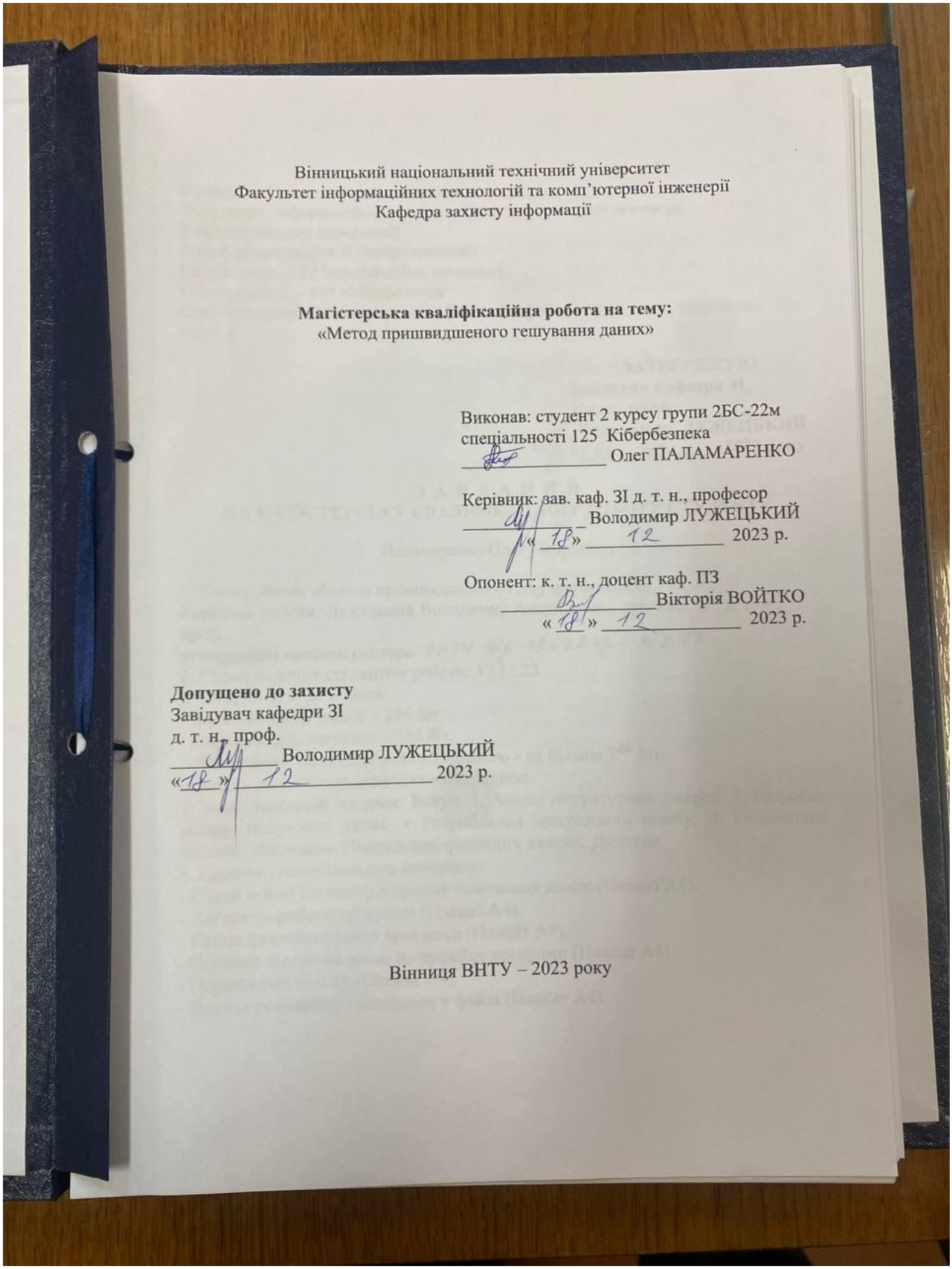


Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

Магістерська кваліфікаційна робота на тему:
«Метод пришвидшеного гешування даних»

Виконав: студент 2 курсу групи 2БС-22м
спеціальності 125 Кібербезпека
_____ Олег ПАЛАМАРЕНКО

Керівник: зав. каф. ЗІ д. т. н., професор
_____ Володимир ЛУЖЕЦЬКИЙ
«18» 12 2023 р.

Опонент: к. т. н., доцент каф. ПЗ
_____ Вікторія ВОЙТКО
«18» 12 2023 р.

Допущено до захисту
Завідувач кафедри ЗІ
д. т. н., проф.
_____ Володимир ЛУЖЕЦЬКИЙ
«18» 12 2023 р.

Вінниця ВНТУ – 2023 року

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень вищої освіти II (магістерський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 125 Кібербезпека
Освітньо-професійна програма – Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри ЗІ,
д. т. н., проф.
Володимир ЛУЖЕЦЬКИЙ
«19» 09 2023 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Паламаренко Олегу Ігоровичу

1. Тема роботи: «Метод пришвидшеного гешування даних»
Керівник роботи: Лужецький Володимир Андрійович , зав. каф. ЗІ , д. т. н., проф.,
затверджені наказом ректора. ВНТУ від 18.09.23р. №247
2. Строк подання студентом роботи: 18.12.23
3. Вихідні дані до роботи:
- довжина блоку даних – 256 біт,
- довжина геш-значення – 256 біт.
- обсяг даних, що підлягають гешуванню - не більше 2^{64} біт.
- модель даних і геш значення – кватерніон.
4. Зміст текстової частини: Вступ. 1. Аналіз літературних джерел. 2. Розробка методу гешування даних. 3. Розроблення програмного засобу. 4. Економічна частина. Висновки. Перелік використаних джерел. Додатки.
5. Перелік ілюстративного матеріалу:
- Схема однієї ітерації алгоритму гешування даних (Плакат А4).
- Алгоритм роботи програми (Плакат А4).
- Схема функціонування програми (Плакат А4).
- Основне діалогове вікно інтерфейсу програми (Плакат А4).
- Порівняльні оцінки (Плакат А4).
- Вигляд результату гешування у файлі (Плакат А4).

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Володимир ЛУЖЕЦЬКИЙ, зав. каф. ЗІ д. т. н., проф.	19.09	25.10
2	Володимир ЛУЖЕЦЬКИЙ, зав. каф. ЗІ д. т. н., проф.	19.09	10.10
3	Володимир ЛУЖЕЦЬКИЙ, зав. каф. ЗІ д. т. н., проф.	19.09	26.10
4	Володимир ЛУЖЕЦЬКИЙ, зав. каф. ЗІ д. т. н., проф.	19.09	30.10
5	Ольга РАТУШНЯК, к. т. н., доц. каф. ЕПВМ	19.09	27.10

7. Дата видачі завдання 1 вересня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання. Вступ	01.09.2023 – 10.09.2023	
2	Аналіз літературних джерел за напрямком магістерської кваліфікаційної роботи	10.09.2023 – 15.09.2023	
3	Розробка рішень	16.09.2023 – 22.09.2023	
4	Розробка модуля програмного засобу	23.09.2023 – 29.09.2023	
5	Практична реалізація, моделювання, експериментування, результати	30.09.2023 – 12.10.2023	
6	Розробка розділу тестування і обгрунтування доцільності розробки	14.10.2023 – 10.11.2023	
7	Аналіз виконання ІЗ, висновки	11.11.23 – 17.11.2023	
8	Оформлення пояснювальної записки	18.11.2023 – 24.11.2023	
9	Попередній захист та доопрацювання МКР	25.11.2023 – 30.11.2023	
10	Представлення МКР до захисту	28.11.2023 – 01.12.2023	
11	Захист МКР	14.12.2023 – 21.12.2023	

Студент _____ Олег ПАЛАМАРЕНКО

Керівник роботи _____ Володимир ЛУЖЕЦЬКИЙ

АНОТАЦІЯ

Магістерська робота присвячена розробці методу пришвидшеного гешування даних за рахунок розпаралелення обчислень, над даними, які представляються у вигляді кватерніонів. Вінниця: ВНТУ, 2023.

В першому розділі було проведено комплексний аналіз існуючих методів гешування даних, зосереджуючись на підходах. Досліджено різноманітні техніки, їх переваги та недоліки в контексті гешування даних. Наведено приклади застосування цих методів у сучасних системах з урахуванням їхньої ефективності та потенційних обмежень.

Другий розділ присвячено розробці нового методу пришвидшеного гешування на основі кватерніонів. Описано кватерніон. Описано технічні аспекти алгоритму, його унікальні особливості та потенційні переваги порівняно з існуючими методами. Наведено деталі його реалізації та можливості його використання в різних сферах.

У третьому розділі представлена практична реалізація розробленого методу гешування даних на основі моделі кватерніона. Описано програмні або апаратні аспекти впровадження алгоритму, результати тестування, включаючи ефективність та надійність, а також можливість інтеграції в існуючі системи.

Четвертий розділ містить аналіз економічних аспектів розробленого методу шифрування. Обговорюються витрати на реалізацію та впровадження, можливість зменшення витрат часу чи ресурсів завдяки новому методу, а також його конкурентоспроможність на ринку і потенційні переваги для різних секторів.

ABSTRACT

The master's thesis is devoted to the study and development of a method of accelerated data hashing by parallelising computations on data represented as quaternions. Vinnytsia: VNTU, 2023.

In the first chapter, a comprehensive analysis of existing data hashing methods was conducted, focusing on approaches. Various techniques, their advantages and disadvantages in the context of data hashing are investigated. Examples of the application of these methods in modern systems are given, taking into account their effectiveness and potential limitations.

The second section is devoted to the development of a new method of accelerated hashing based on quaternions. The quaternion is described. The technical aspects of the algorithm, its unique features and potential advantages over existing methods are described. The details of its implementation and the possibilities of its use in various fields are given.

The third section presents the practical implementation of the developed method of data hashing based on the quaternion model. It describes the software and hardware aspects of the algorithm implementation, test results, including efficiency and reliability, as well as the possibility of integration into existing systems.

The fourth section contains an analysis of the economic aspects of the developed encryption method. It discusses the costs of implementation and implementation, the possibility of reducing time or resources due to the new method, as well as its competitiveness in the market and potential benefits for various sectors.

ЗМІСТ

ВСТУП

На сьогоднішній день надшвидкими темпами розростається залежність та використання інформаційно-комунікативних систем. Саме через це зросла кількість користувачів, об’ємів інформаії, потенційних зловмисників, потреб передачі та обробки все більше та щвидше інформації, як користувачами так і різними системами включно зі штучним інтелектом, тому все це стає проблемою для стійкості, надійності, швидкості та захисту інформації.

Захист інформації включає в себе ряд заходів, спрямованих на уникнення (або зниження ризику) витоку, крадіжки, втрати, поширення, знищення, перекручування, підробки або блокування інформації. Криптографічні методи відіграють ключову роль серед різноманітних заходів захисту даних, використовуючи відкриті алгоритми шифрування, які базуються на обчислювальних засобах.

Захист інформації за допомогою криптографії передбачає використання спеціальних даних (ключів) для перетворення інформації з метою забезпечення конфіденційності, цілісності, автентичності та інших параметрів. Геш-функції широко застосовуються в криптографії, перетворюючи вхідні дані невизначеної довжини та формату у вхідне значення фіксованого розміру, яке використовується для електронно цифрового підпису та автентифікації.

Важливо враховувати, що стійкість геш-функцій прямопропорційна їх довжині, але збільшення розрядності може призвести до збільшення часу обчислень. Оптимізація апаратури може поліпшити швидкість, але також може збільшити ризик "зламання" геш-функції. Тому важливо розробляти геш-алгоритми, які забезпечують високу швидкість обчислень, не втрачаючи при цьому стійкості.

У зв'язку з необхідністю адаптації гешування для багатоядерних платформ, важливо використовувати методи підвищення стійкості до мультиколізій на етапі розробки функції ущільнення. Розпаралелювання може сприяти прискоренню обчислень, але важливо уникати значної втрати стійкості, особливо в умовах загальних атак на геш-функції.

Головна задача полягає у зав’язанні каналів під час розпаралелення, у геш-функціях в яких викоистовується зав’язанні канали, крипто стійкіші та важко зламані.

Метою магістерської кваліфікаційної роботи є пришвидшення формування значень геш-функції, за рахунок розпаралелення обчислень над даними, які представляються у вигляді кватерніонів.

Задачі які потрбіно розвязати:

– проаналізувати відомі підходи до побудови геш-функцій;

– розробити метод гешування з розпаралеленням обчислень і зав’язуванням блоків на основі математичної моделі кватерніонів;

– розробити програмний засіб, що реалізує запропонований метод гешування даних;

– отримати порівняльні оцінки складності обчислення геш-функцій;

– протестувати та дослідити розроблений метод гешування даних.

Об'єкт дослідження – процес гешування даних.

Предмет дослідження – метод та засіб пришвидшеного гешування даних.

Наукова новизна одержаних результатів:

1. Вперше запропоновано метод гешування даних з розпаралеленням обчислень і зав’язуванням блоків, який на відміну від відомих методів, використовує зав’язування блоків на основі математичної моделі кватерніона, що забезпечує пришвидшення процесу гешування.

Практичну цінність складає програмний засіб, що реалізує метод пришвидшенного гешування.

1. АНАЛІЗ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1.1. Геш Функція.

Хешування — це процес створення виходу фіксованого розміру на основі входу змінного розміру за допомогою математичних формул, відомих як хеш-функції, які реалізуються як алгоритми хешування. Криптографічні хеш-функції, які використовуються в криптовалютах, забезпечують високий рівень цілісності та безпеки даних в блокчейнах та інших розподілених системах.

Як детерміновані, звичайні і криптографічні хеш-функції завжди дають одинаковий результат , якщо вхідні дані залишаються незмінними. Ці алгоритми часто розглядають як односторонні, оскільки їх важко скасувати без значного обсягу обчислювального часу і ресурсів. Генерація вихідних даних із входу здійснюється легко, але зворотний процес (генерація входу лише із виходу) є відносно складним. Загалом, чим складніше знайти вхід, тим більш безпечним вважається алгоритм хешування.

Можна для прикладу пропустити значення "Binance" і "binance" через алгоритм хешування SHA-256 (він же і використовується в Bitcoin)[1].

SHA-256	
Вхід	Вихід (256 біт)
Binance	f1624fcc63b615ac0e95daf9ab78434ec2e8ffe402144dc631b055f711225191
binance	59bba357145ca539dcd1ac957abc1ec5833319ddcae7f5e8b5da0c36624784b2

Рисунок 1 - робота sha-256(приклад гешування)

Звичайні хеш-функції знаходять широке застосування у різних випадках, таких як пошук у базах даних, аналіз великих файлів та управління даними. У той час як криптографічні хеш-функції використовуються в програмах

інформаційної безпеки для аутентифікації повідомлень та вилучення цифрових відбитків пальців. В контексті Bitcoin, криптографічні хеш-функції необхідні для майнінгу та генерації адрес та ключів.

Справжня ефективність хешування виявляється при роботі з великим обсягом інформації. Наприклад, можна обробити великий файл чи набір даних через хеш-функцію та використовувати отримані виходи для швидкої перевірки точності та цілісності даних. Детермінований характер хеш-функцій дозволяє отримувати спрощений, стиснений вихід (хеш) при кожному введенні, уникнувши необхідності зберігання великих обсягів даних.

В контексті блокчейн-технології хешування особливо важливе. Блокчейн Bitcoin використовує хешування у багатьох операціях, здійснюваних під час майнінгу. Майже усі крипто-валютні протоколи використовують гешування для скріплення всіх транзакцій у так звані блоки, та для побудови криптографічних зв'язків між усіма блокоми, утворюючи таким чином блокчейн.

1.2. Криптографічні геш-функції.

Крім того, можна визначити хеш-функцію, яка працює за рахунок криптографічних методів, як криптографічну хеш-функцію. Взагалі, зламання криптографічної хеш-функції повинні містити безлічі спроб грубого перебору чисел. Для "розгортання" криптографічної хеш-функції необхідно випробовувати різні вхідні значення методом проб і помилок, доки не буде отримано відповідного результату. Однак існує ймовірність, що різні вхідні дані можуть давати один і той самий результат, і в такому випадку відбудеться "колізія". [18].

Криптографічна хеш-функція повинна відповідати трьом основним властивостям для технічної надійності. Ці властивості можна узагальнити так:

- Стійкість до колізії: Забезпечує неможливість знаходження двох різних входів, що генерують однаковий хеш.

- Стійкість до атаки знаходження першого першовзору: Гарантує відсутність можливості "розгортання" хеш-функції, тобто знаходження входу за заданим вихідом.

- Стійкість до атаки знаходження другого першовзору: Забезпечує відсутність можливості знайти будь-який інший вхід, що має такий самий хеш, як і перший[2].

Деякі поширені криптографічні геш-функції:

- SHA-1 - одна з найпоширеніших криптографічних геш-функцій. Вона має довжину геш-значення 160 біт.

- SHA-2 - сімейство криптографічних геш-функцій, яке включає SHA-224, SHA-256, SHA-384 та SHA-512. Вони мають довжини геш-значення 224, 256, 384 та 512 біт відповідно.

- SHA-3 - криптографічна геш-функція, яка була розроблена в рамках конкурсу NIST. Вона має довжину геш-значення 256 біт.

У сучасних криптографічних геш-функціях використовується комбінація різних методів, щоб забезпечити високу стійкість до колізій. До таких методів належать:

- Лінійні функції. Лінійні функції використовуються для того, щоб ускладнити знаходження колізій шляхом нейтралізації ефекту додавання або віднімання одиничних бітів у геш-значення.

- Нелінійні функції. Нелінійні функції використовуються для того, щоб ускладнити знаходження колізій шляхом створення складних взаємозв'язків між бітами в геш-значенні.

- Ітеративні процеси. Ітеративні процеси використовуються для того, щоб посилити стійкість до колізій шляхом повторного застосування лінійних та нелінійних функцій.

Криптографічні геш-функції є важливим інструментом у криптографії та мають широкий спектр застосувань. Вони продовжують розвиватися, щоб забезпечити високу стійкість до колізій та інших атак.

1.3 Геш в блокчейні.

У традиційній фінансовій системі усі операції підтверджуються уповноваженими працівниками банків. Введення невірних даних в контрольну програму може призвести до здійснення неправомірної транзакції. У випадку блокчейну ситуація інша. Криптовалюта представляє собою електронний запис, в якому послідовно фіксуються всі проведені операції. Будь-яка інформація, незалежно від обсягу, піддавалася шифруванню та кодуванню за допомогою криптографічного методу. Отриманий хеш має фіксовану довжину: 64, 128 або 256 біт. Найдовший криптокод у мережі біткоїн складається з 64 символів і є необхідним для надійності, оскільки його неможливо змінити або вилучити. При кожній новій операції в мережі проводиться валідація, яка перевіряє всі попередні транзакції у блоку. Якщо виявляються порушення, угода вважається недійсною.

