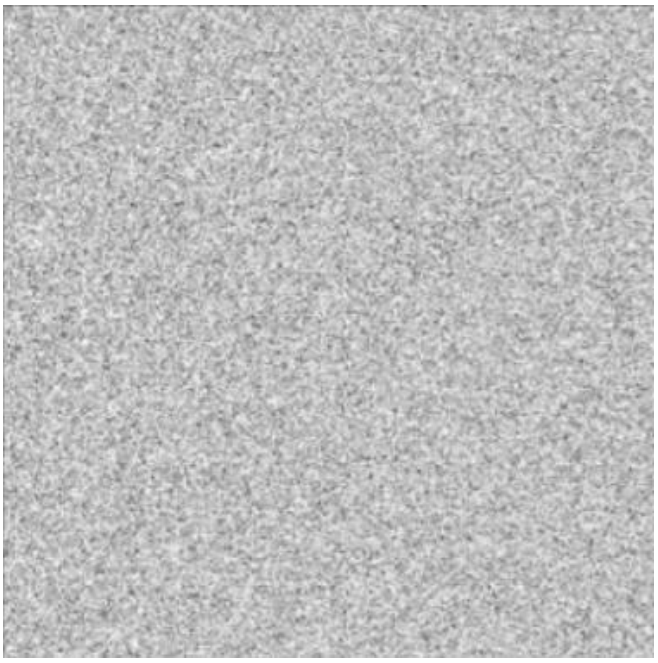

Рисунок 2.4 – Зображення другого типу

2.2 MurmurHash

MurmurHash — це некриптографічна хеш-функція, яка підходить для загального пошуку на основі хешування. Вона була створена Остіном Епплбі в 2008 році і зараз розміщена на GitHub разом із набором тестів під назвою

«SMHasher». Вона також існує в кількох варіантах, усі з яких були опубліковані у відкритому доступі. Назва походить від двох основних операцій, множення (MU) і обертання (R), які використовуються у внутрішньому циклі.

На відміну від криптографічних хеш-функцій, до цієї функції легко підібрати зворотну функцію, що робить її непридатною для криптографічних цілей. Блок-схема алгоритму зображена на рис. 2.5.

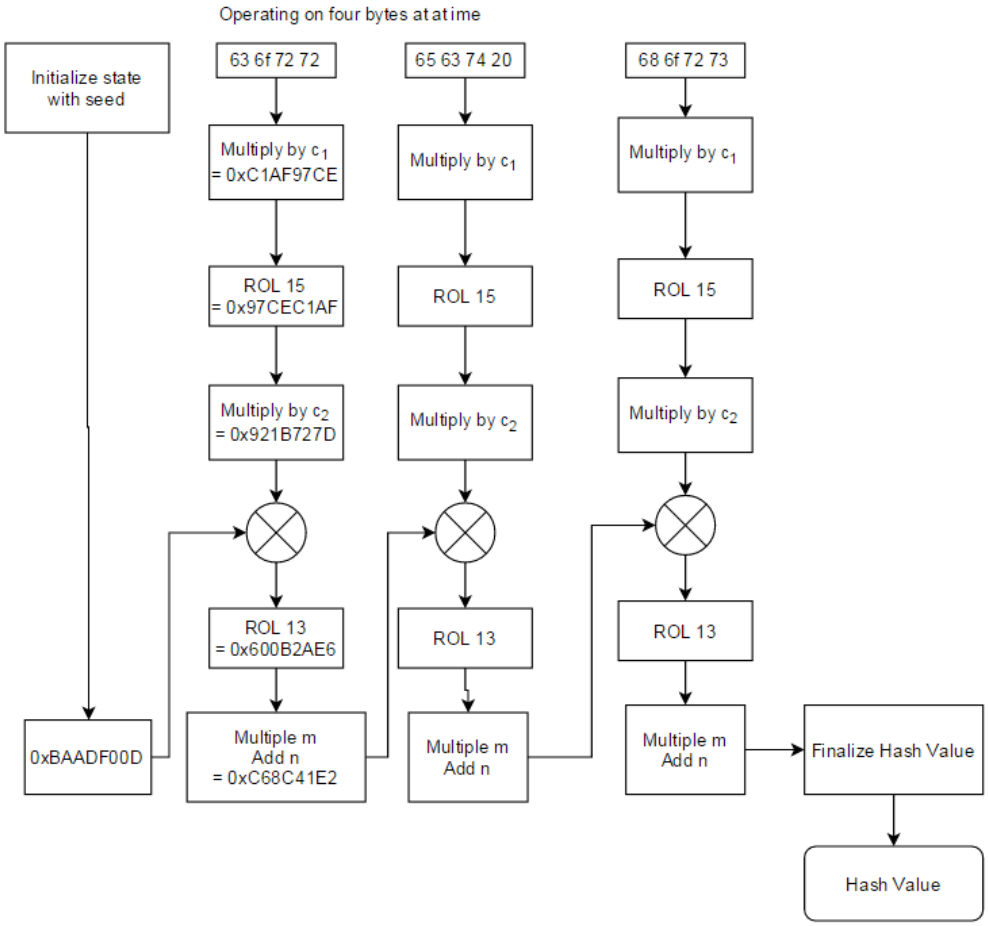

Рисунок 2.5 – Блок схема алгоритму MurmurHash

MurMurHash не створює патернів при використанні і-тих елементів різних послідовностей при різних зернах, що можна побачити на наступних рисунках.