Схематично, операції в блокчейні працюють наступним чином:

1. Користувач ініціює процес, відправляючи транзакцію.

2. Операції групуються в блок, який має початковий стан (адресу та час) та передбачуваний кінцевий стан (якщо угода буде схвалена).

3. Система розсилає інформацію про сформований блок всім учасникам (майнерам) для верифікації даних початкового стану.

4. Користувачі, отримавши дані про передбачувану транзакцію, підтверджують їхню правильність та включають у свій ланцюжок. Немає конкретного місця для зберігання блокчейну криптовалюти; впорядковані дані розташовані всюди і доступні кожному учаснику мережі у будь-який момент.

5. Після верифікації блоку майнери підтверджують весь ланцюжок, і транзакція вважається завершеною.

Якщо початковий стан блоку не верифікований, операція скасовується, монети «заморожуються» у сховищі з некоректною ділянкою ланцюга.[2]

У процесі майнінгу використовуються різноманітні етапи, які базуються на використанні хеш-функцій. Ці етапи включають в себе перевірку балансу,

зв'язування входів і виходів транзакцій, а також хешування всіх операцій у блоку для створення дерева Меркла. Важливою причиною, яка робить блокчейн Bitcoin безпечним, є необхідність для майнерів виконувати багато операцій, пов'язаних з хешуванням, для знаходження правильного рішення для наступного блоку [19].

Майнерам необхідно експериментувати з різними вхідними даними при формуванні хешу для свого блоку-кандидата. Перевірити правильність блоку можна лише в тому випадку, якщо згенерований хеш має певну кількість початкових нулів. Рівень складності майнінгу, який змінюється від хешрейту мережі, визначає кількість цих нулів.

Тут хешрейт вказує на потужність комп'ютера, вкладену в майнінг біткоїнів. Зі зростанням хешрейту протокол Bitcoin автоматично регулює складність майнінгу, щоб середній час формування блоку залишався навколо 10 хвилин. Якщо кілька майнерів вирішать припинити майнінг і зменшить хешрейт, то складність майнінгу буде відповідно скоригована для тимчасового полегшення обчислювальної роботи (поки час формування блоку не повернеться до 10 хвилин).

Важливо відзначити, що майнерам не потрібно шукати колізії для певної кількості хешів, які вони можуть генерувати як валідний вихід (з певною кількістю нулів). Таким чином, для конкретного блоку може існувати кілька можливих рішень, і майнерам потрібно знайти лише одне, яке відповідає установленому порогу складності майнінгу. Майнінговий алгоритм, що базується на SHA-256, відомий як "доказ роботи" (Proof of Work). Розглянутий підхід забезпечує встановлення консенсусу та мінімізує ризики атак на блокчейн. Більше того, використання криптографії робить децентралізовані системи більш безпечними порівняно з традиційними банківськими системами, які використовують фізичні методи підтвердження платежів [20].

Оскільки процес майнінгу Bitcoin є витратним завданням, майнери не мають мотивації обманювати систему, оскільки це може веде до серйозних фінансових втрат. З цієї причини чим більше майнерів приєднується до блокчейну, тим відмітнішим і міцнішим він стає.

1.4 Аналіз відомих функцій хешування

Давайте розглянемо деякі відомі криптографічні хеш-функції. Зараз існують різні хеш-функції, які в різній мірі захищені від описаних вище атак. Розглянемо односторонні та безколізійні хеш-функції, які широко використовуються в сучасних криптографічних системах для захисту інформації, зокрема в схемах цифрового підпису. До цих функцій відносяться такі, як SHA-1 та SHA-2 [6].

SHA-1, що є скороченням від "Secure Hash Algorithm", є безпечним хеш-алгоритмом. Він був розроблений Національним інститутом стандартів і технологій (NIST) та опублікований як федеральний інформаційний стандарт (FIPS PUB 180) у 1993 році [2, 11, 4].[5]

Процес виконання алгоритму SHA-1 включає кілька кроків, які можна описати наступним чином:

Крок 1: Доповнення відсутніх бітів. На вхід алгоритм отримує повідомлення максимальної довжини 264 бітів і генерує хеш-код повідомлення, який має довжину 160 бітів. Для досягнення цього результату виконуються наступні операції: Повідомлення доповнюється так, щоб його довжина була кратною 512 за модулем 448 бітів. Доповнення завжди виконується, навіть якщо повідомлення вже має потрібну довжину. Наприклад, якщо довжина повідомлення складає 448 бітів, воно доповнюється 512 бітами до 960 бітів. Кількість доданих бітів може знаходитися в діапазоні від 1 до 512. Доповнення складається з одиниці, за якою слідує необхідна кількість нулів.[10]

Крок 2: Додавання довжини. 64-бітне представлення довжини вихідного повідомлення в бітах приєднується до результату першого кроку. Якщо початкова довжина перевищує 264 біти, то використовуються тільки останні 64 біти. Таким чином, це поле містить довжину вихідного повідомлення за модулем 264. Після виконання перших двох кроків утворюється повідомлення, довжина якого кратна 512 бітам. Це розширене повідомлення представлене як послідовність 512-бітних блоків, при цьому загальна довжина розширеного

повідомлення дорівнює бітам. Таким чином, отримана довжина розширеного повідомлення кратна шістнадцяти 32-бітним словам.

Крок 3: Перше налаштування буфера SHA-1. Для зберігання проміжних та остаточних результатів хеш-функції використовується 160-бітний буфер. Цей буфер може бути представлений чотирма 32-бітними регістрами (A, B, C, D та E), які початково заповнюються наступними значеннями.

Крок 4: Блоками по 512 бітів (16-слівні блоки). Основний модуль алгоритму включає від 80 циклових обрахунків. Усі ці цикли мають однакову структуру і приймають на вхід поточний 512-бітний оброблюваний блок Yq та 160-бітне значення буфера ABCDE, змінюючи вміст цього буфера [6].

Кожен цикл працює за допомогою додоткавих констант які примають 4 повністю різних значень які вказані в табл. 1.1.

Для отримання виходу SHAq+1 в 80-му циклі значення формується на основі SHAq. Плюсуються за модулем 232 виконується не залежачи ні для якого з п'яти слів у буфері, і кожне слово обчислюється як сума відповідного слова у SHAq та відповідного слова у SHAq+1.

Крок 5: По завершенню обробки 512-бітних блоків виходом L стадії є 160-бітний хеш-код.

Таблиця 1.1 - Цикли виконання функції хешування SHA-1.

$0 \le t \le 19$	K_t = 5A827999	(ціла частина числа [$2^{30} \times 2^{½}$])
$20 \le t \le 39$	K_t = 6ED9EBA1	(ціла частина числа [$2^{30} \times 3^{½}$])
$40 \le t \le 59$	K_t = 8F1BBCDC	(ціла частина числа [$2^{30} \times 5^{½}$])
$60 \le t \le 79$	K_t = CA62C1D6	(ціла частина числа [$2^{30} \times 10^{½}$])

Таблиця 1.2 - Виконання функцій в залежності від номеру циклу

Номер циклу	$f_t(B,C,D)$
(0 ≤ t ≤ 19)	$(B \& C)^{\vee} (\bar{B} \& D)$
(20 ≤ t ≤ 39)	$B \oplus C \oplus D$
(40 ≤ t ≤ 59)	$(B \& C)^{\vee} (B \& D)^{\vee} (C \& D)$
(60 ≤ t ≤ 79)	$B \oplus C \oplus D$

Перші 16 значень Wt беруться безпосередньо з 16 слів поточного блоку. Значення, що залишилися, визначаються так:

У перших 16 циклах вхід складається з 32-бітного слова даного блоку. Для 64 циклів вхід формується як XOR декількох слів із блоку повідомлення.

$$Ch(x,y,z) = (x \oplus y) \oplus (\bar{x} \oplus z),$$
$$Maj(x,y,z) = (x \oplus y) \oplus (x \oplus z) \oplus (y \oplus z),$$

$$\sum_0^{(256)}(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x),$$
$$\sum_1^{(256)}(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x),$$

$$\sigma_0^{(256)}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x),$$
$$\sigma_1^{(256)}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x).$$

Перша обробка повідомлення, включаючи додавання конкретних бітів до цілого числа блоків і подальшу його розбивку на блоки, виконується аналогічно, як це робилося в SHA-1, з урахуванням довжини блоку кожної хеш-функції. Після цього кожне повідомлення можна представити у вигляді послідовності N блоків.

У цьому випадку ініціалізуються вісім 32-бітних змінних (a, b, c, d, e, f, g, h), які будуть використовуватися в якості проміжних значень хеш-коду. Основою алгоритму є модуль, який складається з 64 циклічних обробок кожного блоку M(i):

$$T_1 = h + \sum_1^{(256)}(e) + Ch(e, f, g) + K_t^{(256)} + W_t,$$

$$T_2 = \sum_0^{(256)}(a) + Maj(a, b, c),$$

$$h = g, g = f, f = e, e = d + T_1, d = c, c = b, b = a, a = T_1 + T_2.$$

де Ki{256} - шістдесят чотири 32-бітних константи, кожна з яких є першими 32-ма бітами дробової частини кубічних коренів перших 64 простих чисел; Wt обчислюються з чергового блоку повідомлення за наступними правилами:

$$W_t = M_t^{(i)}, 0 \le t \le 15,$$

$$W_t = \sigma_1^{(256)}(W_{t-2}) + W_{t-7} + \sigma_0^{(256)}(W_{t-15}) + W_{t-16}, 16 \le t \le 63.$$

і-те проміжне значення хеш-кода H(t) обчислюється в такий спосіб:

$$H_0^{(i)} = a + H_0^{(i-1)}, \quad H_1^{(i)} = b + H_1^{(i-1)}, \quad H_2^{(i)} = c + H_2^{(i-1)}, \quad H_3^{(i)} = d + H_3^{(i-1)},$$

$$H_4^{(i)} = e + H_4^{(i-1)}, \quad H_5^{(i)} = f + H_5^{(i-1)}, \quad H_6^{(i)} = g + H_6^{(i-1)}, \quad H_7^{(i)} = h + H_7^{(i-1)}.$$

Кінцевий хеш-код формується з лівих 256 бітів остаточного хеш-коду H(n)= H0(n) || H1(n) || H2(n) || H3(n) || H4(n) || H5(n) .

1.5 Сучасні методи пришвидшенного гешування даних.

Дерево Меркла є унікальною структурою даних, спроектованою для компактного утримання інформації про великі обсяги даних та забезпечення перевірки їх цілісності. Процес створення дерева Меркла включає розділення даних на невеликі блоки, які піддаються хешуванню за допомогою Leaf Tiger Hash. Internal Tiger Hash обчислюється з отриманих хешів для кожної пари гешів. У випадку одиночних гешів вони переносяться в новий ланцюжок без змін. Цей процес повторюється, поки не залишиться один геш, який отримує назву Tiger Tree Root. Цей геш служить унікальним ідентифікатором файлу і використовується у різних P2P посиланнях.

Геш-дерева використовуються для перевірки цілісності, достовірності та надійності даних в комп'ютерних системах. Вони забезпечують перевірку

цілісності та достовірності окремих блоків даних, що передаються між вузлами в peer-to-peer мережах. Геш-дерева також застосовуються в системах довіреного завантаження, геш-криптографії та файлових системах, таких як IPFS, Btrfs і ZFS, для запобігання деградації даних. Вони використовуються в різних системах, таких як Dat для розповсюдження даних, Google Wave для протоколів, системи керування версіями Git та Mercurial, резервні системи типу Tahoe-LAFS, мережі Bitcoin та Ethereum, фреймворк прозорості сертифікатів, а також в системах Raik та Dynamo.

Оригінальна реалізація дерева Меркла в Bitcoin від Сатосі Накомато використовує етап стиснення геш-функцій до зайвого рівня, що полегшує використання так званих дерев типу Fast Merkle.

Дерево гешів є бінарною структурою, де листки представляють геш-блоки даних, таких як файли або групи файлів. Верхні вузли дерева є гешами їхніх "дітей". Наприклад, hash 0 представляє собою результат гешування конкатенації hash 0-0 та hash 0-1. Іншими словами, hash 0 = hash (hash 0-0 + hash 0-1), де + вказує на конкатенацію.

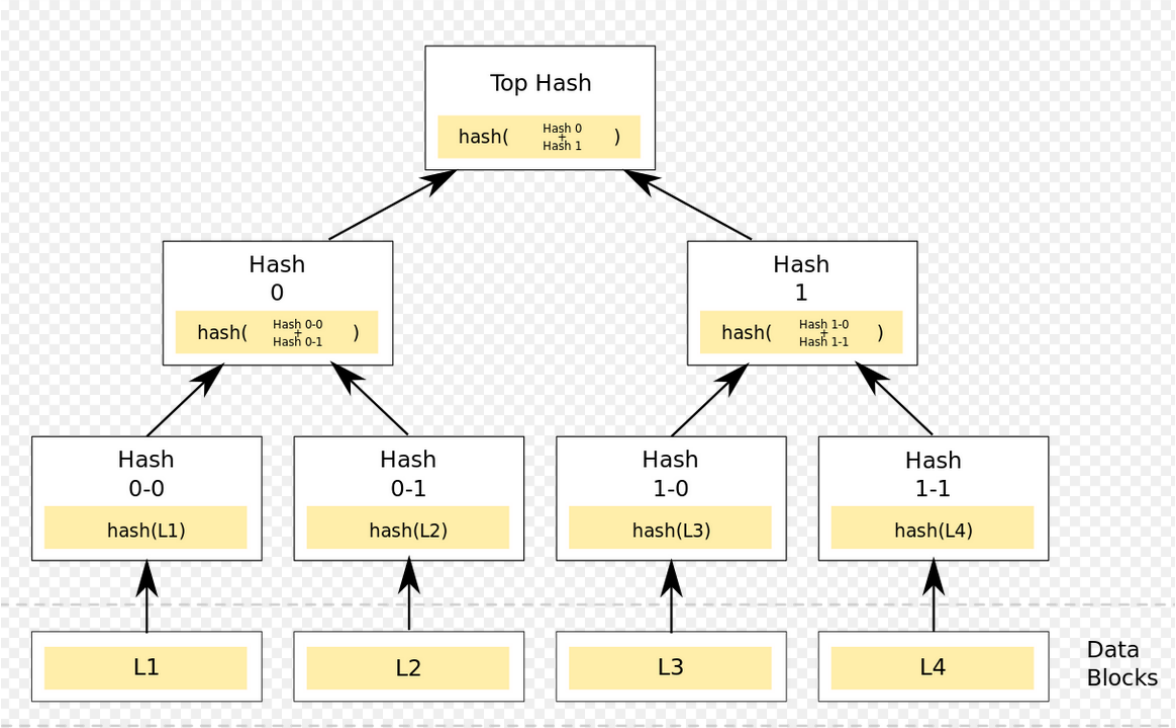


Рисунок 1.2 - Дерево Меркла

Більшість виконань геш-дерев є двійковими, але можуть використовувати і багато інших дочірніх вузлів під кожним вузлом.

Зазвичай для гешування використовують криптографічну геш-функцію, таку як SHA-2.

ASIC-модулі (Application-Specific Integrated Circuit) представляють собою мікросхеми, розроблені спеціально для виконання конкретних завдань чи функцій. У випадку геш-функцій використання ASIC-модулів може значно підвищити ефективність цього процесу.

ASIC-модулі, призначені для геш-функцій, оптимізовані для швидкого виконання обчислень через спеціалізовану апаратуру. Вони можуть забезпечити значну швидкість обчислень, що стає критичним для деяких застосувань, зокрема майнінгу криптовалют.

Хоча використання ASIC-модулів може покращити продуктивність та ефективність геш-обчислень, слід враховувати витрати та обмеженість їх функціональності порівняно із загальнопризначеними пристроями.

Паралелізація обчислень геш-функцій — це метод для прискорення геш-обчислень та підвищення продуктивності обробки даних. Цей підхід використовує розділення геш-обчислень на незалежні процеси, що виконуються одночасно. Паралелізація може бути доцільною в сферах, де важлива швидкість обробки даних в режимі реального часу. Для цього використовуються різні методи, такі як розподілені системи, масштабовані сервери, кластери, графічні процесори (GPU) та інші. Наприклад, кластер з багатьма вузлами може обчислювати хеші для окремих транзакцій одночасно, допомагаючи великим обсягам даних.

Узагальнюючи, використання ASIC-модулів та паралелізації є ефективними методами для оптимізації геш-обчислень в різних сценаріях. Однак їх використання повинно бути узгодженим з конкретними вимогами та обмеженнями конкретного застосування.

Однак, при використанні паралельних методів гешування транзакцій важливо враховувати питання синхронізації та консистентності даних. Різні

транзакції можуть залежати одна від одної або мати взаємозалежність, тому необхідно правильно керувати порядком виконання обчислень та забезпечити цілісність даних[8].

Ураховуючи всі аспекти, використання паралелізації гешування транзакцій може значно прискорити обробку даних і підвищити продуктивність системи. Цей підхід дозволяє ефективно використовувати наявні ресурси та забезпечує швидку реакцію на запити гешування. Однак важливо враховувати питання синхронізації даних та ретельно вибирати методи та технології для досягнення оптимального рішення відповідно до конкретних вимог та обмежень системи.

Кожен метод має свої переваги та недоліки, і вибір залежить від конкретних умов та вимог системи. Наприклад, для блокчейн систем може бути виправданим використання методу Дерева Меркла, тоді як у корпоративних мережах може бути більш доцільним застосування ASIC-модулів. Важливо ретельно оцінювати, який метод буде на найефективнішим для конкретного випадку.

Виходячи з аналізу джерел буде пропонуватись метод гешування на основі кватерніонів.

Метод пришвидшеного гешування на основі кватерніонів є захопливим напрямком досліджень в області обчислень і криптографії. Кватерніони, як гіперкомплексні числа, здатні представляти обертання в тривимірному просторі. Використання їх у геш-функціях може допомогти покращити швидкість обчислень і збільшити безпеку.

Однією з переваг кватерніонів є їхні властивості при обертанні. Математичні операції, пов'язані з кватерніонами, можуть бути ефективно використані для створення геш-функцій, що використовують обертання як основний елемент. Це може призвести до швидших обчислень порівняно з традиційними методами гешування.

Крім того, застосування кватерніонів може забезпечити додатковий рівень безпеки в геш-функціях. Їхні математичні властивості дозволяють створювати

складні та унікальні геш-коди, що ускладнює завдання зловмисникам зламати генеровані значення.