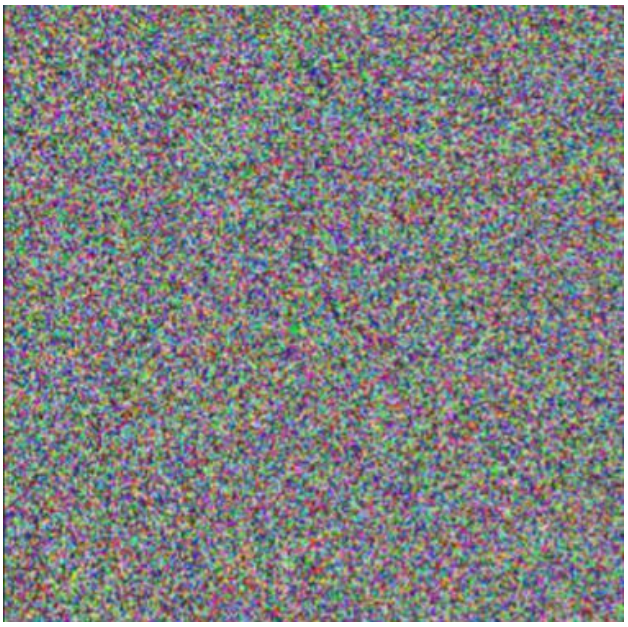

Рисунок 2.6 – Зображення першого типу

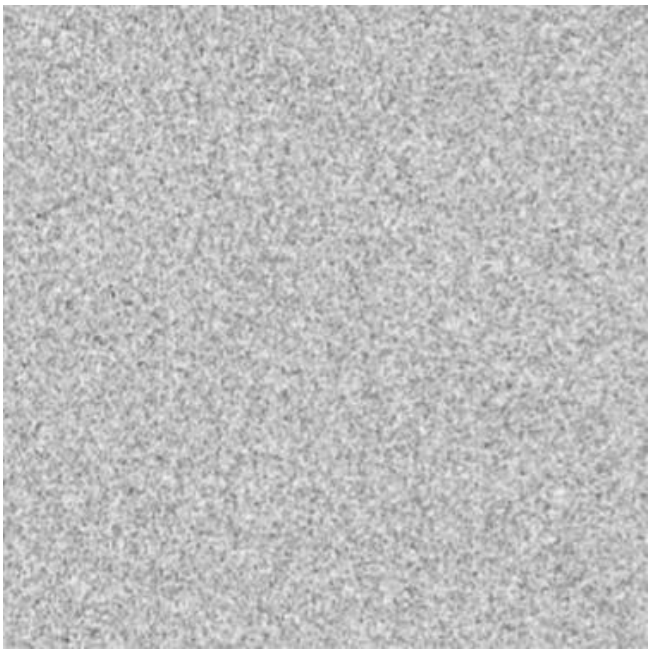

Рисунок 2.7 – Зображення другого типу

2.3 xxHash

xxHash – це надзвичайно швидкий алгоритм хешування, який обмежується лише швидкістю оперативної пам’яті. Код має високу переносимість і створює хеші, ідентичні на всіх платформах (з прямим/зворотним порядком байтів).

Це некриптографічна хеш-функція, тому, найбільшою її перевагою є швидкість обробки даних. Як стверджує автор цієї функції – це один з найшвидших алгоритмів хешування [18]

Продуктивність алгоритму може відрізнятися для різних сценаріїв, оскільки частини алгоритму, такі як ініціалізація або завершення, є фіксованими за довжиною. Вплив неправильного передбачення переходів також стає набагато помітнішим.

XXH3 був розроблений для забезпечення відмінної продуктивності як на довгих, так і на малих входах, що можна побачити на наступному графіку [19]:

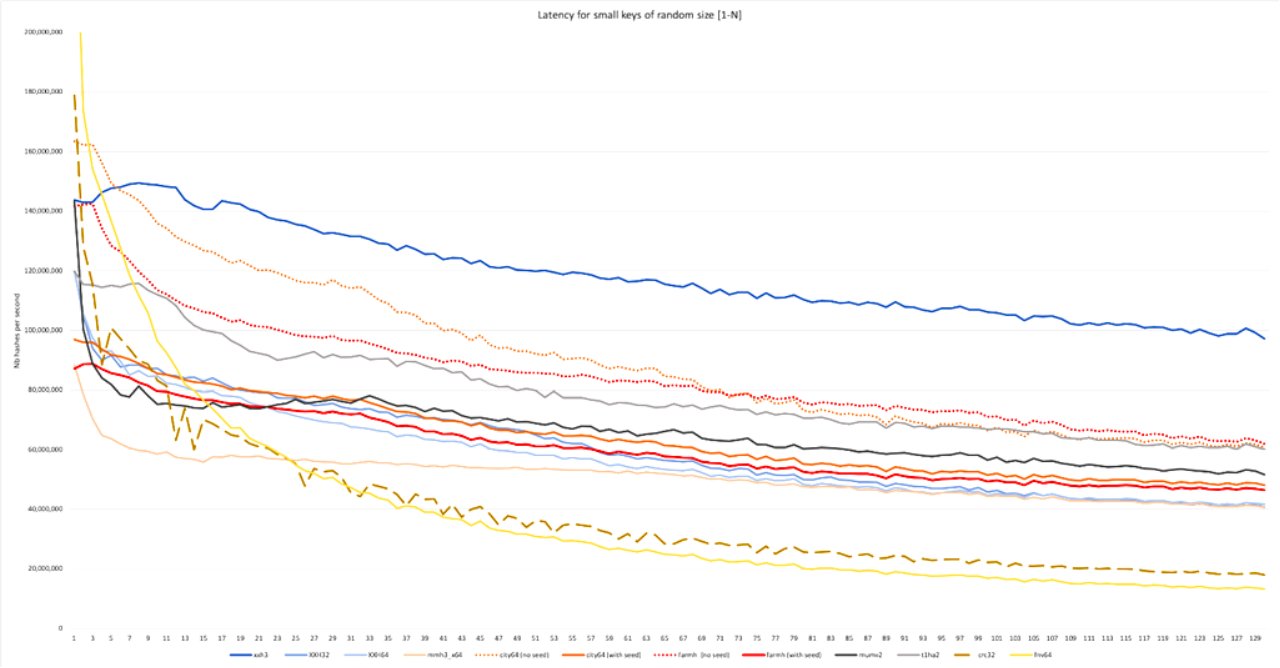

Рисунок 2.8 – Графік швидкості алгоритму

Швидкість – не єдина важлива властивість. Створювані хеш-значення повинні відповідати відмінним властивостям дисперсії та випадковості, щоб будь-який його підрозділ можна було використовувати для максимального розширення таблиці або індексу, а також зменшення кількості колізій до мінімального теоретичного рівня відповідно до парадоксу дня народження .

xxHash був протестований за допомогою набору тестів SMHasher від Austin Appleby і проходить усі тести, забезпечуючи прийнятний рівень

якості. Він також проходить розширені випробування нових форків SMHasher з додатковими сценаріями та умовами.

Нарешті, xxHash надає власний масовий тестер колізій, здатний генерувати та порівнювати мільярди хешів для перевірки меж 64-бітових хеш-алгоритмів. На цьому фронті xxHash також показує хороші результати відповідно до парадоксу дня народження.