Однак важливо враховувати, що впровадження таких методів вимагає детального аналізу їхньої ефективності та безпеки в конкретних застосуваннях. Також слід враховувати ресурси, необхідні для реалізації цих методів в реальних системах, і можливі обмеження, пов'язані з їхнім застосуванням у певних областях.

Виходячи з аналізу джерел буде пропонуватись метод гешування на основі кватерніонів.

Метод пришвидшеного гешування на основі кватерніонів є захопливим напрямком досліджень в області обчислень і криптографії. Кватерніони, як гіперкомплексні числа, здатні представляти обертання в тривимірному просторі. Використання їх у геш-функціях може допомогти покращити швидкість обчислень і збільшити безпеку.

Однією з переваг кватерніонів є їхні властивості при обертанні. Математичні операції, пов'язані з кватерніонами, можуть бути ефективно використані для створення геш-функцій, що використовують обертання як основний елемент. Це може призвести до швидших обчислень порівняно з традиційними методами гешування.

Крім того, застосування кватерніонів може забезпечити додатковий рівень безпеки в геш-функціях. Їхні математичні властивості дозволяють створювати складні та унікальні геш-коди, що ускладнює завдання зловмисникам зламати генеровані значення.

2. РОЗРОБКА МЕТОДУ ПРИШВИДШЕНОГО ГЕШУВАННЯ

2.1 Кватерніони.

Кватерніон — гіперкомплексне число, яке реалізується в 4-вимірному просторі. Вперше описане В.Р. Гамільтоном у 1843 році [21].

Кватерніони застосовуються як у теоретичній, так і в прикладній математиці, зокрема для виконання обчислень поворотів у тривимірній графіці та області машинного зору.[5]

Кватерніони мають вигляд:

$$q = a + bi + cj + dk\,, \tag{2.1}$$

де a, b, c, d — цілі числа;

i, j, k — одиниці, які задовільняють співвідношенню:

$$i^2 = j^2 = k^2 = i \cdot j \cdot k = -1, \tag{2.2}$$

За стандартом записуються *i1, i2, i3* замість *i, j*, k для уявних одиниць, при цьому приймають i1:=1. Також не стандартний варіант запису - e0, e1, e2, e3. Для кватерніона $q=a+bi+cj+dk$, число a називається скалярною частиною, а вектор $\vec{\upsilon}=bi+cj+dk$ – його векторною частиною. [21].

Припустим $\vec{\upsilon} = 0$, тоді кватерніон буде названим чисто скалярним, при $a = 0$ – векторним.

Кватерніон $\bar{q} = a - bi - cj - dk$ називається спряженим до q.

$$\overline{q_1 q_2} = \bar{q_1}\,\bar{q_2}, \tag{2.3}$$

Аналогічно як для комплексних чисел, норма кватерніона буде визначатись так:[5].

$$|q| = \sqrt{q\bar{q}} = \sqrt{a^2 + b^2 + c^2 + d^2}, \tag{2.4}$$

$$q^{-1} = \frac{\bar{q}}{|q|}, \tag{2.5}$$

Припустим $|q| = 1$ то q буде тоді одиничним кватерніоном. Легко перевірити, що:

$$|p \cdot q| = |p| \cdot |q|, \quad (2.6)$$

Отже, кватерніони обладнані мультиплікативною нормою, що визначається тотожністю чотирьох квадратів [5]. Їх також можна визначити за допомогою комплексних чисел за допомогою процедури, відомої як проведення Келі-Діксона. Ця рекурсивна процедура конструює алгебри над полем дійсних чисел, подвоюючи їх розмірність на кожному етапі. Таким чином, вона дозволяє визначити комплексні числа, кватерніони та інші математичні структури [5].

Довільний кватерніон $q = a + bi + cj + dk$ можна представити у вигляді:

$$q = (a + bi) + (c + di)j, \quad (2.7)$$

або

$$\begin{Bmatrix} q = z_1 + z_2 j \\ z_1 = a + b_i \\ z_2 = c + c_i \end{Bmatrix}, \quad (2.8)$$

де z_1, z_2 – комплексні числа, оскільки $i^2 = -1$ виконується як для комплексних чисел так і для кватерніонів [23].

Кватерніон $q = a + bi + cj + dk$ можна представити у вигляді пари скаляра та 3-вимірного вектора [5]:

$$q = (s, \vec{v}),\ s = a,\ \vec{v} = (b, c, d), \quad (2.9)$$

Виявляється, що множення кватерніонів можна записати через скалярний та векторний добутки відповідних 3-вимірних векторів [21]:

$$q_1 q_2 = (s_1, \vec{v}_1)(s_2, \vec{v}_2) = (s_1 s_2 - \vec{v}_1 \cdot \vec{v}_2, s_1 \vec{v}_2 + s_2 \vec{v}_1 + \vec{v}_1 \times \vec{v}_2), \quad (2.10)$$

В даному підході чисто векторні кватерніони можна розглядати як еквівалентні 3-вимірним векторам. У такому випадку множення двох таких кватерніонів можна отримати шляхом віднімання скалярного добутку від їх векторного добутку. [5]:

$$(0, \vec{v}_1)(0, \vec{v}_2) = (-\vec{v}_1 \cdot \vec{v}_2, \vec{v}_1 \times \vec{v}_2), \quad (2.11)$$

Кватерніон може бути представлений у вигляді матриці 2×2 із комплексних чисел [5]:

$$\begin{pmatrix} a + bi & c + di \\ -c + di & a - bi \end{pmatrix}, \quad (2.12)$$

Запишемо кватерніони, як $q = a_1 + b_1 i + c_1 j + d_1 k$ і $p = a_2 + b_2 i + c_2 j +$

d_2k, і розглянемо загальні операції над ними.

Додавання кватерніонів, що позначається знаком «+», визначається формулою [5]:

$$q + p = \begin{bmatrix} a_1 & + & a_2 \\ ib_1 & + & ib_2 \\ jc_1 & + & jc_2 \\ kd_1 & + & kd_2 \end{bmatrix}, \tag{2.13}$$

Кватерніони складаються покомпонентно. В результаті складання двох кватерніонів виходить новий кватерніон. Додавання кватерніонів коммутативно [5].

Аналогічно визначається і віднімання кватерніонів [20]:

$$q - p = \begin{bmatrix} a_1 & - & a_2 \\ ib_1 & - & ib_2 \\ jc_1 & - & jc_2 \\ kd_1 & - & kd_2 \end{bmatrix}, \tag{2.14}$$

Для кватерніонів справедливі такі рівності [20]:

$$\begin{cases} \overline{p+q} = \bar{p} + \bar{q}; \\ \overline{p-q} = \bar{p} - \bar{q}. \end{cases}, \tag{2.15}$$

При множенні кватерніона на число, буде визначатись формулою:

$$a \cdot q = q \cdot a = \begin{bmatrix} a & \cdot & a_1 \\ a & \cdot & ib_1 \\ a & \cdot & jc_1 \\ a & \cdot & kd_1 \end{bmatrix}, \tag{2.16}$$

У результаті множення числа на кватерніон або кватерніона на число виходить новий кватерніон. Розглянута операція є коммутативною [20].

При множенні двох кватерніонів враховуються такі правила множення їх елементів [25]:

$i^2 = j^2 = k^2 = -1$;

$i \cdot j = k$; $j \cdot i = -k$;

$i \cdot k = j$; $k \cdot i = -j$;

$j \cdot k = i$; $k \cdot j = -i$.

$$\begin{Bmatrix} i^2 = j^2 = k^2 = -1; \\ i \cdot j = k; j \cdot i = -k; \\ i \cdot k = j; k \cdot i = -j; \\ j \cdot k = i; k \cdot j = -i. \end{Bmatrix}, \tag{2.17}$$

Множення кватерніонів визначається формулою:

$$q \cdot p = \begin{bmatrix} (a_1 a_2 - b_1 b_2 - c_1 c_2 - d_1 d_2) \\ i(a_1 b_2 + b_1 a_2 + c_1 d_2 - d_1 c_2) \\ j(a_1 c_2 + c_1 a_2 + b_1 d_2 - d_1 b_2) \\ k(a_1 d_2 + d_1 a_2 + b_1 c_2 - c_1 b_2) \end{bmatrix}, \tag{2.18}$$

Результатом множення двох кватерніонів є новий кватерніон. Потрібно відзначити, що множення кватерніонів є не комутативним, що означає, що його результат узагальнено залежить від порядку співмножників [25]. Давайте розглянемо деякі характеристики цього множення, базуючись на властивостях уявних одиниць, які були визначені вище.

— додавання кватеріонів є асоціативним та комутативним;

— множення кватерніонів є асоціативним, але не є комутативним.

Множення кватерніонів асоціативно [20]:

$$q \cdot p \cdot r = (q \cdot p) \cdot r = q \cdot (p \cdot r), \tag{2.19}$$

Множення кватерніонів дистрибутивно по відношенню до підсумовування [25]:

$$\begin{Bmatrix} q \cdot (p + r) = q \cdot p + q \cdot r; \\ (p + r) \cdot q = p \cdot q + r \cdot q; \end{Bmatrix}, \tag{2.20}$$

Оскільки множення в системі кватерніонів є некомутативним, це призводить до висновку, що ця система не є полем, але є тілом і, отже, не має дільників нуля. Зазвичай тіло кватерніонів позначається як H. Це вказує на можливість ділення в системі кватерніонів, проте важливо розрізняти між лівим та правим діленням [5]. Таким чином, ми виклали загальні властивості та характеристики кватерніонів, які є основою для опису запропонованого методу, і також проаналізували математичні операції, які можна виконувати над ними.

2.2 Метод гешування на основі моделі кватерніона.

Основна концепція запропонованого методу гешування даних полягає в тому, що міжнародні значення hi-1 та значення блоків даних mi, які мають розрядність n, розділяються на чотири частини і виражаються у формі кватерніонів. Отже, подання проміжного значення hi-1 матиме такий вигляд:

$$h_{i-1} = a_1 + b_1 i + c_1 j + d_1 k, \tag{2.21}$$

Аналогічний вигляд буде мати представлення блоку даних m_i:

$$m_i = a_2 + b_2 i + c_2 j + d_2 k, \tag{2.22}$$

Як було показано, що операція множення може забезпечити однаковий вплив кожного біта на кінцеве геш-значення. Тому раціональним є використання цієї операції для впровадження результатів дій над блоками даних у геш-значення, отримане на попередньому етапі. З цим урахуванням доведеться отримати геш-функції як добуток вхідних елементів за модулем 2n. [9].

$$h_i = (h_{i-1} \cdot m_i) mod 2^n, \tag{2.23}$$

Результат множення h_{i-1} на m_i буде мати такий вигляд:

$$\begin{gathered} \left((a_1 + b_1 i + c_1 j + d_1 k) \cdot (a_2 + b_2 i + c_2 j + d_2 k)\right) = \\ = (a_1 a_2 - b_1 b_2 - c_1 c_2 - d_1 d_2) \\ +i(a_1 b_2 + b_1 a_2 + c_1 d_2 - d_1 c_2) \\ +j(a_1 c_2 + c_1 a_2 + b_1 d_2 - d_1 b_2) \\ +k(a_1 d_2 + d_1 a_2 + b_1 c_2 - c_1 b_2) \end{gathered} , \tag{2.24}$$

Всі добутки має розрядність $2n$. Нехай p_1 і p_2 числа що відповідають молодшим n-розрядам і старшим n-розрядам отриманого добутку. Тоді функцію ущільнення усіх добутків можна представити як:

$$(p_1 + p_2) mod 2^n, \tag{2.25}$$

Отримавши результати, робимо висновок, що множення кватерніонів з ущільненням представиться у вигляді:

$$\begin{aligned}\big((a_1 + b_1 i + c_1 j + d_1 k) \cdot (a_2 + b_2 i + c_2 j + d_2 k)\big)\text{ущ.} = \\ = \big(P_{a_1 a_2} - P_{b_1 b_2} - P_{c_1 c_2} - P_{d_1 d_2}\big) mod 2^n \\ + i\big(P_{a_1 b_2} + P_{b_1 a_2} + P_{c_1 d_2} - P_{d_1 c_2}\big) mod 2^n \\ + j\big(P_{a_1 c_2} + P_{c_1 a_2} + P_{b_1 d_2} - P_{d_1 b_2}\big) mod 2^n \\ + k\big(P_{a_1 d_2} + P_{d_1 a_2} + P_{b_1 c_2} - P_{c_1 b_2}\big) mod 2^n\end{aligned} \quad (2.26)$$

де

$$P_{a_1 a_2} = (p_{1,a_1 a_2} + p_{2,a_1 a_2}) mod 2^n, P_{b_1 b_2} = (p_{1,b_1 b_2} + p_{2,b_1 b_2}) mod 2^n,$$

$$P_{c_1 c_2} = (p_{1,c_1 c_2} + p_{2,c_1 c_2}) mod 2^n, P_{d_1 d_2} = (p_{1,d_1 d_2} + p_{2,d_1 d_2}) mod 2^n,$$

$$P_{a_1 b_2} = (p_{1,a_1 b_2} + p_{2,a_1 b_2}) mod 2^n, P_{b_1 a_2} = (p_{1,b_1 a_2} + p_{2,b_1 a_2}) mod 2^n,$$

$$P_{c_1 d_2} = (p_{1,c_1 d_2} + p_{2,c_1 d_2}) mod 2^n, P_{d_1 c_2} = (p_{1,d_1 c_2} + p_{2,d_1 c_2}) mod 2^n,$$

$$P_{a_1 c_2} = (p_{1,a_1 c_2} + p_{2,a_1 c_2}) mod 2^n, P_{c_1 a_2} = (p_{1,c_1 a_2} + p_{2,c_1 a_2}) mod 2^n,$$

$$P_{b_1 d_2} = (p_{1,b_1 d_2} + p_{2,b_1 d_2}) mod 2^n, P_{d_1 b_2} = (p_{1,d_1 b_2} + p_{2,d_1 b_2}) mod 2^n,$$

$$P_{a_1 d_2} = (p_{1,a_1 d_2} + p_{2,a_1 d_2}) mod 2^n, P_{d_1 a_2} = (p_{1,d_1 a_2} + p_{2,d_1 a_2}) mod 2^n,$$

$$P_{b_1 c_2} = (p_{1,b_1 c_2} + p_{2,b_1 c_2}) mod 2^n, P_{c_1 b_2} = (p_{1,c_1 b_2} + p_{2,c_1 b_2}) mod 2^n,$$

Схема подає одну ітерацію алгоритму гешування даних, базованого на моделі кватерніонів, яку можна оглянути на рис. 2.1. На цьому зображенні використані наступні позначення:

- K – ключ (початкове значення);
- mi – вхідний блок даних, який підлягає гешуванню;
- Бл01–Бл04 – блоки, де відбуваються математичні операції з вхідними даними та геш-значення яке було отримане.

Аналіз схеми вказує на можливість ефективного розпаралелення обчислень, що призводить до прискорення методу гешування даних відносно з послідовною процедурою. Крім того, з схеми видно, як відбувається зв'язування даних, також інша частина вхідних даних має рівномірний вплив на отриманий результат. Висновком можна зробити, що навіть невелика зміна частини вхідного повідомлення значно вплине на кінцеве геш-значення, що є ключовим для алгоритмів гешування. Алгоритм обчислення геш-значення, що

відповідає цій схемі, представлено на рис. 2.2. Таким чином, описано одну з розроблюваних моделей паралельного гешування даних. Подані формули, за якими оброблюються дані та виконуються обчислення геш-значень, а також показано схему одного циклу в алгоритмі гешування.

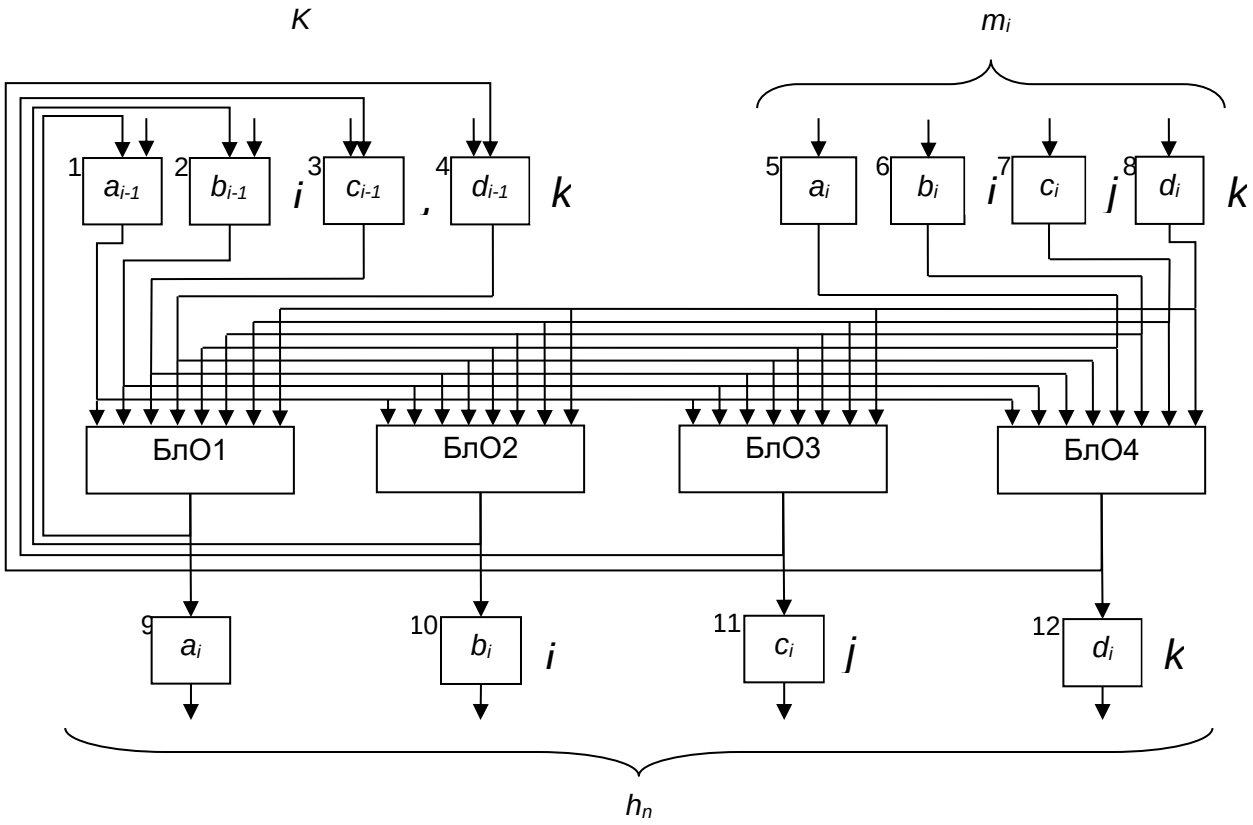


Рисунок 2.1 – Схема однієї ітерації алгоритму гешування.

2.3 Порівняльна оцінка існуючих методів гешування.

Складність запропонованих алгоритмів гешування даних буде оцінюватися на фоні найбільш відомих та широко застосовуваних криптографічних функцій для прискореного гешування даних.

Skein - це сімейство геш-алгоритмів, розроблених Національною лабораторією Oak Ridge (ORNL) з метою забезпечення ефективності, безпеки та масштабованості. Застосовуються в різноманітних галузях, таких як блокчейн, електронна ідентифікація та криптографія. Алгоритми Skein знаходять використання у різних контекстах, включаючи блокчейн, де вони використовуються, зокрема, в Bitcoin і Ethereum. Для електронної ідентифікації

алгоритми Skein використовуються для створення електронних підписів і сертифікатів. Криптографічні аспекти алгоритмів Skein включають застосування генерації випадкових чисел.

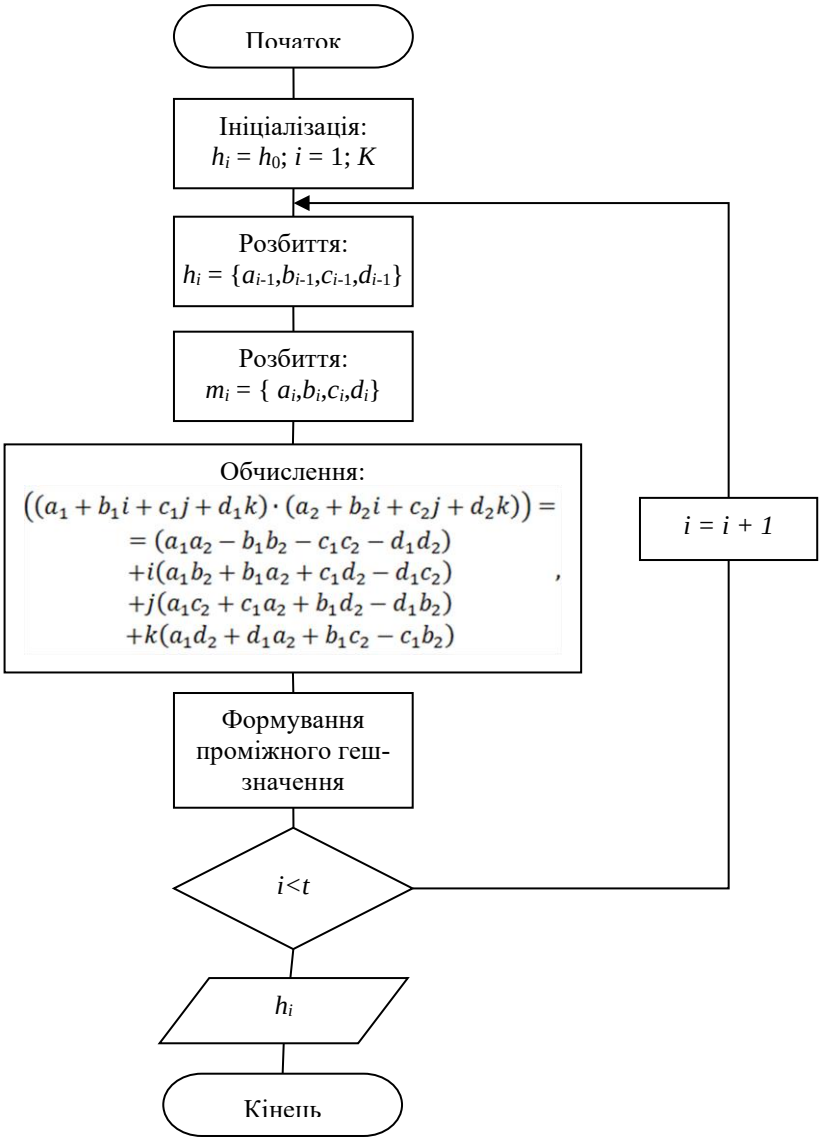


Рисунок 2.2 – Алгоритм обчислення геш-значення за моделлю кватерніона

Keccak є сімейством алгоритмів гешування, розроблених групою авторів під керівництвом Йоана Даймена, який є співавтором Rijndael і автором таких шифрів, як MMB, SHARK, Noekeon, SQUARE і BaseKing. Метою розробки алгоритмів Keccak було створення ефективних, безпечних та масштабованих рішень. Застосовуються в різних областях, таких як блокчейн, електронна ідентифікація та криптографія.

Таблиця 2.1 – Властивості алгоритму Skein

Алгоритм	Довжина дайджесту повідомлення (біт)	Довжина блоку (біт)	Довжина пові-домлення (біт)	Довжина слова (біт)	Кількість ітерацій в циклі
Skein	256	256	$<2^{64}$	64	243

Алгоритми Keccak отримали широке використання у різних галузях, включаючи блокчейн, де їх використовують у сучасних системах, таких як Ethereum, Zcash і Tezos. У сфері електронної ідентифікації алгоритми Keccak використовуються для створення електронних підписів і сертифікатів, а в криптографії генерації випадкових чисел (див.рис.2.3)

Для порівняння швидкодії перерахованих функцій гешування використовується аналіз кількості циклів, яку представлені виконують при обчисленні геш-значення на один байт вхідних даних. Результати порівняння швидкодії представлених функцій гешування представлені в таблиці 2.1.

Таблиця 2.2 – Властивості алгоритму Keccak

Алгоритм	Довжина дайджесту повідомлення (біт)	Довжина блоку (біт)	Довжина пові-домлення (біт)	Довжина слова (біт)	Кількість ітерацій в циклі
Keccak	256	256	$<2^{64}$	64	320

Таблиця 2.3 – Складність алгоритмів гешування

№	Алгоритм	Циклів/байт
1.	Skein	6.1
2.	Keccek	10

Розглядані функції можуть створювати геш-значення різної довжини, починаючи як 256 біт, так і закінчуючи 512 біт. Таблиця 2.3 порівнює складність цих алгоритмів за кількістю циклів, та ітерацій в циклі які виконуються при обчисленні геш-значення на один байт вхідних даних для різних розрядностей коду (64-бітний код та 32-бітний код) та різних розмірів вихідних геш-значень.

Таблиця 2.4 – Порівняльна таблиця складності алгоритмів гешування

Keccak	Skein
64-бітний код, 256-бітний геш, цикли/байт	
10	7.6
64-бітний код, 512-бітний геш, цикли/байт	
20	6.1
32-бітний код, 256-бітний геш, цикли/байт	
31	21.6
32-бітний код, 512-бітний геш, цикли/байт	
62	20.1

Основні відмінності між Skein і Keccak полягають у ряді параметрів. Наприклад, довжина блоку Skein складає 64 біти, тоді як у Keccak вона становить 512 біт. Довжина слова Skein дорівнює 64 бітам, у той час як в Keccak вона також складає 64 біти. Кількість ітерацій в циклі у Skein може бути 16, 32 або 64, в той час як у Keccak завжди вона дорівнює 24. Обидва алгоритми, Skein і Keccak, є ефективними та безпечними і знаходять застосування в різних сферах. Хоча вони мають спільні риси, вони також мають ключові відмінності [11].

Після порівняння складності представлених та існуючих алгоритмів встановлено, що швидкість обчислення геш-значень залежить від конкретного алгоритму, його розрядності та розрядності коду. Представлені алгоритми гешування проявляють швидкість в діапазоні від 6.1 циклів/байт для 64-розрядного варіанту алгоритму Skein до 62 циклів/байт для 256-бітного геш-значення алгоритму Keccak у 64-розрядному варіанті.

2.4 Аналіз та порівняння розробленого методу пришвидшеного гешуваня.

Розроблювані алгоритми гешування можуть бути застосовані для отримання геш-значення різної довжини. Для того щоб провести аналіз з існуючими алгоритмами, розглянемо реалізацію 256-бітного геш-значення в 64-розрядному варіанті коду.

У випадку реалізації алгоритму гешування, що базується на моделі кватерніонів для 256-бітного геш-значення, функції оброблятимуть 4 канали вхідних даних, кожний розрядністю 64 біти. Обробка вхідного значення розміром 32 байти виконуватиметься за один цикл. Для обчислення одного проміжного геш-значення за запропонованим алгоритмом потрібно виконати 16 операцій множення і 28 операцій додавання на кожній ітерації. Загальні характеристики алгоритмів наведено в таблиці 2.5.[5].

Отже, для обчислення геш-значення на один байт вхідних даних розроблюваний алгоритм виконує 44/32 = 1.38 операцій, що є показником ефективності геш-функції на основі моделі кватерніона.

Таблиця 2.5 – Характеристика розроблюваного алгоритму

Алгоритм	Довжина дайджесту повідомлення (біт)	Довжина блоку (біт)	Довжина пові-домлення (біт)	Довжина слова (біт)	Кількість операцій в ітерації
На основі моделі кватерніонів	256	256	$<2^{64}$	64	44

На відміну від цього, один з провідних учасників конкурсу за алгоритмами гешування, таких як Keccak і Skein, вивів результат на рівень 6.1 циклів на байт у 64-розрядному варіанті коду для гешування довжиною 512 біт.

Для додаткової оцінки можна обернутися до відомих криптографічно стійких геш-функцій, які використовують операції піднесення до ступеня. Якщо реалізувати таку функцію з використанням 4 або 8 каналів розрядності n, то в середньому буде потрібно виконати 1,5n операцій множення на кожній ітерації для одного каналу, або відповідно 6n операцій множення для 4 каналів і 12n операцій множення для 8 каналів. У випадку обчислення геш-значення за алгоритмом на основі моделі кватерніонів, потрібно виконати 16 операцій множення і 28 операцій додавання на кожній ітерації.

Можна відмітити, що при розрядності геш-значення 256 біт, тобто n=64, для відомого випадку буде необхідно 384 операції на кожній ітерації, що призводить до прискорення більше ніж у 8 разів для моделі кватерніонів.

У подальшого порівняння розроблюваних алгоритмів з вже існуючими, розглянемо найшвидші криптографічні функції гешування з довжиною гешу 256 біт. Відповідна порівняльна характеристика швидкодії алгоритмів подана в таблиці 2.6 [21].

Таблиця 2.6 – Порівняльні характеристики алгоритмів

№	Алгоритм	Довжина геш-значення (біт)	Операцій на байт
1.	Розроблюваний алгоритм (кватерніони)	256	44
2.	Skein	256	243
3.	Keccak	256	320

Згідно з таблицею, розроблений алгоритм пришвидшеного гешування даних виявляє виразну перевагу у швидкості в порівнянні з представленими алгоритмами, так як вимагає менше кількість операцій на один байт вхідного повідомлення.

У даному розділі було розглянуто кватерніони, їх властивості та основні операції над ними, які будуть використані для представлення та обробки вхідних

даних у розроблюваному алгоритмі гешування. Крім того, детально описано методи гешування даних, подані формули та схеми для алгоритмів гешування. Також проведено порівняння існуючих функцій гешування та здійснена оцінка швидкодії розроблюваних алгоритмів.

3. РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ ДЛЯ ГЕШУВАННЯ

3.1 Формулювання основних вимог до програмного засобу

Наступним етапом є розробка програмного забезпечення, яке реалізує розроблений алгоритм прискореного гешування даних, базуючись на моделі кватерніонів. Розроблене програмне забезпечення має бути створене у формі додатку з імплементованим зрозумілим інтерфейсом, призначеним для забезпечення зручної взаємодії та візуального представлення результатів. Воно повинно включати всі необхідні функції та можливості для повного виконання основного алгоритму програми.

Програмний засіб має виконувати таки вимоги:

- обсяг даних для гешування не більше – 2^{64} байт;
- можливість опрацювання всіх видів файлів;
- розрядність вихідного геш-значення – 256, 512 біт;
- розмір даних які буде обробляти один цикл – 32, 64 байти;
- довжина слова – 64 біт;
- метод пришвидшення – розпаралелення із зав’язуванням блоків.

Вона повинна задовільняти умови і функціональність та була зручною у використанні, вона повинна виконувати ряд ключових функцій. Показання найбільш зрозумілої короткої інформації про розроблену програму, роботу з файлами різних форматів, інтуітивно зрозумілий пошук необхідного файлу та даних, обчислення геш-значень для окремих файлів, вивід необхідної інформації про файли та проміжні результати роботи програми, а також можливість зміни шляху для збереження контрольної інформації [12].

У програмі слід передбачити обробку всіх можливих помилок та недопустимих дій, таких як введення некоректних даних, неможливість відкриття файлів, з якими працює програма, неправильно вказаний шлях для

збереження файлу та інші. При будь-якій помилці на екрані повинно з’являтися відповідне повідомлення.

З урахуванням усіх цих вимог можна забезпечити повне та нормальне функціонування програми:

а) розробити схему функціонування програми;

б) інтерфейс програми;

в) втілити розроблений алгоритми;

г) тестування програму;

3.2 Обґрунтування вибору програмних засобів

Реалізація поставленої задачі, а саме розробка програмного засобу гешування даних, буде проводитися за допомогою мови програмування JS та Solidity та TypeScrypt для тестування коду.

Solidity - це статично типізована, контрактно-орієнтована мова програмування, створена для розробки розумних контрактів, які працюють на віртуальній машині Ethereum (EVM). Програми на мові Solidity транслюються в байткод EVM.

Solidity - це статично типізована, контрактно-орієнтована мова програмування, створена для розробки розумних контрактів, які працюють на віртуальній машині Ethereum (EVM). Програми на мові Solidity транслюються в байткод EVM.

Solidity була розроблена Гавіном Вудом та його командою в 2014 році. Мова була названа на честь твердого (англ. solid) і чесного (англ. honest) контрактів, які вона використовується для створення.

Solidity використовує синтаксис, схожий на JavaScript, що робить її відносно легкою для вивчення для розробників, які вже знайомі з JavaScript. Мова також підтримує ряд криптографічних функцій, які необхідні для розробки розумних контрактів.

Розумні контракти - це програмні контракти, які виконуються автоматично

без необхідності втручання людини. Вони часто використовуються для зберігання активів, таких як криптовалюта, і для виконання угод між сторонами.

Solidity широко використовується для розробки розумних контрактів на платформі Ethereum. Мова також підтримується іншими блокчейн-платформами, такими як EOS і Tron. Основні особливості мови Solidity:

Статична типізація даних визначаються явно, що допомагає уникнути помилок під час виконання.

Мова **контрактно-орієнтована** підтримує контракти, які є самостійними і самодостатніми програмами.

Мова підтримує **Криптографічні функції,** ряд криптографічних функцій, таких як хешування, шифрування та дешифрування.

Solidity - це потужна мова програмування, яка може використовуватися для розробки складних розумних контрактів. Мова є відносно легкою для вивчення і підтримується широким спектром блокчейн-платформ.

Створення криптовалют Solidity може використовуватися для створення криптовалют, таких як Ethereum і Bitcoin.

Розробка децентралізованих додатків (DApps) Solidity може використовуватися для розробки децентралізованих додатків, таких як торгові платформи і фінансові послуги.

Створення смарт-контрактів Solidity може використовуватися для створення смарт-контрактів, які зберігають активи і виконують угоди між сторонами.

Js - це динамічна, об'єктно-орієнтована прототипна мова програмування. Вона є однією з найпопулярніших мов програмування у світі і використовується для розробки веб-додатків, мобільних додатків, десктопних додатків, а також для створення інтерактивного контенту.

JavaScript була розроблена Бренданом Айхом в 1995 році для використання в браузерах. Мова була спочатку призначена для того, щоб додавати інтерактивність до веб-сторінок, але з часом вона стала потужною мовою програмування, яка може використовуватися для створення складних програм.

JavaScript використовує синтаксис, схожий на синтаксис C++, що робить її відносно легкою для вивчення для розробників, які вже знайомі з C++. Мова також підтримує ряд об'єктно-орієнтованих функцій, таких як спадкування, поліморфізм та інкапсуляція.

Особливостями мови JavaScript можуть бути такі речі як. Динамічна типізація типів даних визначаються неявно, що дозволяє використовувати дані різних типів в одному контексті.

Мова об'єктно-орієнтована підтримує об'єктно-орієнтований підхід до програмування.

Прототипна мова використовує прототипи для успадкування.

Скриптова мова призначена для використання в сценаріях.

JavaScript широко використовується для розробки веб-додатків. Мова використовується для створення інтерактивних елементів на веб-сторінках, таких як кнопки, меню, вікна тощо. JavaScript також використовується для анімації, створення ігор та інших інтерактивних ефектів.

JavaScript також використовується для розробки мобільних додатків. Мова використовується для створення інтерфейсів користувача (UI) для мобільних додатків, а також для додавання інтерактивних елементів до мобільних додатків.

JavaScript також використовується для розробки десктопних додатків. Мова використовується для створення інтерфейсів користувача (UI) для десктопних додатків, а також для додавання інтерактивних елементів до десктопних додатків.

JavaScript також використовується для створення інтерактивного контенту. Мова використовується для створення анімацій, ігор, інтерактивних додатків тощо.

Розробка веб-додатків JavaScript використовується для створення інтерактивних елементів на веб-сторінках, таких як кнопки, меню, вікна тощо. JavaScript також використовується для анімації, створення ігор та інших інтерактивних ефектів.

Розробка мобільних додатків JavaScript використовується для створення

інтерфейсів користувача (UI) для мобільних додатків, а також для додавання інтерактивних елементів до мобільних додатків.

Розробка десктопних додатків JavaScript використовується для створення інтерфейсів користувача (UI) для десктопних додатків, а також для додавання інтерактивних елементів до десктопних додатків.

Створення інтерактивного контенту JavaScript використовується для створення анімацій, ігор, інтерактивних додатків тощо [13].

3.3 Розробка схем функціонування програми

Перед безпосередньою реалізацією програмного засобу потрібно розробити схему його функціонування.

Реалізація програмного засобу буде виконуватися в кілька етапів:

1) Вибір та відкриття необхідного файлу.
2) Визначення розміру файлу.
3) Виклик вибраної функції гешування.
4) Отримання та представлення результуючого геш-значення.
5) Виведення інформації про час і швидкість обчислення геш-значення вибраного файлу.
6) Збереження результату в файл.
7) Закриття файлу.

У створеній програмі обчислення геш-значення вхідних даних виконується за певною послідовністю, яку можна знайти на схемі функціонування програми (рисунок 3.1). Схеми функцій паралельного гешування даних, розроблених на основі математичної моделі кватерніонів, наведено на рисунках 3.2 та 3.3 відповідно [14].