Як і для попереднього алгоритму у xxHash не виявлено видимих патернів під час дослідження алгоритму, що можна побачити на наступних рисунках:

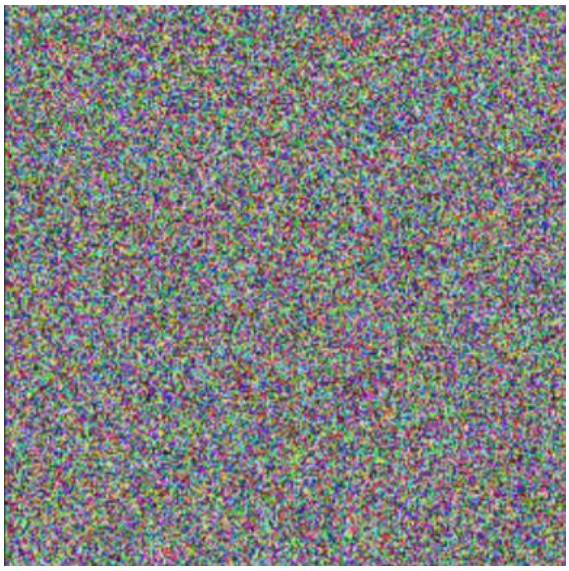

Рисунок 2.9 – Зображення першого типу

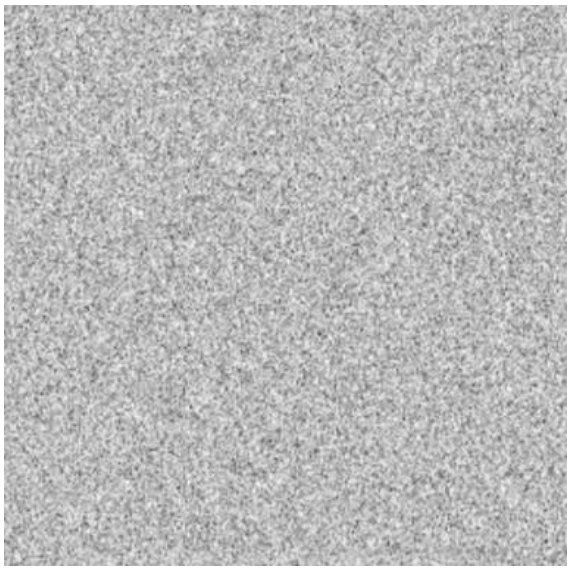

Рисунок 2.10 – Зображення другого типу

2.4 Вимірювання тривалості послідовного генерування алгоритмів

Тривалість послідовного генерування у всіх хеш-функцій приблизно однакова. Однак у MD5 ця характеристика помітно відрізняється, але не тому, що алгоритм поганий, а тому що в стандартній реалізації MD5 багато різних станів, які впливають на швидкодію алгоритму. У таблиці 2.1 наведено тривалість для алгоритмів у мілісекундах. Заміри проводились на ноутбуку Acer Aspire A517-51G.

Таблиця 2. 1 – Тривалість послідовного генерування

	$0 \dots n$
MD5	202
MurmurHash	9
xxHash	8

У даному розділі було розглянуто декілька хеш-функцій які можливо використовувати у якості генератору псевдовипадкових чисел для генератору паролів. Приклад коду використання такої функції для генерування наведено у додатку А. Перевагами використання хеш-функції у якості генератора псевдовипадкових чисел є те, що користувач може отримати доступ одночасно до багатьох елементів послідовності, що генерується, адже можливо передавати не тільки зерно для генерування але й елемент, який ми хочемо отримати.

3 РОЗРОБКА ПРОТОТИПУ МЕНЕДЖЕРУ ПАРОЛІВ НА ОСНОВІ ГЕНЕРАТОРА ПСЕВДОВИПАДКОВИХ ЧИСЕЛ З ВИКОРИСТАННЯМ ХЕШ-ФУНКЦІЇ

3.1 Вимоги та опис розроблюваного прототипу

Прототип, що розробляється в ході дипломної роботи має на меті запропонувати нову концепцію для програм – генераторів паролів. Класична програма – генератор паролів пропонує користувачу згенерувати пароль за допомогою алгоритмів, що засновані на класичних алгоритмах генерації псевдовипадкових чисел та зберігає його у базі даних у зашифрованому вигляді або користувачу необхідно самому запам’ятати пароль та зберігати. Прототип же пропонує іншу концепцію до генерації та зберігання паролю.

Основа концепції полягає у тому, що прототип генерує послідовність символів довжиною n та зберігає цю послідовність у зашифрованому файлі. Далі, за допомогою інтерфейсу програми, авторизований користувач вводить два числа, що є індексами елементів у послідовності $i, k \in n$ при чому $i > k$. Символи, що знаходяться у цьому діапазоні і є паролем, що необхідний користувачу. Перевагами такої концепції є те, що користувачу необхідно буде запам’ятати лише два числа – індекси діапазону у якому знаходиться необхідний пароль. Іншою перевагою є те, що, навіть, якщо зловмисник чи криптоаналітик дешифрують файл з послідовністю то далі необхідно буде підбирати ці два числа, що займе достатньо багато часу. Якщо використовувати перебір елементів, то його складність складає $O(n^2)$. Де n – кількість елементів у послідовності, що перебирається. Це означає, чим більша послідовність – тим більше часу буде необхідно на перебір. Виходячи з умов до програми висуваються наступні вимоги:

систему аутентифікації користувача;

система генерації послідовності та її зберігання;

система шифрування згенерованої послідовності;

система видачі паролю за вказаним діапазоном.