Під час виконання своєї роботи розроблений програмний засіб взаємодіє з різними ресурсами. Серед цих ресурсів ключовими є головний файл програми, заголовний файл із ресурсами програми, відкритий файл для визначення його геш-значення та файл для збереження результатів для подальшого наочного

представлення та порівняння. Схема ресурсів програмного засобу зображена на рисунку 3.4.

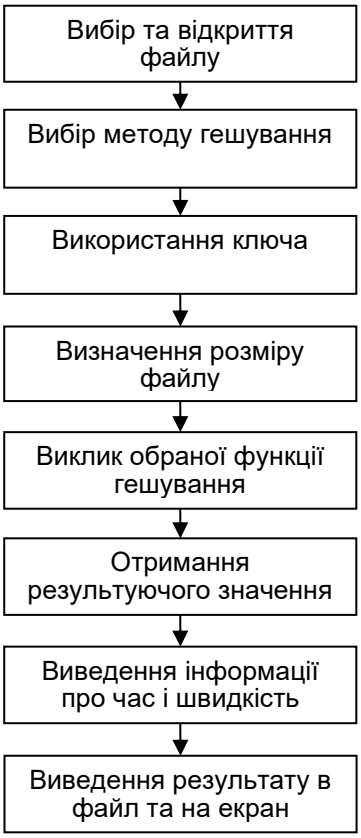


Рисунок 3.1 — Схема функціонування програми

Бази даних для зберігання інформації про всі файли, які піддалися гешуванню, та їх зміст, з метою подальшого наочного представлення та порівняння отриманих значень, використовується окремий текстовий файл. За замовчуванням у програмі цей файл має назву «result.txt», але можна змінити назву файлу та його шлях за потреби під час роботи програми. Структура інформації, що зберігається в цьому файлі, представлена у вигляді, який показано на рис. 3.5.

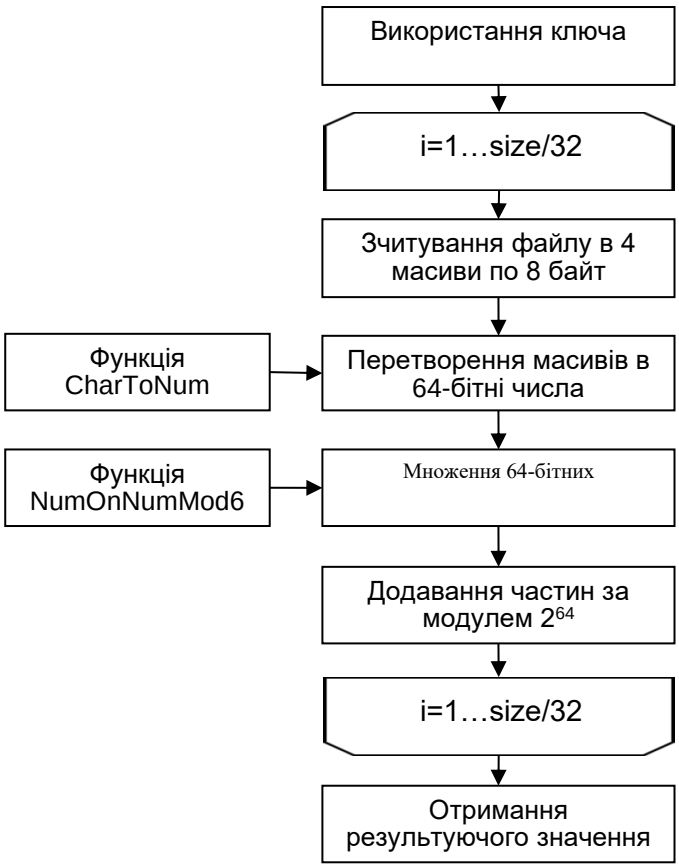


Рисунок 3.2 — Схема функції гешування даних на основі моделі кватерніона

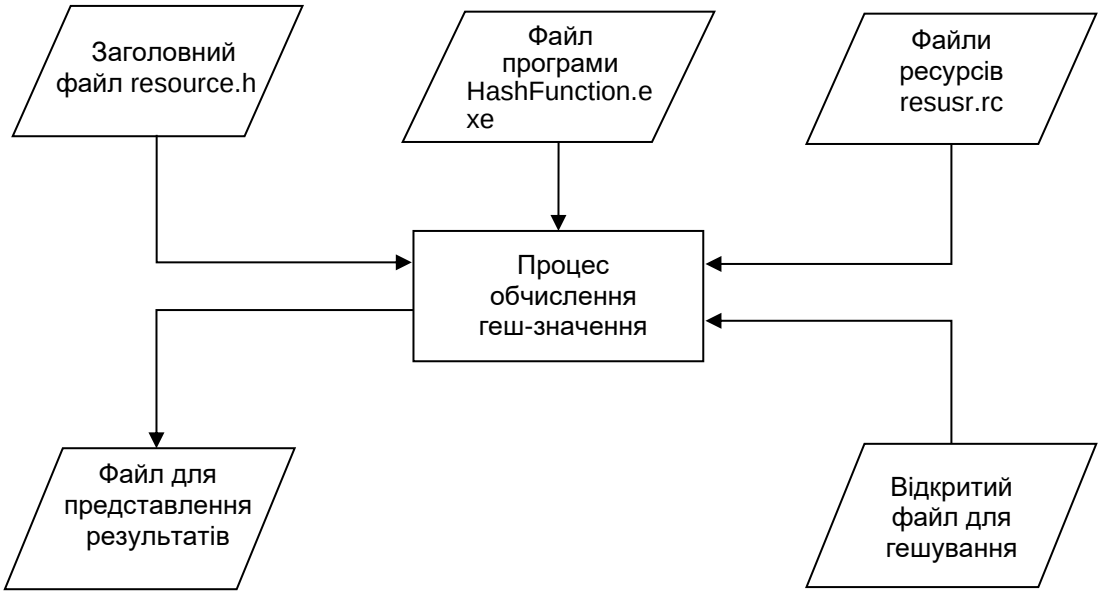


Рисунок 3.4 — Схема ресурсів програми

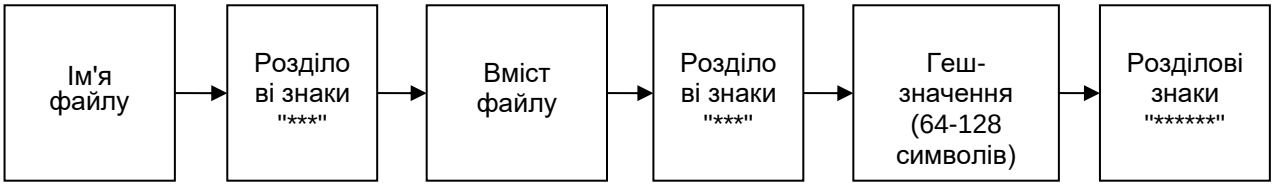


Рисунок 3.5 — Структура інформації про файли, що зберігається

Кожен рядок у файлі містить однаковий тип інформації про файли, які використовувались програмою для отримання геш-значення. Структура інформації, яка зберігається в цьому файлі, представлена на рис. 3.6.

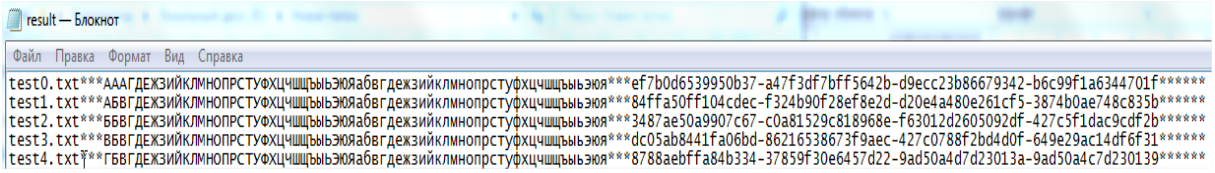


Рисунок 3.6 — Вигляд інформації що зберігається в файлі

Таким чином, була розроблена та розглянута схема функціонування програми, а також схема використання ресурсів програмного засобу. Крім того, була представлена структура інформації, що зберігається про файли, які використовувались для отримання геш-значення, та їх відповідні результати.

3.4 Інтерфейс та функції програмного засобу

Під час розробки інтерфейсу та функціоналу програмного засобу були враховані всі вимоги. З метою забезпечення зручності роботи, був створений простий та інтуїтивно зрозумілий інтерфейс. Використані елементи включають текстові поля для введення та відображення даних, кнопки та перемикачі для полегшення використання, а також панель інструментів із стандартними іконками для поліпшення сприйняття. На панелі інструментів розміщено три стандартні іконки, при наведенні курсору на них виводиться інформація про їхню функціональність (див. рис. 3.7).

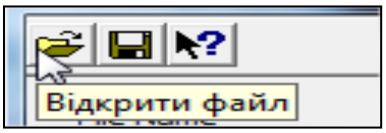


Рисунок 3.7 — Вигляд панелі інструментів

Використано стандарту графічну панель для максимального полегшення роботи з відкриттям та вибору файлів (рис. 3.6).

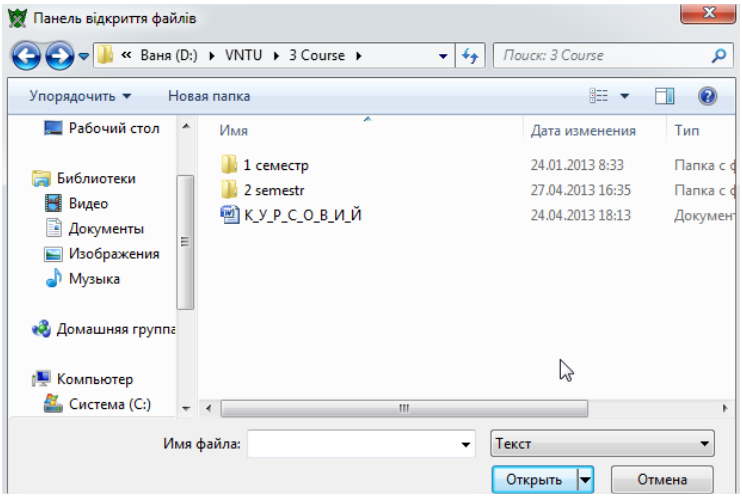


Рисунок 3.6 — Вигляд панелі відкриття файлів

На (рис. 3.7) показані радіо-батони вибору методу гешування.

Рисунок 3.7 — Набір перемикачів

З низу відкритого вікна розташований рядок стану (див. рис. 3.8), який відображає інформацію про відкритий файл, таку як метод гешування, розмір файлу, час, витрачений на отримання геш-значення, і швидкість обчислення.

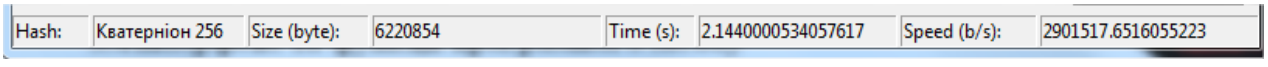


Рисунок 3.8 — Вигляд рядку стану

Програма включає наступні функції: читання масивів даних з файлу, зберігання необхідної інформації про результати гешування у віддільний файл, відображення необхідної інформації про файли та результати програми. Крім того, додаткові функції були реалізовані для спрощення обчислення геш-

значення. Кожен файл, який піддавався гешуванню, читався байт за байтом та зберігався у масиви по 4 або 8 байтів кожен. Відповідний код був написаний для цього:

```
for (int i=0; i<size/32; i++){
 fread(a1, sizeof(char), 8, filer);
 fread(b1, sizeof(char), 8, filer);
 fread(c1, sizeof(char), 8, filer);
 fread(d1, sizeof(char), 8, filer);}
```

Знаючи, як розроблений алгоритм гешування використовує математичні операції добутку і множення, було необхідно конвертувати зчитані дані в числа великої розрядності. Для цього була створена окрема функція, яка трансформувала 8-байтові масиви даних у числа з розміром 64 біти. Програмно, функцію конвертації 8-байтових масивів даних в числа було реалізовано наступним чином:

```
for(int i=0; i<8; i++)
        {a=a1[i];
        if (a<0) {b=256+a1[i];}
        else {b=a1[i];}
        a1n=a1n+NumOnNumMod64(b,pow(256.0,i));}
```

Актуальна збережена інформація про файл:

```
fputs(szFileOne,files);
fputs(buffc,files);
fputs(hashbuff,files);
```

Для візуалізація вигляду необхіодні інформації про швидкодію був використаний такий код:

```
start=clock();
end=clock();
t=((float)end-start)/CLK_TCK;
gcvt(t,20,time);
speed=size/t;
```

В результаті проведених дій, під час розробки програмного засобу, реалізували зручний та гнучкий графічний інтерфейс з усім необхідним для зручного і зрозумілого користування.

3.5 Реалізація гешування

Програмний продукт було створено для впровадження розроблюваних алгоритмів гешування даних, що виробляють геш-значення довжини 128, 256 та 512 біт.

Розроблювані алгоритми передбачають використання ключа або початкового значення, яке застосовується на першій ітерації обчислення геш-значення. У програмі було створено чотири окремих ключі для кожного варіанту методу обчислень. Ключ представляє собою комбінацію чотирьох або вісьми окремих частин, кожна з яких є 32-бітним або 64-бітним числом, представленим у шістнадцятковому форматі. Приклад представлення одного з ключів у програмі має наступний вигляд:

```
static unsigned long long Key[4]={0xfffeda76342dab51, 0xdb6523baf23f9013,
0xabcdefab43123dab, 0x98765abcdf323ec4};
```

Розроблювані алгоритми передбачають обробку вхідних даних у чотири рівні частини за одну ітерацію. З урахуванням цього, дані зчитувалися з файлу у вигляді чотирьох масивів по 8 байт. Процедуру зчитування реалізовано у відповідній функції, яку було описано раніше [16].

Після отримання порції даних необхідно перетворити їх з масиву у числа 32 або 64-бітної розрядності. Для вирішення цього завдання було розроблено відповідну функцію перетворення вхідних даних. Виклик цієї функції для чотирьох масивів виглядає наступним чином:

```
:
a1n=CharToNum(a1,4);
b1n=CharToNum(b1,4);
c1n=CharToNum(c1,4);
```

```
d1n=CharToNum(d1,4);
```

Враховуючи можливість операцій з числами до 64 біт у використовуваній мові програмування та необхідність виконання операцій множення і додавання з числами такої розрядності, яка може призвести до виходу за ліміти, були створені функції множення та додавання для чисел з 64-бітною розрядністю. У реалізації функції множення кожне 64-бітне число розкладалося на два 32-бітних числа, між якими виконувалися операції множення. Для виділення старших та молодших бітів вхідного числа 64-бітної розрядності використовувалися операції зсуву та накладання масок. Програмно це виглядає наступним чином:

```
a=x>>32;
b=x&0x00000000ffffffff;
c=z>>32;
d=z&0x00000000ffffffff;
```

Отримані числа вже мали розрядність, яка вдвічі менша, ніж у вхідних числах, що дозволяло виконувати всі потрібні нам операції над ними. Отримавши 32-бітні числа множилися між собою, утворюючи чотири нові 64-бітні числа. Після цього ці числа додавалися порозрядно в залежності від того, з яких частин вхідного числа вони були утворені. На виході отримували два 64-бітні числа. Для їх утворення використовувався наступний код:

```
y1=x22+x32;
    if (y1>0x00000000ffffffff)
    {x21=x21+1; y1=fmod(y1,4294967295.0); }
    y1=y1+x11;
    if (y1>0x00000000ffffffff)
    {x21=x21+1; y1=fmod(y1,4294967295.0); }
    y1=y1<<32; y1=y1+x12;
    y2=x4+x21+x31;
```

Вирахувані числа наразі представляють молодші та старші розряди 128-бітного числа, сформованого в результаті двох множень вхідних 64-бітних чисел. Оскільки алгоритм не оперує 128-бітними числами, необхідно обрати одне 64-

бітне число. Для додаткового ускладнення дані алгоритму передбачено порозрядне додавання за модулем двох частин цього числа, які відображають його молодші та старші розряди. Для цього реалізовано відповідну функцію додавання за модулем 2^64, виклик якої виглядає наступним чином:

```
y=NumPlusNum64(y1,y2);
```

Щоб реалізувати дії розроблюваних алгоритмів було написано окремі функції. Для реалізації алгоритму на основі моделі кватерніонів, з геш-значенням розрядністю 256 біт, було написано такий код:

```
s[0]=NumToNumMod64(a1,a2)-NumToNumMod64(b1,b2)-NumToNumMod64(c1,c2)-
NumToNumMod64(d1,d2);
s[1]=NumToNumMod64(a1,b2)+NumToNumMod64(b1,a2)+NumToNumMod64(c1,d2)-
NumToNumMod64(d1,c2);
s[2]=NumToNumMod64(a1,c2)+NumToNumMod64(c1,a2)+NumToNumMod64(b1,d2)-
NumToNumMod64(d1,b2);
s[3]=NumToNumMod64(a1,d2)+NumToNumMod64(d1,a2)+NumToNumMod64(b1,c2)-
NumToNumMod64(c1,b2);
```

Отже, програмна реалізація розроблених алгоритмів паралельного гешування даних на основі моделей кватерніонів була успішно завершена, а також надано опис допоміжних функцій, які використовувались для обчислення геш-значення вхідних даних.

3.6 Тестування роботи розробленого програмного засобу

Розроблена програма володіє інтуїтивно зрозумілим та легким у використанні графічним інтерфейсом. Усі елементи вікна, такі як поля для введення та кнопки, мають чіткі підписи, які чітко вказують на їхні функції.

Після запуску програми відображається вікно з нестандартною формою (рис. 3.9), де присутні назва програми, ім'я розробника та дві кнопки. З цього вікна можна перейти до головного діалогового вікна, де будуть виконуватись всі функції програмного засобу, або ж виходити із програми негайно [17].

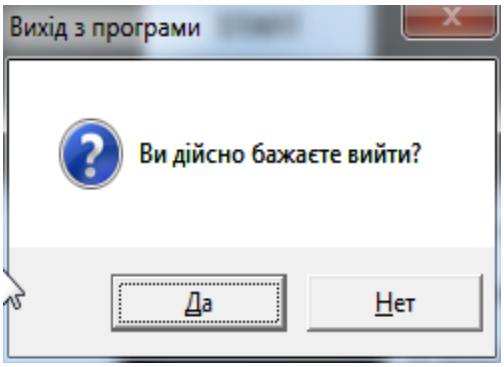


Рисунок 3.9 — Вікно запиту про вихід з програми

При натисканні на «START» відкривається діалогове вікно програми (рис. 3.10).