Блок схема взаємодії користувача з програмою зображено на рис. 3.1

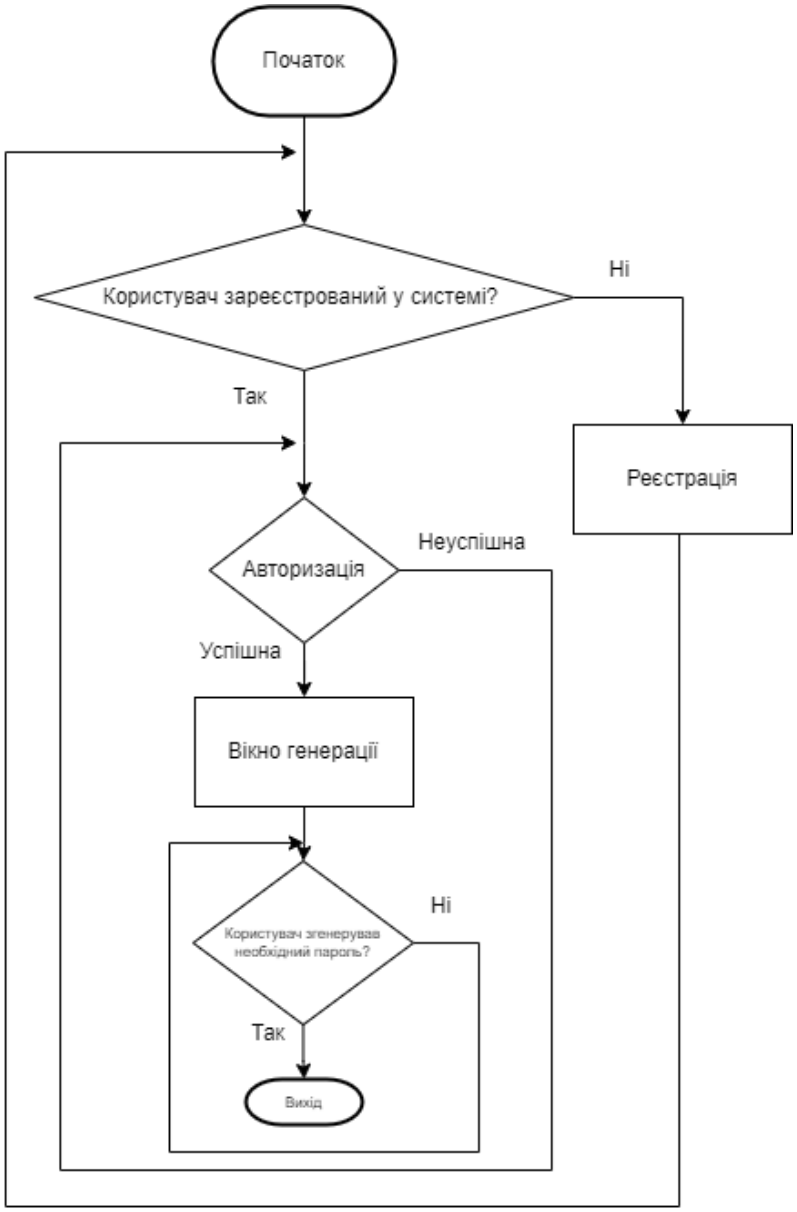

Рисунок 3.1 – Блок-схема взаємодії користувача з інтерфейсом програми

3.2 Вибір інструментальних засобів розробки

Для створення даного програмного модулю було застосовано мову програмування C#, а проєктування проводилось в середовищі Microsoft Visual Studio 2019 який є досить зручним та гнучким, та дає можливість у режимі реального часу відображати зміни що вносяться до програмного коду.

Інтегроване середовище Microsoft Visual Studio 2019 містить в собі велику кількість інструментів для створення програмних засобів, а також різноманітний вибір мов програмування. Це багатофункціональне середовище розробки, дає можливість створювати програмні засоби різних типів складності від самих простих консольних до серйозних з графічним інтерфейсом, а також веб-сайти, веб-додатки, веб-служби. Ще, можна додати, що інтегроване середовище Microsoft Visual Studio 2019 підтримує низьку кількість платформ, які підтримуються від Windows до Xbox. Потрібно зазначити, що в середовищі Microsoft Visual Studio 2019 пропонується потужний набір побудови та налагодження програмного коду.

Тож, можна вважати, що інтегроване середовище Microsoft Visual Studio 2019 досить потужне, зручне, з великим вибором можливостей та вже звичне для написання програми. При створенні графічного інтерфейсу для користувача, було використано один з типів шаблонів в середовищі Microsoft Visual Studio 2019 – Windows Forms.

Windows Forms – це ефективна платформа, завдяки якій можна створити класичний додаток за допомогою візуального конструктора у Microsoft Visual Studio 2019. За допомогою даної платформи, дуже зручно створювати додатки з графічним інтерфейсом для користувача, тому що Windows Forms володіє широкими графічними можливостями. При цьому, доступ додатків створених за допомогою Windows Forms до ресурсів на локальному комп’ютері більш надійно захищене ніж при роботі традиційних додатків Windows.

Також, треба зазначити, що при створенні Windows Forms, для її підтримки, використано .NET Framework 4.8 Дана програмна платформа, яка була випущена у 2019 році компанією Microsoft, відноситься до необхідних системних компонентів та бібліотекам без яких велика кількість програм не зможе працювати. Цей компонент є невід’ємною частиною компонентів. Він підтримує побудову та виконання додатків нового покоління.

Для проєктування вказаного програмного модулю мову програмування

C# обрано завдяки тому, що це потужна мова програмування яка може генерувати ефективні швидкісні програми.

Синтаксис C# близький до C++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML. Перейнявши багато від своїх попередників — мов C++, Object Pascal, Модула і Smalltalk — C#, спираючись на практику їхнього використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад, мова C#, на відміну від C++, не передбачає множинне успадкування класів. [21]

Для зберігання даних про користувача було використано MySQL. MySQL — вільна система керування реляційними базами даних, яка була розроблена компанією «TcX» для підвищення швидкодії обробки великих баз даних. Ця система керування базами даних (СКБД) з відкритим кодом була створена як альтернатива комерційним системам. MySQL з самого початку була дуже схожою на mSQL, проте з часом вона все розширювалася і зараз MySQL — одна з найпоширеніших систем керування базами даних. Вона використовується, в першу чергу, для створення динамічних веб сторінок, оскільки має чудову підтримку з боку різноманітних мов програмування. [22]

Тож, використовуються технології мови програмування C#, які застосовуються при створенні програми, генерування послідовності, виводу даних. Візуальні елементи які будуть використовуватись у програмі:

- кнопки;
- поля для вводу інформації користувачем;
- вивід елементів меню програми;
- та інше.

3.3 Архітектура системи

Архітектура системи насамперед це зручна архітектура, яка робить процес розробки і супроводження програми простим та ефективним.

Насамперед розроблена програма вирішує поставлені задачі, а саме генерація послідовності, зберігання послідовності у фалі, шифрування файлу, виведення символів з послідовності у заданому інтервалі.

Враховуючі складність програми при розробці дотримано принцип ієрархічної декомпозиції, тобто система програми будується з більш простих підсистем, кожна з якої, в свою чергу, будується з частин меншого розміру. Це рішення, окрім зменшення складності, забезпечує гнучкість програми, та надає можливість для масштабування.

Дивлячись на схему ми бачимо програму яка складається з набору модулів/підпрограм взаємодіючих одна з одною по простим правилам, що допомагає контролювати її складність, а також дає можливість отримати наступні переваги:

1) масштабованість - можливість розширювати систему і збільшувати її продуктивність, за рахунок додавання нових модулів;

2) ремонтопридатність - зміна одного модуля не вимагає зміни інших модулів;

3) можливість тестування - модуль можна від'єднати від всіх інших і протестувати / полагодити;

4) супровід - розбиту на модулі програму легше розуміти і супроводжувати.

Нижче, на рис. 3.2, наведено архітектуру програмного забезпечення що розроблено:

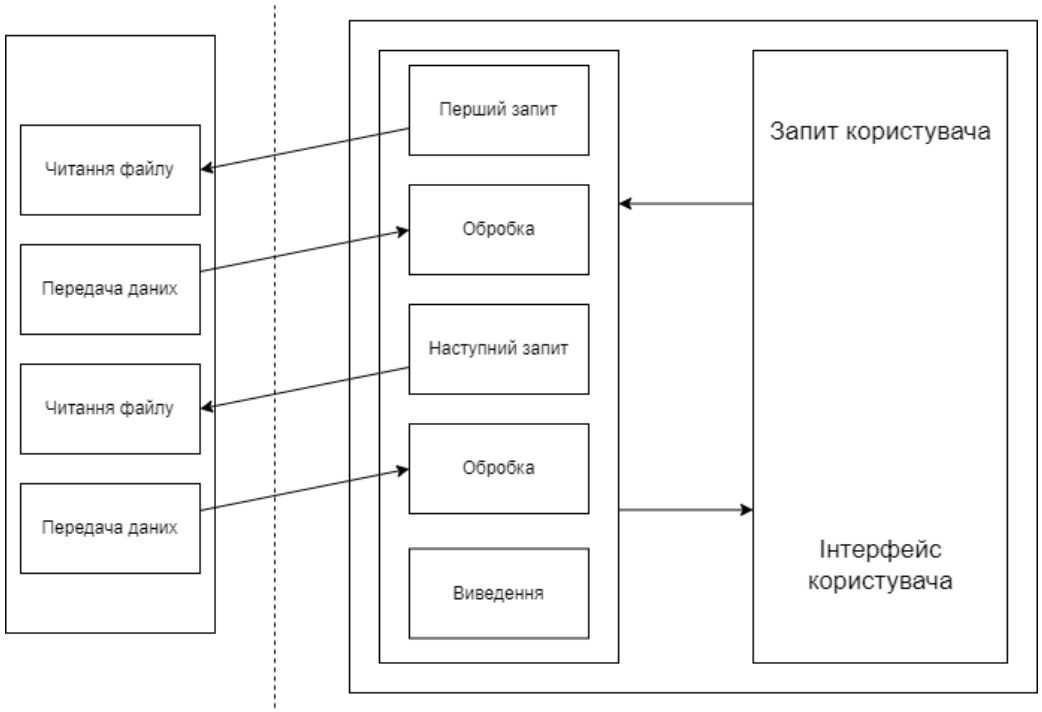

Рисунок 3.2 – Архітектура програмного забезпечення

Тут можна побачити, як взаємодіє файлова система з користувачем. Користувач надсилає запит, програма приймає його звертається до файлу або бази даних. Далі, програма передає дані які далі обробляються та виводяться у інтерфейс до користувача.

3.4 Опис інтерфейсу користувача

Програма складається з трьох основних вікон – вікна авторизації (рис 3.3), вікна реєстрації (рис. 3.4) та основного вікна програми (рис 3.5).

Вікна авторизації та реєстрації інтуїтивно зрозумілі кожному користувачу адже ні чим не відрізняються від стандартних вікон у інших застосунках.

Основне вікно складається з декількох полів:

поле для вводу індексу першого елементу вибірки послідовності;

поле для вводу індексу останнього елементу вибірки послідовності;

поле де виводиться вибрана послідовність з файлу.

Також присутні кнопки «Сгенерувати» та «Очистити поля». При натисканні кнопки «Сгенерувати» у основне поле програми виводяться елементи з заданого проміжку у послідовності. При натисканні кнопки «Очистити поля» очищаються усі без виключення поля в програмі.

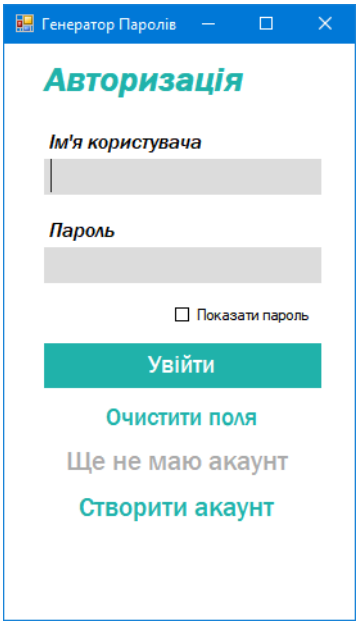

Рисунок 3.3 – Вікно авторизації

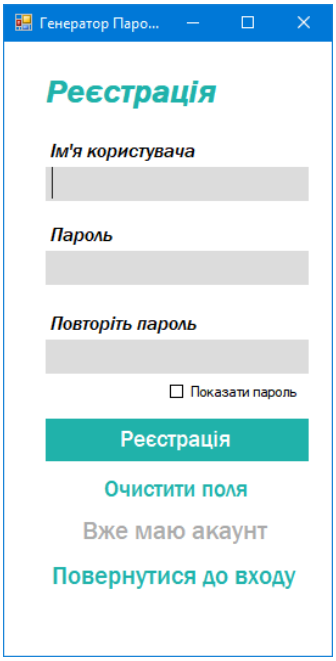

Рисунок 3.4 – Вікно реєстрації

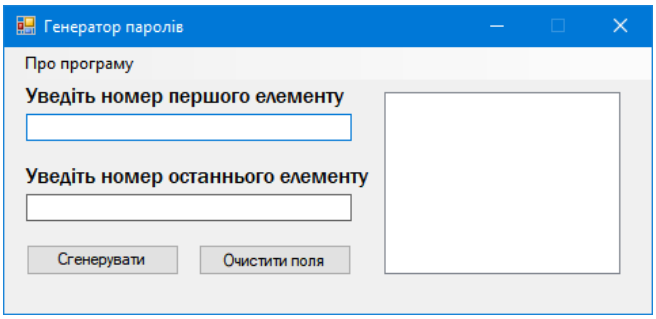

Рисунок 3.5 – Основне вікно програми

3.5 Опис функціональних особливостей програми

Основним модулем програми є модуль генерації послідовності. Цей модуль використовує алгоритм генерації псевдовипадкових чисел на основі хеш – функції MD5, що реалізована у мові програмування C#.