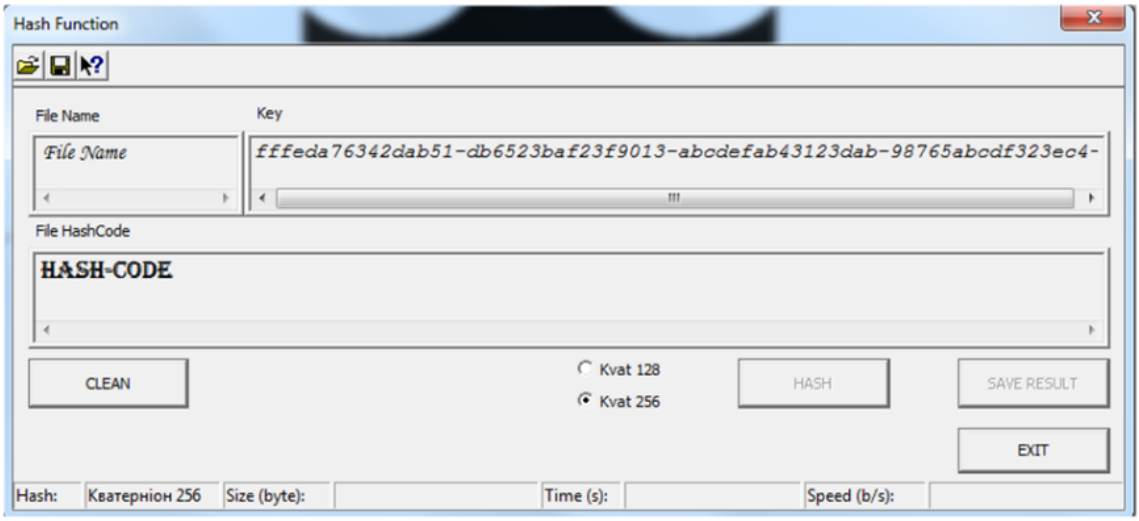


Рисунок 3.10 — Основне діалогове вікно програми

Коли користувач натискає кнопку з назвою "EXIT", на екрані з'являється сповіщення, яке запитує чи дійсно він хоче завершити роботу програми (рис. 3.10). Цей запит було враховано для уникнення випадкового виходу з програми.

Основні можливості програми реалізовані в цьому вікні. Тут розміщені три текстові поля, які використовуються для виведення імені відкритого файлу, значення ключа, що використовується програмою, та обчисленого геш-значення файлу за допомогою розроблених алгоритмів. Додатково, є панель інструментів, яка дозволяє відкривати файли, змінювати шлях для збереження та отримувати інформацію про програмний засіб.

Перед тим як почати використовувати програму, користувач може прочитати основні відомості про неї, які подані в окремому діалоговому вікні (рис. 3.11).

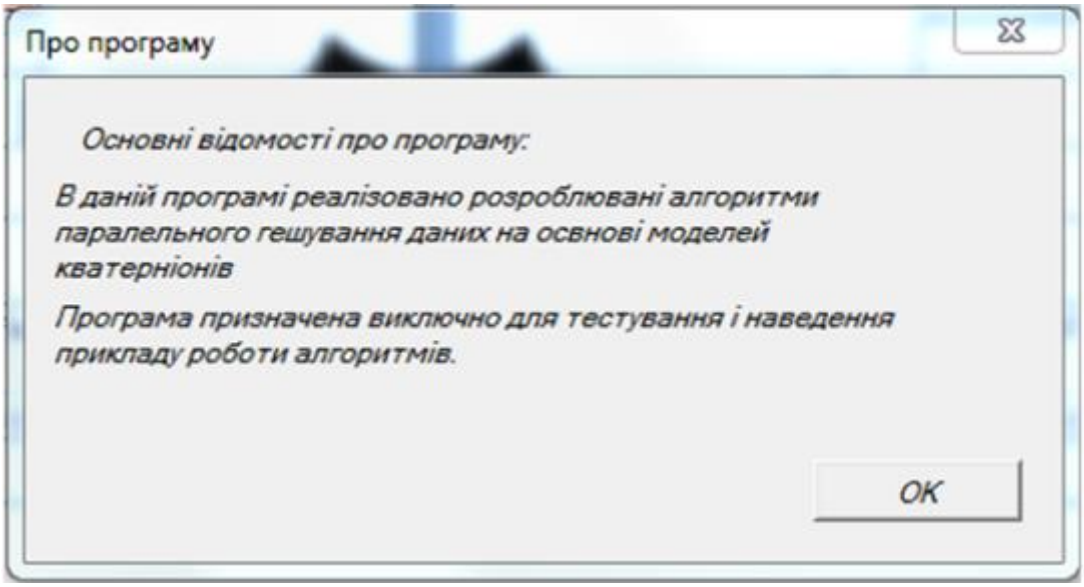


Рисунок 3.11 — Вікно з інформацією про програму

До обрання файлу є можливість заміни шляху і назви файлу де буде зберігатись дані (рис. 3.12).

Рисунок 3.12 — Діалогове вікно для зміни шляху збереження файлу

Усі вказані компоненти програми функціонують бездоганно і повністю виконують свої завдання. Крім того, проведено перевірку обробки всіх можливих помилок, які можуть виникати при використанні програми недосвідченим користувачем. Наприклад, система передбачає можливість введення неправильного шляху для збереження файлу, і у такому випадку користувач отримає повідомлення про помилку та програма автоматично використає стандартні налаштування (рис. 3.13).

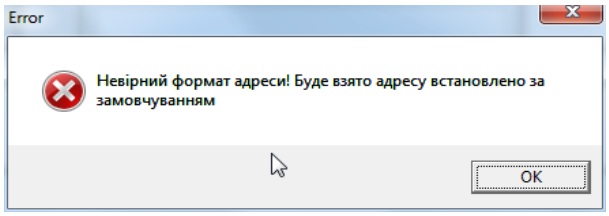


Рисунок 3.13 — Помилка при введенні невірного шляху для збереження файлу

Оскільки програма взаємодіє з різними типами файлів, передбачено обробку помилок, які можуть виникати при безпосередньому взаємодії з ними. Якщо виникає ситуація, коли неможлива робота з певним файлом, користувач буде проінформований через спеціальне повідомлення про помилку (рис. 3.14).

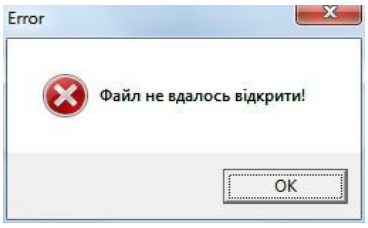


Рисунок 3.14 — Помилка при відкритті файлу

У процесі тестування програми: всі вікна відкривалися, і необхідна інформація про файли виводилась на екран. У ході використання програми не було виявлено непередбачених помилок, оскільки всі можливі ситуації були узгоджені і враховані в програмному коді. Будь-які помилки, які виникали під час тестування, були допущені з уміру, і програма правильно реагувала на них, виводячи відповідне повідомлення про помилку на екран.

3.7 Аналіз роботи програми

Після проведення тестування програми виявлено відсутність помилок та значущих дефектів. Наступним етапом є аналіз коректності роботи програми та оцінка ефективності розроблених алгоритмів гешування. Під час відкриття файлу програма автоматично виводить на екран інформацію, яка включає ім'я файлу, значення ключа, розмір відкритого файлу та обраний метод гешування.

Цю інформацію можна переглянути в основному вікні програми (рис. 3.15).

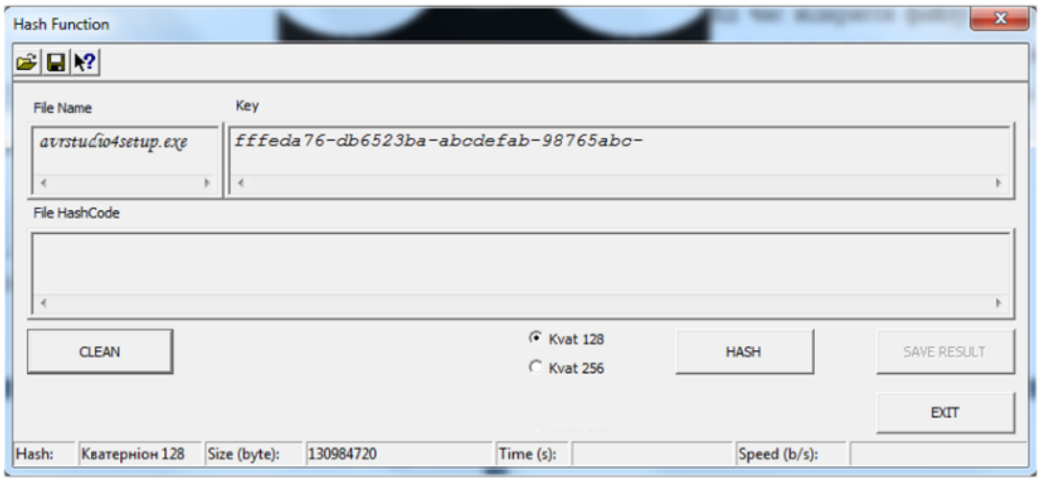


Рисунок 3.15 — Інформація про зчитуваний файл та ключ

Виведена на екран вся інформація про файл та значення ключа представлена коректно, як видно на зображенні вище. Після обрання методу гешування та натискання кнопки "HASH" виконується обраний алгоритм гешування. Отримане геш-значення, яке складається з 64 символів шістнадцяткового коду, визначене з байтової послідовності файлу, виводиться в поле "File HashCode". На рис. 3.16 представлені результати виконання різних алгоритмів.

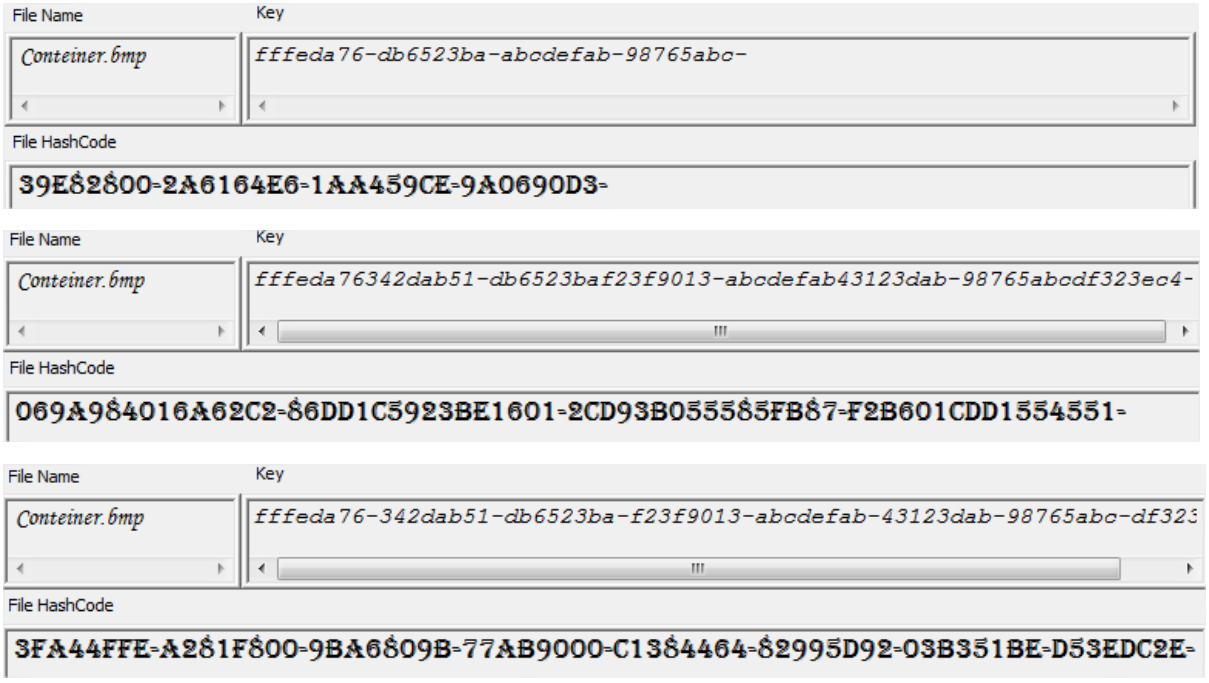


Рисунок 3.16 — Інформація про результат гешування файлу різними методами

Після цього можна натиснути кнопку «SAVE RESULT» щоб зберегти отримані результати гешування для подальшого їх порівняння.

Для повного тестування роботи програми було обчислено геш значення з різних файлів кожним з розроблюваних методів.. Кожен з цих файлів відрізнявся від попереднього.

На рис. 3.17 зображено результати обчислень геш-значень..

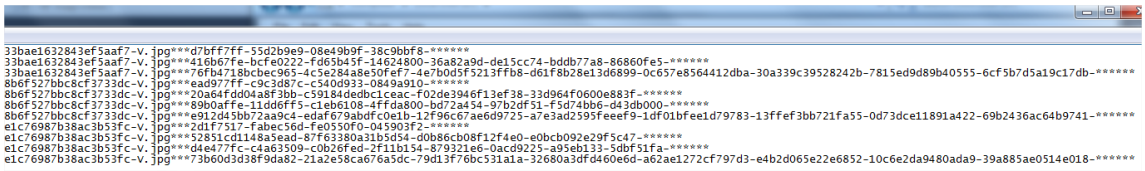


Рисунок 3.17 — Відкриття файлу з результатами

З рисунку 3.17 видно, що застосування різних методів гешування призвело до отримання відмінних геш-значень на всій довжині. Додатково був поданий приклад повторного гешування одного й того ж файлу для переконання, що алгоритм працює правильно і згенеровані значення не залишаються незмінними.

З результатів аналізу роботи програми можна визначити, що вона функціонує коректно і ефективно, виконуючи своє основне призначення — реалізацію розроблених алгоритмів гешування даних. Таким чином, програма готова для використання у порівнянні результатів.

3.8 Лавинний тест

Криптографічних функцій гешування даних великим значенням є здатність забезпечувати лавинний ефект, тобто суттєві зміни в значенні функції при маленьких змінах вхідного повідомлення. Це важливо, оскільки значення гешу не повинно видачі жодної інформації навіть щодо окремих бітів вхідного аргументу. Ця вимога є ключовою для криптостійкості алгоритмів гешування, які, зокрема, використовуються для гешування паролей користувачів та отримання ключів.

Тестування проводилося для розроблених методів, зокрема:

- На основі моделі кватерніонів із геш-значенням розрядністю 256 біт.

Для проведення тестів розроблені чотири текстові файли, різні за довжиною повідомлень, від 4 до 812 байт (рис. 6.1).

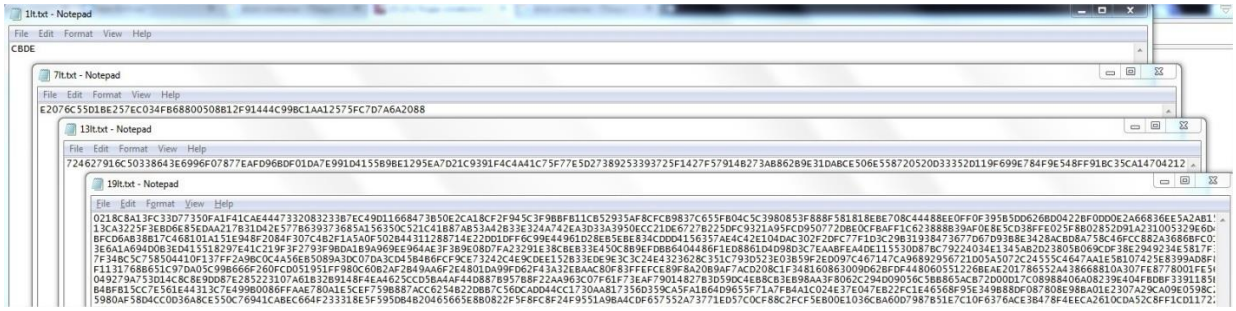


Рисунок 3.18 – Вигляд файлів, що використовувались для тестування

Далі було створено по 5 модифікацій для кожного з файлів:

– додавання одного байту в кінець файлу;

– віднімання одного байту з кінця фалу;

– зміна одного біту в кінці файлу;

– зміна одного біту на початку файлу файлу;

– зміна одного біту в середині файлу.

Це було виконано для перевірки, чи будь-яка зміна в будь-якій частині повідомлення впливає на отримане геш-значення. Оскільки файл представляється у вигляді байтів, для зміни одного біту використовувалася система ASCII-кодів. За допомогою цієї системи байти повідомлення переводились у двійковий вигляд, і знаходився найближчий за значенням байт. В результаті байт, значення якого у двійковому вигляді відрізнялося всього на 1 біт, замінювався.

Приклад модифікації одного з файлів наведено на рис. 3.19, де кожен змінений байт виділено.

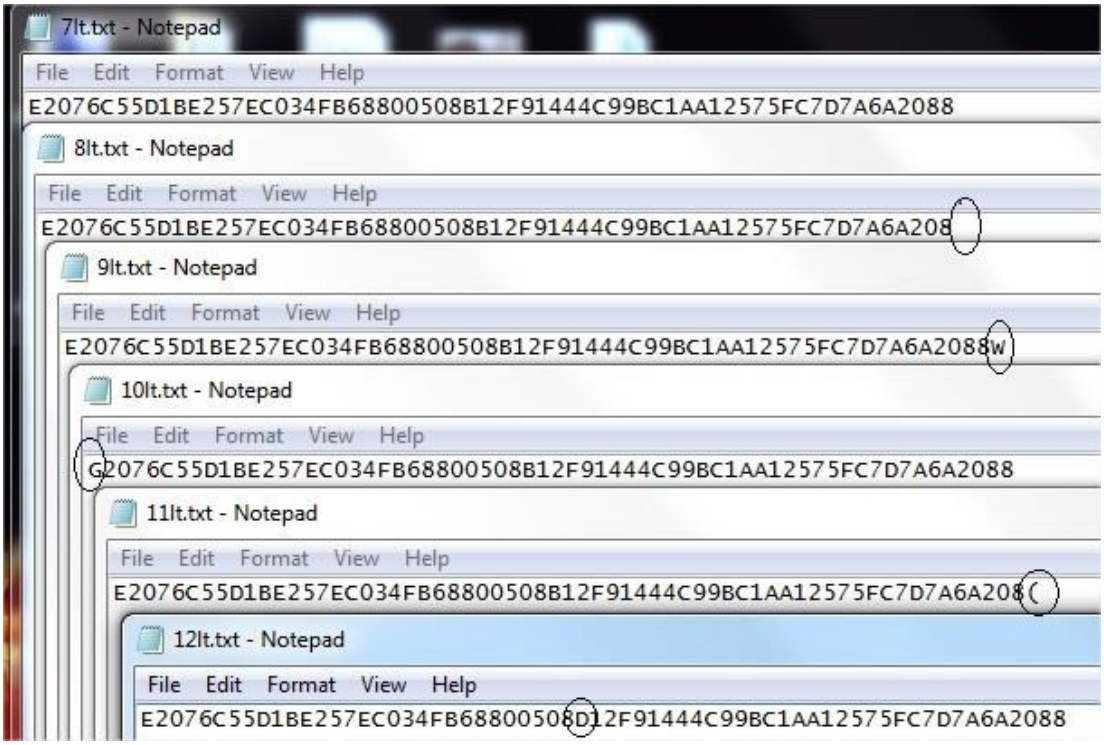


Рисунок 3.19 – Приклад модифікацій одного файлу

Після цього було обчислено геш-значення для кожного файлу і всіх його модифікацій. Результати для геш функції на основі моделі кватерніонів, з геш-значенням розрядністю 256 біт наведено на рисунку 3.20.