У програмі було реалізовано клас Generator, що використовує дану хеш – функцію у якості основи генератору. Клас має наступні поля та методи:

поле string seed для ініціалізації генератора;

поле int counter це поле дає можливість вибирати елементи не послідовно, а у довільному поряду на відмінну від класичних генераторів;

метод int Next(), що повертає згенероване значення;

метод int Next(int top), де користувач може задати верхню межу генерування;

метод int Next(int button, int top), де користувач може задати верхню та нижню межу генерування.

У методі Next() викликається метод реалізованої у C# хеш-функції ComputeHash який вираховує хеш значення для заданого масиву байтів у який передається масив байтів який задано під час ініціалізації генератору. Цей масив байтів складається з зерна та лічильника, що збільшується при кожному наступному виклику. Далі цей метод повертає значення в залежності від заданого початкового значення.

На даному етапі алгоритм генерує числа послідовно, як у випадку класичних псевдогенераторів, але можлива модифікація де для збільшення

ентропії можливо використання або поточного часу, або передавати у хеш-функцію значення з іншого класичного генератора псевдовипадкових чисел.

Генерація послідовності відбувається під час реєстрації нового користувача де у якості зерна генерації використовується поточний час користувача, після чого ініціалізований генератор генерує значення у діапазоні від нуля до шестидесяти чотирьох. Ці згенеровані значення – це індекси масиву символів з яких генерується послідовність. Після чого ця послідовність записується у файл й шифрується за допомогою за допомогою відкритого шифру AES, відкритий ключ якого зберігається у базі даних у полі з користувачем.

Вигляд зашифрованого файлу зображений на рис 3.6.

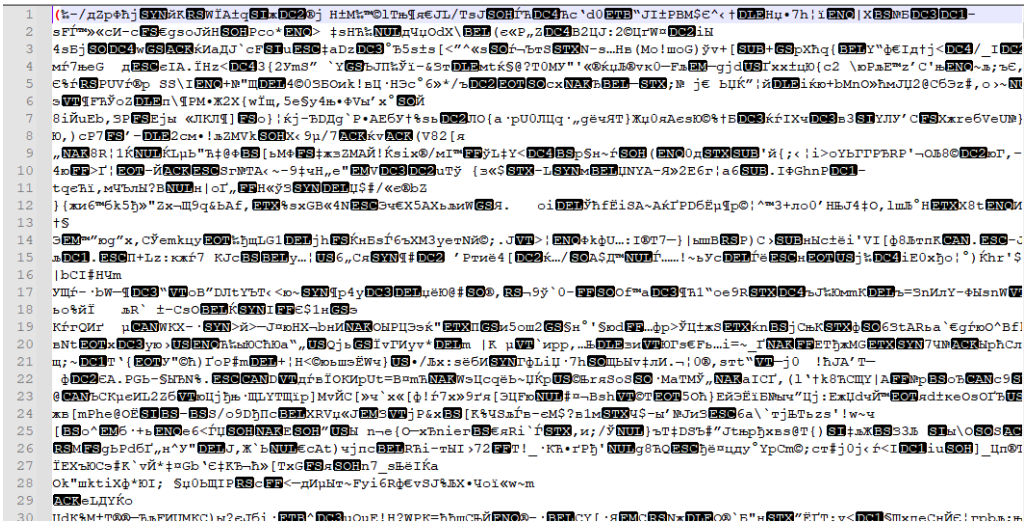

Рисунок 3.6 – Зашифрований файл з послідовністю

Після авторизації файл дешифрується и надається доступ користувачу. У дешифрованому вигляді файл має вигляд простого текстового файлу (рис. 3.7).