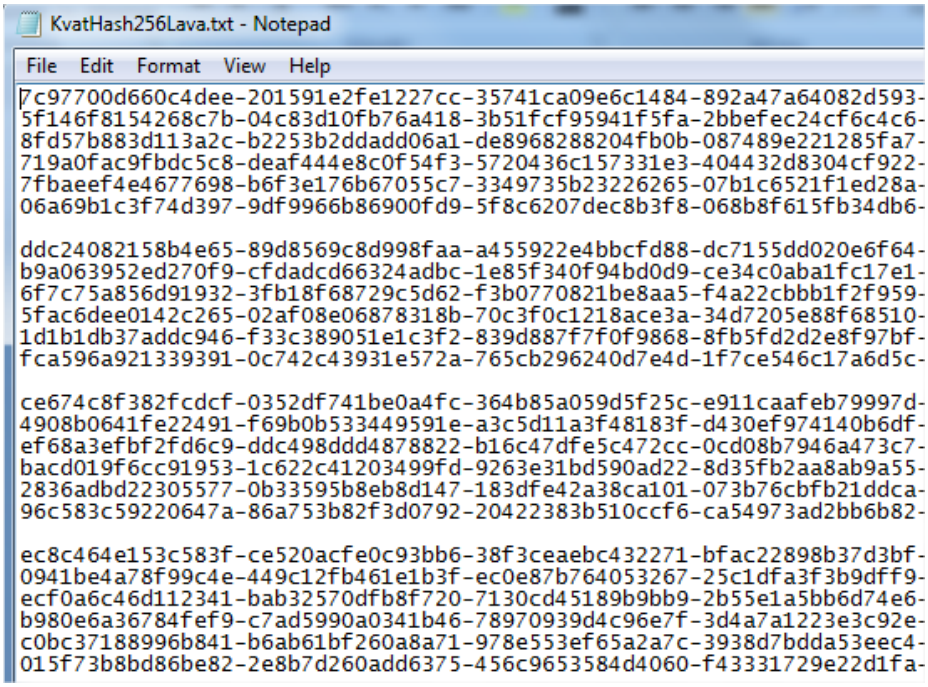


Рисунок 3.20 – Результати тестування геш-функції на основі моделі кватерніонів

Візулаьне представлення отриманих результатів відображає, що геш-значення явно змінюються і відрізняються одне від одного, але для більш детального оцінювання буде визначено відстань за Хемінгом між солідніми геш значеннями. Результати наведено в таблиці 3.1.

Таблиця 3.1 – Результати обчислення відстані Хемінга

Алгоритм	Довжина геш-значення (біт)	Мінімальна Хемінгова відстань	Максимальна Хемінгова відстань	Середня Хемінгова відстань
На основі моделі кватерніонів	256	108	137	123

Отримані результати, можна визначити, що геш-значення зазнає змін по всій своїй довжині, і для різних модифікацій файлів не має однакових значень. Виявлена закономірність в результатах першого блоку геш-значення, отриманого з короткого повідомлення завдовжки 4 байти, не є суттєвою проблемою. З огляду на те, що такі короткі повідомлення рідко використовуються і їх легко підібрати звичайним перебором, ця обставина не впливає на загальну ефективність алгоритмів. Отже, можна зробити висновок, що розроблені методи гешування успішно пройшли тест на лавинний ефект для повідомлень різної довжини.

4 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка може бути прийнята та успішно впроваджена, якщо вона відповідає вимогам сучасності в напрямку науково-технічного прогресу та враховує економічні аспекти. Тому необхідно оцінювати економічну ефективність отриманих результатів науково-дослідної роботи.

Магістерська кваліфікаційна робота з розробки та дослідження за темою «Метод пришвидшеного гешування даних» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки. Цей напрямок визначається як пріоритетний, оскільки розроблені результати можуть бути використані різними зацікавленими сторонами, приносячи економічні вигоди. Однак для здійснення цього процесу необхідно знайти потенційного інвестора, який був би зацікавлений у втіленні цього проекту, і переконати його у виправданості вкладання інвестицій в даний проект.

Для цього визначені наступні етапи виконання робіт:

1. Проведено комерційний аудит науково-технічної розробки, що включає в себе визначення науково-технічного рівня та комерційного потенціалу.

2. Розраховані витрати на реалізацію науково-технічної розробки.

3. Проведено розрахунок економічної ефективності науково-технічної розробки в разі її впровадження та комерціалізації потенційним інвестором, а також обґрунтовано економічну доцільність комерціалізації для інвестора.

4.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Метод пришвидшеного гешування даних» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в табл. 4.1 [22].

Таблиця 4.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї

9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь- які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно звести до таблиці. Для опитування було залучено трьох незалежних експертів:

1. Кузін Олег Володимирович - директор IT компанії DevOcean та головний sale manager компанії.

2. Лужецький Володимир Андрійович - Керівник: д. т. н., професор каф. ЗІ

3. Мельничук Оксана Павлівна - Кандидат економічних наук доцент, Вінницький торговельно-економічний інститут Державного торговельно-економічного університету.

Таблиця 4.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	Кузін Олег Володимирович	Лужецький Володимир Андрійович	Мельничук Оксана Павлівна
	Бали:		
1. Технічна здійсненність концепції	3	3	3
2. Ринкові переваги (наявність аналогів)	4	4	4
3. Ринкові переваги (ціна продукту)	4	3	4
4. Ринкові переваги (технічні властивості)	3	3	2
5. Ринкові переваги (експлуатаційні витрати)	3	3	4
6. Ринкові перспективи (розмір ринку)	2	2	1
7. Ринкові перспективи (конкуренція)	3	3	3
8. Практична здійсненність (наявність фахівців)	2	2	3
9. Практична здійсненність (наявність фінансів)	1	1	1
10. Практична здійсненність (необхідність нових матеріалів)	4	4	4
11. Практична здійсненність (термін реалізації)	3	3	4
12. Практична здійсненність (розробка документів)	4	4	4
Сума балів	$СБ_1=36$	$СБ_2=35$	$СБ_3=37$
Середньоарифметична сума балів $СБ_с$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{36+35+37}{3} = 36$		

За результатами розрахунків, наведених в таблиці 4.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 4.3 [22].

Таблиця 4.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Метод пришвидшеного гешування даних» становить 36 балів, що, відповідно до таблиці 4.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

Розробка реалізується програмним шляхом. А саме заміненням існуючих методів гешування на пропонований. Буде реалізована у окремому програмному застосунку, та протестована в технології блокчейн. Користування даним методом дуже обширне, починаючи від цивільного життя де вона зможе знайти в простих речах як пошук інформації чи обчислювальні операції так і в складних таких як транзакції, блокчейн технології та смарт контракти, та закінчуючи військовими та іншими держ. структурами.

4.2 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Метод пришвидшеного гешування даних», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

4.2.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [22]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \qquad (4.1)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дн.;

T_p – середнє число робочих днів в місяці, T_p=21 дні.

$$З_о = 20000 \cdot 5 / 21 = 4545 \; грн.$$

Проведені розрахунки зведемо до таблиці.

Таблиця 4.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	20000	909,1	5	4545
Інженер-програміст	50000	2272,7	20	45455
Дата аналітик	40000	1818,2	20	36364
Всього				86364

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему «Метод пришвидшеного гешування даних» розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i \,, \tag{4.2}$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}} \,, \tag{4.3}$$

де M_M – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $Мм$=6500 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення

тарифної ставки робітнику відповідного розряду (табл. Б.2, додаток Б) [22];

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

T_p – середнє число робочих днів в місяці, приблизно T_p = 22 дн;

$t_{зм}$ – тривалість зміни, год.

$C_1 = 6500{,}00 \cdot 1 \cdot 1{,}65 / (22 \cdot 8) = 62{,}8$ *грн.*

$З_{р1} = 62{,}8 \cdot 16 = 609{,}1$ *грн.*

Таблиця 4.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
1. Аналіз вимог та формулювання специфікацій	16	1	62,8	1005,0
2. Онтологічне моделювання та проектування бази знань	32	3	84,8	2713,5
3. Розробка програмного коду та системної архітектури	30	5	106,8	3203,4
4. Тестування та оптимізація розробленого рішення	40	3	84,8	3391,9
Всього				10313,8

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (4.4)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 11%.

$З_{дод}$ = (86364+ 10313,8) · 11 / 100% = 10634,52 грн.

4.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (4.5)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

$Зн$ = (86364+ 10313,8+ 10634,52) · 22 / 100% = 23608,63 грн.

4.2.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за темою «Метод пришвидшеного гешування даних».

Витрати на матеріали ($М$), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$М = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{вj}\ , \quad (4.5)$$

де $Н_j$ – норма витрат матеріалу j-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу j-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = 1,1 ... 1,15);

$В_j$ – маса відходів j-го найменування, кг;

$Ц_{вj}$ – вартість відходів j-го найменування, грн/кг.

Проведені розрахунки зведемо до таблиці.

Таблиця 4.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Вартість витраченого матеріалу, грн
Папір для записів FAXE 70 A5-250	200	1	200
Органайзер офісний OFFICE 100	220	1	220
Диск оптичний OPTIMA CD	15,5	2	31
Flesh-пам'ять GOODRAM 64 C10A	420	1	420
Всього			871
З врахуванням коефіцієнта транспортування			958,1

4.2.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Метод пришвидшеного гешування даних» відсутні.

4.2.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення.

Балансову вартість спецустаткування розраховуємо за формулою:

$$B_{спец} = \sum_{i=1}^{k} Ц_i \cdot C_{пр.i} \cdot K_i \ , \qquad (4.6)$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, марки, грн;

$C_{пр.i}$ –кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, ($K_i = 1,10…1,12$);

k – кількість найменувань устаткування.

$B_{спец} = 44000 \cdot 3 \cdot 1,11 = 145200$ *грн.*

Отримані результати зведемо до таблиці:

Таблиця 4.7 – Витрати на придбання спецустаткування по кожному виду

Найменування устаткування	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Apple MacBook Air 2022	3	44000	145200
Всього			145200

4.2.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (4.7)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_i = 1{,}10…1{,}12$);

k – кількість найменувань програмних засобів.

Оскільки в роботі використовувалися безкоштовні програмні засоби, тому дані витрати не враховані.

4.2.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12} , \qquad (4.8)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо,

які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (45000 \cdot 1) / (2 \cdot 12) = 1875$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 4.10 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Комп'ютер	45000	2	1	1875,00
Принтер	3000	2	1	125,00
Приміщення лабораторії	250000	20	1	1041,67
Всього				3041,67

4.2.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (B_e) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \quad (4.9)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e = 7,5$ грн;

$K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0,25 \cdot 240,0 \cdot 7,5 \cdot 0,5 / 0,8 = 281,25$ *грн.*

4.2.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи на тему «Метод

пришвидшеного гешування даних» належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \quad (4.10)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 20%.

$В_{св}$ = (86364+ 10313,8) · 20 / 100% = 19335,39 грн.

4.2.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» відсутні.

4.2.11 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (4.11)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$І_в$ = (86364+ 10313,8) · 50 / 100% = 48338,72 грн.

4.2.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (4.12)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 100%.

$В_{нзв}$ = (86364+ 10313,8) · 100 / 100% = 96677,45 грн.

Витрати на проведення науково-дослідної роботи на тему «Метод пришвидшеного гешування даних». розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + B_{спец} + B_{прг} + А_{обл} + B_е + B_{св} + B_{сп} + І_в + B_{нзв}. \quad (4.13)$$

$В_{заг}$= 86364 +10313,8 +10634,52 +23608,63+958,1+145200+3041,67+281,25 +19335,39+48338,72+96677,45 = 234058,01 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (4.13)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,4.

ЗВ = 234058,01 / 0,4 = 1148183,2 грн.

4.3 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження проведені за темою «Метод пришвидшеного гешування даних» передбачають комерціалізацію протягом 3-х років реалізації на ринку.

В цьому випадку основу майбутнього економічного ефекту будуть формувати:

ΔN – збільшення кількості споживачів яким надається відповідна інформаційна послуга у періоди часу, що аналізуються;

N – кількість споживачів яким надавалась відповідна інформаційна послуга у році до впровадження результатів нової науково-технічної розробки, приймемо 1 особа

$Ц_б$ – вартість послуги у році до впровадження інформаційної системи, приймемо 50000,00 грн;

$\pm\Delta Ц_o$ – зміна вартості послуги від впровадження результатів, приймемо зростання на 1000,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [22]:

$$\Delta П_i = (\pm\Delta Ц_o \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \qquad (4.14)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на

додану вартість. У 2021 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту). Приймемо $\rho = 40\%$;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2023 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1 = (1 \cdot 1000 + 50000 \cdot 200) \cdot 0,83 \cdot 0,4 \cdot (1 - 0,18/100\%) = 1742601,1$ грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2 = (1 \cdot 1000 + 50000 \cdot (200 + 150)) \cdot 0,83 \cdot 0,4 \cdot (1 - 0,18/100\%) = 3050253$ грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3 = (1 \cdot 1000 + 50000 \cdot (200 + 150 + 100)) \cdot 0,83 \cdot 0,4 \cdot (1 - 0,18/100\%) = 3921468,2$ грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (4.15)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, $\tau = 18\%$;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = 1742601,1 /(1+0,18)^1+3050253/(1+0,18)^2+3921468,2 /(1+0,18)^3=$$
$$= 5850322,23 \text{ грн.}$$

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \quad (4.16)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=2;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 1148183,2 грн.

$PV = k_{інв} \cdot ЗВ$ = 2 · 1148183,2 = 2296366,4 грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \quad (4.17)$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 5850322,23 грн;

PV – теперішня вартість початкових інвестицій, 2296366,4 грн.

$E_{абс} = ПП - PV$ = 5850322,23 - 2296366,4 = 3553955,83 грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (4.18)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, грн;

PV – теперішня вартість початкових інвестицій, грн;

$T_{ж}$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 3 роки.

$$E_{в} = \sqrt[Tж]{1+\frac{E_{абс}}{PV}} - 1 = (1+3553955{,}83\ /2296366{,}4)^{1/3} - 1 = 0{,}6.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f\,, \tag{4.19}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d =0,1;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,25.

$\tau_{мін}$ = 0,1+0,25 = 0,35 < 0,6 свідчить про те, що внутрішня економічна дохідність інвестицій $E_{в}$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Метод пришвидшеного гешування даних» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_{в}}\,, \tag{4.20}$$

де $E_{в}$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 0,6 = 1,7 р.

$T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

ВИСНОВКИ

В магістерській кваліфікаційній роботі розроблено метод пришвидшення гешування даних, що представляється у вигляді кватерніонів. Дана задача є актуальною оскільки визначення геш-значень за допомогою відомих геш-функції вимагає багато машинного ресурсу і тому займає багато часу.

В результаті виконання роботи було проведено аналіз літературних джерел, які показали, що таке геш-функції, основи їх побудови, застосування їх на практиці. Також були проаналізовані сучасні методи та функції гешування даних, та розглянуті методи пришвидшеного гешування даних.

В роботі був описаний кватерніон та метод гешування на основі моделі кватерніона після чого було проведене порівняння та аналіз пропонованого метода з вже відомими.

Сформульовано вимоги до розроблюваного програмного засобу, який впроваджує запропонований алгоритм гешування даних, та вмотивовано вибір мови програмування для його реалізації. До реалізації програмного засобу попередньо розроблено та описано схему його функціонування та ресурсів, а також представлено структуру інформації, яка зберігається в базі даних, стосовно файлів, використовуваних для отримання геш-значення, і їх результатів.

Після цього створено програмний продукт, який реалізує метод швидкого гешування даних. Описано його інтерфейс та визначено основні функціональні можливості, які були втілені. Розроблений програмний засіб вдосконалено протестовано, і було проведено докладний аналіз його роботи.

У розгляді для дослідження взятий метод гешування на основі моделі кватерніонів з розрядністю 256 біт. Метод успішно пройшов тест на лавинний ефект і продемонстрував відмінні результати у інших випробуваннях.

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Метод пришвидшеного гешування даних» становить 36,0 балів, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

Також термін окупності становить 1,7 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Метод пришвидшеного гешування даних.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Принципи гешування [Електронний ресурс], - Режим доступу: URL :https://academy.binance.com/uk/articles/what-is-hashing – Алгоритм sha-256

2. Геш в блокчейні [Електронний ресурс], - Режим доступу: URL :https://tradinginfo.club/ - стійкість геша в блокчейні.

3. Лужецький В. А. Конструкції гешування стійкі до мультиколізій [Електронний ресурс] / В. А. Лужецький, Ю. В. Баришев // Наукові праці Вісник Вінницького політехнічного інституту. — № 1. — 2010. — 8 с. — Режим доступу : http://www.nbuv.gov.ua/e-journals/vntu/2010_1/2010-1.files/uk/1 Ovalsam_ua.pdf.

4. iSearch [Електронний ресурс] – Режим доступу: URL : http://www.isearch.kiev.ua/index.php/uk/searchpractice/internetsecurity/837-hashing-message-digest – Гешування і захист інформації.

5. Кобзар І. В. Пришвидшення процесу обчислень значень геш-функцій на основі використання моделі кватерніонів / І. В. Кобзар, В. А. Лужецький // Тези доповідей Четвертої Міжнародної науково-практичної конференції «Інформаційні технології та коп‘ютерна інженерія» м. Вінниця, 28-30 травня 2014 року. – Вінниця: ВНТУ, 2014. – с. 195-198.

6. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы и исходные тексты на языке С. 2-е изд. / Брюс Шнайер. – СПб. : Вильямс, 2000. – 789 с. – ISBN 5-89392-055-4.