Детальний опис усіх компонентів у вигляді блок-схем та коду знаходиться у додатках.

```
bv38pBPARvki83ODCCBXVAkxZAf9DnC8XR3YNdwdEeB020LqVJVZ7xzBwkTW9gLU99gLVu2iZE3bIbITJfNzz3AdeXPZBbaidgSom54z6isfzGLe
Xgi37JAy2zjmLE7M0ya7Wbv38pBPARvki83ODCCBXVAkxZAf9DnC8XR3YNdwdEeB020LlRt0zL2prbTWaojzic5rjJC9QQf0ZhHAiPawttXDdizN
V4MNEw2KCVWpb0N8FvN94B9FBMDmO9q6szDrFVItaLbv38pBPARvki83ODCCBXVAkxZAf9DnC8XR3YNdwdEeB020fB57SjvK8JseiBWZ5l8EGb8y
WY5FDAZeG25cCgg2pwsh9Y5cSRYdiGcPz787UryKJYURGrIWoWVYaU6QB2ntTg2bv38pBPARvki83ODCCBXVAkxZAf9DnC8XR3YNdwdEeB020LXJ
IstpF7JcBjeAAjFjup3DoP2vB9N49aGVW4pWlRwDwc1fhIaJlHVZqchjObDoIyXqdt339RUKCFTTTbjArTOMbv38pBPARvki83ODCCBXVAkxZAf9
pcLqGHQxTtZgsQLX2btgOriwVc33yaN6rgMOtzMElUAXxJRebQE3yE1wKmjtcQxKzH4EYey2trnbgauP5Hud2dPR5ivZ6e3Tk3oF369PQbv38pBP
2y6NJxWl2lJtJ03v7J4Ec0g9z7UaaWKo3lgtMNNUOfjgygH79qn1ggegxjjUHjpckLGevRpZuhlEaco0D2ki3eVeUxqD7xdsOEeLcJfTdJFSmeu0
LqaT3CZKUsSliYGnaWPCaqaQO4eV3qCr5jIPVhWi19XgySM1I5eJZPPtAcfRU2CbUBnU2V7UCnCAnS6h7FRogeSum1bpPxXeh2Sbf7qcA14ELFtz
F4BKBPD4rsXi1GeK5ZQ2HSLWCF56wtn411L8oMOUhIVHAW1KyBBJLXp4LjpsTlCSfldIoHqa3daOCO9bv7bPbaUNuJ8aZsrYTfxqXik5b6ZAD1m9
QLttkpv2AEJ0gBZhm2OASMpEhz6pB5Mp3esexW22vMUzyCbI4L6P6MmdOHucrpqOK0InenuSwymzYCjMS4jRZ2aFJjm5n3PPGk6WziEwgKK2nQJW
2kuuGqghgv0U2X7uvfYW0auCDucpoT3lMWKyeNq08Krjlagvx FF7HPt032Adk0b7r60ieUwYLMP5YxsOxvW2Xewj1KrHWGvDrQfO5KPYplpPkMhY
Af9DnC8XR3YNdwdEeB020os1I9hBpg6omafTTdiwwxVHJmNvQjd3PHu5ZB2722f8DPlaj9u05tv5nlLjeBucuWZfP8UsWKIJsUx2BzCHPQNKaljo
pBPARvki83ODCCBXVAkxZAf9DnC8XR3YNdwdEeB020GIw10SWo5SVy9QaOiKI0QJJZuP65iQ1BH89qX5NAyqAXkwDT9hxtKsG7JXRTACorURN0GH
FEnTMZfBksuWYsjllbv38pBPARvki83ODCCBXVAkxZAf9DnC8XR3YNdwdEeB020EU6xaXbilUpn5i7rJE31n748bvEgk1OXNysAQH41f9pFdInPA
cmSvhYLbU0DzmJO19mbtfhh7ZJ8XX3wECgoMj8bv38pBPARvki83ODCCBXVAkxZAf9DnC8XR3YNdwdEeB020oJzDytk7DDZH2S2ZhZSMXWTdk7Lg
2nmExCys0gAHQE7cmj1g6SijPO4hO1WaEdsjc38aovhQwFC1kaYnFOMXJbfbv38pBPARvki83ODCCBXVAkxZAf9DnC8XR3YNdwdEeB020Vls9yAu
EbrOjjPhyrxFuaTxFAbKvpbJgoze8AVq7qy7x4SAu4RqHjPnp4GY0eeN2FyEODfU4t6ktCqrwkgHf6ADbv38pBPARvki83ODCCBXVAkxZAf9DnC8
2P4fqPS5w3ILnClLQ4R4YPPX7YNAKKP0cX5qZBL5jhpwCZiASBHV1MLhI21ZHStYg26IkbF783qnchckt7DgaWH1v3ZnHR25psicibv38pBPARvk
PxwaH8odgQFnS2WZguYN6wdw6ApispjWYYZOn1ei5yGxKH97JzzB3SV00yEMq0O4oFHZpkgtPy8yxhtp5GCeHgFGV1KiqfvBj4CCExpzNKOtQIYu
baJjrNkqNTLx2nGEtwA6hLY82wwxng52LKiVZoEMiyhly4kvyp1lKub024L48GQ4P4lsslwiutmFB2NUMlPtN2xYLAaNy7FYMbXv53wFoMlNVVHO
2FGEd1XL5HeL9CpLjH9PPUTtgtZ6RXK8tq2Wl81hd5p5eMF1YYygnImaUCPxn3AIwUmt4KSyeGylLsEpMRQdtiXmlovForTeFMVsHHl7akVliCHv
ioUe78PHdNxNF4VUaCebkdMVAd1wUSpJWVYPX0R6APtvT2FzjZJMKaDPHysn2fBdvbPr3evykSJDl9MNXMRrJAozE42bp48gLNiQIiRgttX0p21G
SgMukkEMA12sLRSHJ7DHD60n9VthpV3T0AOpTOkqOo2tY1NaBKlxVAiHd8bMAePkF0K5Q3gv6hZcrmseXjoyeeZvwKa1dz0RCfc15vG7NrJUzAuB
nC8XR3YNdwdEeB0201S2jiWSO4txfDgLDyl4QDdECvnbW9nM8fFNb3O9s0YoVfU8XaU6ORWoN8ELvscKRkPYh1JHi2mdXsZE8u6Y6yCnqqZm3WGa
Rvki83ODCCBXVAkxZAf9DnC8XR3YNdwdEeB020Noxb0MBxxFUAAdx8SgTSMdN4F9K5TjmseoGzeIKfUnbabg2U0EvVxb8qbwCCpIbBKv5eb0DywM
YsZ70w4V83D35bv38pBPARvki83ODCCBXVAkxZAf9DnC8XR3YNdwdEeB020tWB506yMxoVAfrF8EtBFjJ8vIExM5wGQparewJ6XMC58TeE2YMk8U
htNvXH7SO958XGNlW0OFaLHEwMAx8tX9A3bv38pBPARvki83ODCCBXVAkxZAf9DnC8XR3YNdwdEeB02013Xt4447d3c2X7PU7s1CA6FvwCEr8NUL
cyLDlwkJTczRqo0qxMGgUecMrwZGmuP07MMt5xsSH4KtZS3rNzRwsgnbv38pBPARvki83ODCCBXVAkxZAf9DnC8XR3YNdwdEeB020ex0Bd464PZM
5Amgq4hX1Fpr8a9UJQEFEzA9DsDWfeWN4qECO5S8WCQdiNPu5WdAVpahunRXKHLtodF7KaoRoMQ2bv38pBPARvki83ODCCBXVAkxZAf9DnC8XR3Y
mm3rGN7jtcF6DdHKkdIldjxxn9hwF4G8L9aqJiVY52GexKfDDwb0L9NfPGtCJOtJTNd7jabArrOuwv8c3NlOSpAvmKJFln3vbbv38pBPARvki83O
```


Рисунок 3.7 – Файл з послідовністю після розшифрування

3.6 Аналіз результатів отриманих під час розробки прототипу

У результаті розробки було отримано менеджер паролів, що використовує генератор на основі хеш – функції та замість класичної схеми зберігання паролів у вигляді зашифрованих значень у базі даних використовує алгоритм де генерується достатньо довга послідовність з якої потім можливо пароль необхідної довжини в наслідок чого спрощується як зберігання так і запам’ятовування паролю користувачем, адже користувачу всього необхідно запам’ятати два числа.

Прототип генерує послідовність символів довжиною n та зберігає цю послідовність у зашифрованому файлі. Далі, за допомогою інтерфейсу програми, авторизований користувач вводить два числа, що є індексами елементів у послідовності $i, k \in n$ при чому $i > k$. Символи, що знаходяться у цьому діапазоні і є паролем, що необхідний користувачу. Перевагами такої концепції є те, що користувачу необхідно буде запам’ятати лише два числа – індекси діапазону у якому знаходиться необхідний пароль. Іншою перевагою є те, що, навіть, якщо зловмисник чи криптоаналітик дешифрують файл з послідовністю то далі необхідно буде підбирати ці два числа, що займе достатньо багато часу. Якщо використовувати перебір елементів, то його

складність складає $O(n^2)$. Де $n -$ кількість елементів у послідовності, що перебирається. Це означає, чим більша послідовність – тим більше часу буде необхідно на перебір. З цього випливають наступні переваги:

користувачу необхідно запам’ятати лише два числа;

все зберігається у одному зашифрованому файлі;

простір для модифікації алгоритму генерації через використання інших генераторів або комбінацій враховуючи особливості хеш-функції.

Також, недоліками розробленого прототипу є наступне:

при використанні послідовності маленького розміру зменшується кількість варіацій доступних паролів і збільшується можливість злому;

оскільки відкритий ключ зберігається у базі даних то при зломі зловмисник зможе розшифрувати файл з послідовністю;

при пошкодженні або втраті файлу користувач втрачає усю послідовність, значить, й усі паролі.

Виходячи з вище зазначеного пропонуються наступні модифікації:

збільшення ентропії алгоритму шляхом додавання у зерно значень іншого генератора;

модифікування алгоритму шифрування файлу з послідовністю для зменшення ризиків дешифрування;

створення системи резервного копіювання та доопрацювання системи роботи з файлом.

У даному розділі розглянуто основні особливості проєктування та розробки прототипу а також вимоги, описано основні технології, що використовувалися під час розробки даного прототипу. Було описано інтерфейс програми та основний алгоритм, що використовується під час генерування послідовності та як користувачу необхідно взаємодіяти з програмою.

ВИСНОВКИ

Під час написання випускної кваліфікаційної роботи було розроблено прототип, що використовує нову концепцію для програм – генераторів паролів.

У ході написання роботи, було визначено та розкрито етапи процесу розробки самого програмного модулю. Першим етапом у ході розробки є формулювання вимог та вибір засобів розробки прототипу. Визначено архітектуру програмного модулю, що розроблено. Зазначено за допомогою яких інструментальних засобів розроблено програмний модуль та описано інтерфейс користувача. Описано алгоритм, який використовує прототип та запропоновано нові рішення для використання у менеджерах паролів. Було виявлено ряд недоліків розробленого прототипу та запропоновано подальші покращення для удосконалення

У ході дослідження було розглянуто основні алгоритми генерації псевдовипадкових чисел визначено їх переваги та недоліки й розглянуто як їх використовують при генерації паролів.

Також було розглянуто використання ряд існуючих хеш-функцій у якості генераторів псевдовипадкових чисел та використано даний алгоритм на практиці у розробленому прототипі. Перевагами використання хеш-функції у якості генератора псевдовипадкових чисел є те, що користувач може отримати доступ одночасно до багатьох елементів послідовності, що генерується, адже можливо передавати не тільки зерно для генерування але й елемент, який ми хочемо отримати.

У ході роботи було проведено оцінку якості алгоритмів на основі графічних тестів. Більшість алгоритмів мали задовільний результат та якісно генерували ряд чисел.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ДСТУ ISO/IEC 10118-1:2018 Інформаційні технології. Методи захисту. Хеш-функції. Частина 1. Загальні положення (ISO/IEC 10118-1:2016, IDT). На заміну ДСТУ ISO/IEC 10118-1:2003. Чинний від 2019-01-01. Вид. офіц. Київ : УкрНДЦ, 2018. 18 с.

2. E. Barker. Recommendation for Key Management, Part 1: General // NIST Special Publication 800-57 Part 1, Revision 4. 2015. P. 21

3. Pierre L'Ecuyer. Random Number Generation // Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice. 2007. P. 93 -137

4. Knuth E.D. The Art of Computer Programming – 3rd ed. -Boston, 2013. -652p.

5. M. Greenberger. Method in randomness, Comm // ACM 8. 1965. P 177—179.

6. T.E. Hull and A.R. Dobell «Random Number Generators» // SIAM Review 4-3. 1962. P 230-254.

7. Stephen K. Park; Keith W. Miller. Random Number Generators: Good Ones Are Hard To Find. 1988.

8. Вихор Мерсенна. [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Вихор_Мерсенна. Дата звернення: 16.10.2022.

9. M. Matsumoto, T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator // ACM Trans. on Modeling and Computer Simulations : journal. — 2017. — Vol. 8, no. 1 (3 October). — P. 3—30.

10. Marsaglia, George. "Xorshift RNGs" // Journal of Statistical Software. 2003. Vol. 8 (14).

11. Brent, Richard P. "Note on Marsaglia's Xorshift Random Number Generators". // Journal of Statistical Software. 2004. Vol. 11 (5).

12. Panneton, François; L'Ecuyer, Pierre. “On the xorshift random number generators” // Transactions on Modeling and Computer Simulation. 2005. Vol. 15 (4).

13. Vigna Sebastiano. Xorshift*/xorshift+ generators and the PRNG shootout. 2014.

14. Lemire, Daniel; O’Neill, Melissa E. “Xorshift1024*, Xorshift1024+, Xorshift128+ and Xoroshiro128+ Fail Statistical Tests for Linearity”. // Computational and Applied Mathematics. 2019.

15. Naomi B. Lefkovitz. Jamie M. Danker. Mary F. Theofanos. Yee-Yin Choong. NIST Special Publication 800-63B. Digital Identity Guidelines. Authentication and Lifecycle Management // National Institute of Standards and Technology. Gaithersburg, 2017

16. Брюс Шнаейр, Прикладна криптографія. Протоколи, алгоритми, вихідні тексти на мові Сі. Москва, 2002. 610 с.

17. M. Kleppmann Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems (1 ed.). O'Reilly Media. Boston, 2017. 203 p.

18. Performance comparison. / Github. Електронний ресурс] – Режим доступу: https://github.com/Cyan4973/xxHash/wiki/Performance-comparison #benchmarks-concentrating-on-small-data. Дата звернення: 05.11.2022.

19. Cyan4973 / xxHash / Github URL: https://github.com/Cyan4973/xxHash (дата зверення 05.11.2022)

20. Григор’єв О. Ю. Методи тестування генераторів випадкових та псевдовипадкових послідовностей // Вчені записки УлДУ. Сер. Математика та інформаційні технології. УлДУ. Електрон. журн. 2017, №1, с. 22-28.

21. C Sharp. [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/C_Sharp. Дата звернення: 25.11.2022.

22. MySQL. [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/MySQL. Дата звернення: 25.11.2022.