7. Studopedia [Електронний ресурс] – Режим доступу: URL : http://studopedia.info/3-115290.html/ – Основи безпеки даних в комп'ютерних системах.

8. Van Houtven L. Crypto 101 / L. Van Houtven // Creative Commons Attribution-NonCommercial 4.0 International (CC BY-NC 4.0) – 2014 – 243 p.

9. Лужецкиц В.А. Криптографічні примітиви для реалізації керованого гешування / В.А. Лужецький, Ю.В. Баришев //Вісник ВПІ, - 2011. - №1.-С. 108-111.

10. E. Biham. A Framewoik for Iterative Hash Functions: HAIFA [Електронний ресурс] / E. Biham, O. Dunkelman // COSIC internal report, 2007. — 20

с. — Режим доступу до ресурсу: https://wwwxosic.esatJadeuven.be/pubHcations/aiticle-934.pdf.

11. National Bureau of Standards, NBS FIPS PUB 81, DES Modes of Operation, U. S. Department of Commerce, Dec 1980.

12. Joux A. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions / Antoine Joux // Lecture Notes in Computer Science. – 2004. – № 3152. – С. 306-316.

13. Гамільтон У. Р. О кватерніонах, чи про нову систему величин в алгебрі. / Оптика. Динаміка. Кватерніони. / У. Р. Гамільтон // – М.: «Наука» 1994. – 391 с.

14. Baez J. C. The octonions./ John Baez // Bull. Amer. Mathem. Soc. 39: 2, – 2002. – p. 145-205

15. Solutionmes [Електронний ресурс] – Режим доступу: URL : http://solutionmes.wikidot.com/crypto-sha/ – SECURE HASH Алгоритмы (SHA-1, SHA-2).

16. Vunivere [Електронний ресурс] – Режим доступу: URL : http://vunivere.ru/work23768/page5/ – Стандарты Геш-функций. Основные определения и требования безопасности к функциям гешированія.

17. Cypherpunks [Електронний ресурс] – Режим доступу: URL : http://www.cypherpunks.ru/hash/comparison.html/ – Сравнение хэш функций.

18. The KECCAK sponge function family [Електронний ресурс] – Режим доступу: URL : http://keccak.noekeon.org/ – KECCAK

19. Пат. 57342 Україна, МПК[7] G09C 1/00. Спосіб паралельного ключового гешування [Текст] / Лужецький В. А., Баришев Ю. В. (Україна); заявник та патентовласник ВНТУ „Він. нац. техн. ун-т.” - № u201008801; заявл. 15.07.10; опубл. 25.02.11, Бюл. № 4. – 4 с.

20. A Statistical Test Suite for the Validation of Random Number Generators [Електронний ресурс] / – Режим доступу : http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf

21. Pal Revesz. Random Walk in Random and Non-Random Environments. /

Revesz Pal – Singapore: World Scientific, 1990

22. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

ДОДАТКИ

ДОДАТОК А

ПРОТОКОЛ ПЕРЕВІРКИ
МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Метод пришвидшеного гешування даних
Автор роботи: Паламаренко Олег Ігорович

Тип роботи: магістерська кваліфікаційна робота

Підрозділ кафедра захисту інформації ФІТКІ
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність – 84,7 %. Схожість – 15,3 %.

Аналіз звіту подібності (відмітити потрібне):

√ 1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ 2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ 3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Валентина КАПЛУН
(підпис)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Олег ПАЛАМАРЕНКО
(підпис)

Керівник роботи _____ Володимир ЛУЖЕЦЬКИЙ
(підпис)

ДОДАТОК Б
ЛІСТИНГ ПРОГРАМИ

```
#include <windows.h>
#include <time.h>
#include <commctrl.h>
#include <fstream>
#include <sys\stat.h>
#include "resource.h"
#pragma comment (lib,"comctl32.lib")
static char szWindowClass[] = ("winClass1");
static char szTitle[] = ("HashFunction!!!");
static char adressf[510]="D:\\result.txt";
HWND hWnd;
HINSTANCE hInst;
HACCEL hAccel;
RECT r;
static HDC hdc;
static HBRUSH hb,hb1;
static COLORREF fonColor  = RGB(0,0,0), textColor = RGB(255,0,0);
static HFONT font;
static char str1[512]="HashFunction";
static char str4[512]="Hash Function";
static char str5[512]="version 1.1";
static CHOOSECOLOR cc;
static LOGFONT lf;
HDC scr = GetDC(NULL);
int w = GetDeviceCaps(scr,HORZRES);
int h = GetDeviceCaps(scr,VERTRES);
unsigned long long s[4]={0, 0, 0, 0};
unsigned long long s2[8]={0, 0, 0, 0, 0, 0, 0, 0};
unsigned long long s3[8]={0, 0, 0, 0, 0, 0, 0, 0};
unsigned long long s4[4]={0, 0, 0, 0};
static unsigned long long Key4[4]={0xfffeda76, 0xdb6523ba, 0xabcdefab, 0x98765abc};
static unsigned long long Key[4]={0xfffeda76342dab51, 0xdb6523baf23f9013, 0xabcdefab43123dab, 0x98765abcdf323ec4};
static unsigned long long Key2[8]={0xfffeda76, 0x342dab51, 0xdb6523ba, 0xf23f9013, 0xabcdefab, 0x43123dab, 0x98765abc,
0xdf323ec4};
static unsigned long long Key3[8]={0xfffeda76342dab51, 0xdb6523baf23f9013, 0xabcdefab43123dab, 0x98765abcdf323ec4,
0xfffeda76342dab51, 0xdb6523baf23f9013, 0xabcdefab43123dab, 0x98765abcdf323ec4};
static char keybuff1[75]= { NULL };
static char keybuff2[75]= { NULL };
static char keybuff3[150]= { NULL };
static char keybuff4[75]= { NULL };
int k=0;
char buff[80];
void KvatHash128 (FILE *filer);
void KvatHash256 (FILE *filer);
unsigned long long NumOnNumMod32 (unsigned long long a, unsigned long long b);
unsigned long long NumOnNumMod64 (unsigned long long x, unsigned long long z);
unsigned long long NumPlusNum64 (unsigned long long a, unsigned long long b);
unsigned long long CharToNum (char *a1, int numsize);

BOOL NewWindowClass(WNDPROC Proc, TCHAR szName[],UINT brBackground,UINT icon, UINT cursor, UINT menu)
{
            WNDCLASSEX wcex;
   wcex.cbSize = sizeof(WNDCLASSEX);
   wcex.style         = CS_HREDRAW | CS_VREDRAW;
   wcex.lpfnWndProc    = Proc;
   wcex.cbClsExtra     = 0;
   wcex.cbWndExtra     = 0;
   wcex.hInstance      = hInst;
   wcex.hIcon          = LoadIcon(NULL, IDI_APPLICATION);
   wcex.hCursor        = LoadCursor(hInst,(LPCSTR) cursor);
   wcex.hbrBackground  = (HBRUSH)(brBackground+9);
   wcex.lpszMenuName   = (LPCSTR)menu;
   wcex.lpszClassName  = szName;
   wcex.hIconSm        = LoadIcon(hInst, (LPCSTR)icon);

   if (!RegisterClassEx(&wcex))
   {   MessageBox(NULL,("Call to RegisterClassEx failed!"),szName, NULL);
      return 0;
   }
```

```
        else return 1;
}
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);
BOOL CALLBACK DlgProc1(HWND hdWnd,UINT mes,WPARAM wParam,LPARAM lParam);
BOOL CALLBACK DlgProc6(HWND hdWnd,UINT mes,WPARAM wParam,LPARAM lParam);
BOOL CALLBACK DlgProc8(HWND hdWnd,UINT mes,WPARAM wParam,LPARAM lParam);

int WINAPI WinMain(HINSTANCE hInstance,
          HINSTANCE hPrevInstance,
          LPSTR lpCmdLine,
          int nCmdShow)
{
   hInst = hInstance;

        if (!NewWindowClass(WndProc,szWindowClass,COLOR_WINDOW,IDI_S,IDC_C,NULL)) return FALSE;

        hWnd = CreateWindow(szWindowClass,szTitle, WS_OVERLAPPEDWINDOW|WS_VISIBLE, w/2-180, h/2-250,  370, 400,
NULL, NULL, hInst, NULL);

        PAINTSTRUCT ps;
        HDC hdc =BeginPaint(hWnd, &ps);
        HBRUSH br=(HBRUSH) CreateSolidBrush(RGB(0,0,0));
        GetClientRect(hWnd,&r);
        HRGN R1,R2;
        R1 = CreateEllipticRgn(-150,-300,355,400);
        R2 = CreateEllipticRgn(365+150,-300,10,400);
        CombineRgn(R1,R1,R2,RGN_AND);
        R2 = CreateEllipticRgn(-3,-30,142,80);
        CombineRgn(R1,R1,R2,RGN_DIFF);
        R2 = CreateEllipticRgn(220,-30,363+3,80);
        CombineRgn(R1,R1,R2,RGN_DIFF);
        R2 = CreateEllipticRgn(130,-30,232,55);
        CombineRgn(R1,R1,R2,RGN_DIFF);
        SelectObject(hdc,br);
        FillRect(hdc,&r,br);
        SetWindowRgn(hWnd,R1,TRUE);
        DeleteObject(br);

        lf.lfHeight = 18;
        strcpy (lf.lfFaceName, "Algerian");
        GetClientRect(hWnd,&r);
        HBRUSH br1;
        br1=CreateSolidBrush(fonColor);
        FillRect(hdc,&r,br1);
        SetTextColor(hdc,textColor);
        SetTextAlign(hdc,TA_CENTER);
        SetBkMode(hdc,TRANSPARENT);
        font=CreateFontIndirect(&lf);
        SelectObject(hdc,font);
        TextOut(hdc,r.right/2,r.bottom/2+55,str1,strlen(str1));
        TextOut(hdc,r.right/2,r.bottom/2+73,str2,strlen(str2));
        TextOut(hdc,r.right/2,r.bottom/2+91,str3,strlen(str3));
        lf.lfHeight = 12;
        font=CreateFontIndirect(&lf);
        SelectObject(hdc,font);
        TextOut(hdc,r.right/2,r.bottom/2+130,str5,strlen(str5));
        lf.lfHeight = 35;
        font=CreateFontIndirect(&lf);
        SelectObject(hdc,font);
        TextOut(hdc,r.right/2,70,str4,strlen(str4));
        DeleteObject(font);
        EndPaint(hWnd,&ps);
   if (!hWnd)
   {
                MessageBox(NULL,("Call to CreateWindow failed!"),("Win32 Guided Tour"),NULL);
      return 1;
   }

   MSG msg;
   while (GetMessage(&msg, NULL, 0, 0))
   {
                if (!TranslateAccelerator(hWnd,hAccel,&msg))
                {
```

```
                    TranslateMessage(&msg);
                    DispatchMessage(&msg);
                }
    }
        DestroyAcceleratorTable(hAccel);
    return (int) msg.wParam;
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
static HWND hButton1,hButton2;
    switch (message)
        {
        case WM_CREATE:
            hButton1 = CreateWindow("BUTTON", "START", WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 120, 135, 115,
35, hWnd, (HMENU)ID_BUTTON1, hInst, NULL);
            hButton2 = CreateWindow("BUTTON", "EXIT", WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 120, 195, 115,
35, hWnd, (HMENU)ID_BUTTON2, hInst, NULL);
            break;

        case WM_PAINT:
            PAINTSTRUCT ps;
            lf.lfHeight = 18;
            strcpy (lf.lfFaceName, "Algerian");
            hdc = BeginPaint(hWnd,&ps);
            GetClientRect(hWnd,&r);
            HBRUSH br1;
            br1=CreateSolidBrush(fonColor);
            FillRect(hdc,&r,br1);
            SetTextColor(hdc,textColor);
            SetTextAlign(hdc,TA_CENTER);
            SetBkMode(hdc,TRANSPARENT);
            font=CreateFontIndirect(&lf);
            SelectObject(hdc,font);
            TextOut(hdc,r.right/2,r.bottom/2+55,str1,strlen(str1));
            TextOut(hdc,r.right/2,r.bottom/2+73,str2,strlen(str2));
            TextOut(hdc,r.right/2,r.bottom/2+91,str3,strlen(str3));
            lf.lfHeight = 12;
            font=CreateFontIndirect(&lf);
            SelectObject(hdc,font);
            TextOut(hdc,r.right/2,r.bottom/2+130,str5,strlen(str5));
            lf.lfHeight = 35;
            font=CreateFontIndirect(&lf);
            SelectObject(hdc,font);
            TextOut(hdc,r.right/2,70,str4,strlen(str4));
            DeleteObject(font);
            EndPaint(hWnd,&ps);
            return 0;

        case WM_SIZE:
        case WM_MOVE:
            InvalidateRect(hWnd,NULL,TRUE);
            return 0;

        case WM_DESTROY:
            PostQuitMessage(0);
            break;

        case WM_COMMAND:
            switch(LOWORD(wParam))
            {
            case ID_BUTTON1:
                DialogBox(hInst,MAKEINTRESOURCE(IDD_DIALOG8),hWnd,DlgProc8);
                return 0;

            case ID_BUTTON2:
                if (MessageBox(hWnd, "Ви дійсно бажаєте вийти?","Вихід з програми",
MB_YESNO|MB_ICONQUESTION)==IDYES)
                PostQuitMessage(0);
                MessageBeep(MB_OK);
                break;
            }
            return 0;
```

```
   default:
      return DefWindowProc(hWnd, message, wParam, lParam);
      break;
   }
   return 0;
}
BOOL CALLBACK DlgProc1(HWND hdWnd,UINT mes,WPARAM wParam,LPARAM lParam)
{
	static HWND hEdit1;
	FILE *file;
	switch(mes)
	{
	case WM_INITDIALOG:
		hEdit1=GetDlgItem(hdWnd,IDC_EDIT1);
		SetWindowText(hEdit1,adressf);
		break;
	case WM_COMMAND:
		switch(LOWORD(wParam))
		{
		case IDOK:
			GetDlgItemText(hdWnd,IDC_EDIT1,adressf,510);
			if((file = fopen(adressf, "a"))==NULL)
			{
				MessageBox(hWnd, "Невірний формат адреси! Буде взято адресу встановлено за
замовчуванням","Error", MB_OK|MB_ICONHAND);
				puts("Error");
				SetWindowText(hEdit1,"D:\\result.txt");
				GetDlgItemText(hdWnd,IDC_EDIT1,adressf,510);
			}
			EndDialog(hdWnd,0);
			return 1;
		case IDOK2:
			SetWindowText(hEdit1,"D:\\result.txt");
			GetDlgItemText(hdWnd,IDC_EDIT1,adressf,255);
			EndDialog(hdWnd,0);
			return 1;
		case IDCANCEL:
			EndDialog(hdWnd,0);
			return 1;
		}
		return 1;
  }
  return 0;
}
BOOL CALLBACK DlgProc6(HWND hdWnd,UINT mes,WPARAM wParam,LPARAM lParam)
{
	switch(mes)
	{
	case WM_INITDIALOG:
		break;
	case WM_COMMAND:
		switch(LOWORD(wParam))
		{
		case IDOK:
			EndDialog(hdWnd,0);
			return 1;
		}
		return 1;
  }
  return 0;
}
BOOL CALLBACK DlgProc8(HWND hdWnd,UINT mes,WPARAM wParam,LPARAM lParam)
{
	time_t start, end;
	char time[50];
	double speed=0;
	char  speeds[50];
	float t=0;
	static HWND hEdit1,hEdit2,hEdit3,hButton1,hButton2;

	static char hashbuff[150]= { NULL };

	char st[66];
```

```
	FILE *filer, *files;
	static unsigned long long size=0;


	static HWND hStatus;
	static int parts[8];
	static int type=1;

	static OPENFILENAME ofn;
	static char szFileAll[512];
	static char szFileOne[512];
	static char szFilter[256]="Текст\0*.*\0Все форматы\0*.*\0";

	static HWND hToolBar;
	static TBBUTTON but[3];
	but[0].iBitmap = STD_FILEOPEN;
	but[0].idCommand = ID_OPENFILE;
	but[0].fsState = TBSTATE_ENABLED;
	but[0].fsStyle = TBSTYLE_BUTTON;
		but[1].iBitmap = STD_FILESAVE;
		but[1].idCommand = ID_SAVEFILE;
		but[1].fsState = TBSTATE_ENABLED;
		but[1].fsStyle = TBSTYLE_BUTTON;
			but[2].iBitmap = STD_HELP;
			but[2].idCommand = ID_MYHELP;
			but[2].fsState = TBSTATE_ENABLED;
			but[2].fsStyle = TBSTYLE_BUTTON;
switch(mes)
{
case WM_INITDIALOG:
		hEdit1=GetDlgItem(hdWnd,IDC_EDIT1);
		hEdit2=GetDlgItem(hdWnd,IDC_EDIT2);
		hEdit3=GetDlgItem(hdWnd,IDC_EDIT3);
		hButton1=GetDlgItem(hdWnd,ID_SAVEHASH);
		hButton2=GetDlgItem(hdWnd,ID_HASH);

		SendDlgItemMessage(hdWnd,IDC_RADIO1,BM_SETCHECK,1,0);
		t=1;

		strcpy (lf.lfFaceName, "Algerian");
		lf.lfHeight = 20;
		font = CreateFontIndirect(&lf);
		SendDlgItemMessage(hdWnd, IDC_EDIT3, WM_SETFONT, (WPARAM)font, MAKELPARAM(TRUE,0));

		strcpy (lf.lfFaceName, "Courier New");
		lf.lfHeight = 18;
		lf.lfItalic=1;
		lf.lfCharSet=RUSSIAN_CHARSET;
		font = CreateFontIndirect(&lf);
		SendDlgItemMessage(hdWnd, IDC_EDIT2, WM_SETFONT, (WPARAM)font, MAKELPARAM(TRUE,0));

		strcpy (lf.lfFaceName, "Monotype Corsiva");
		lf.lfHeight = 18;
		lf.lfItalic=1;
		lf.lfCharSet=RUSSIAN_CHARSET;
		font = CreateFontIndirect(&lf);
		SendDlgItemMessage(hdWnd, IDC_EDIT1, WM_SETFONT, (WPARAM)font, MAKELPARAM(TRUE,0));

		EnableWindow(hButton1, FALSE);
		EnableWindow(hButton2, FALSE);

		SetWindowText(hEdit1,"File Name");
		SetWindowText(hEdit2,"Key");
		SetWindowText(hEdit3,"Hash-code");

		memset(&ofn, 0, sizeof(OPENFILENAME));
		ofn.lStructSize       = sizeof(OPENFILENAME);
		ofn.hwndOwner         = hWnd;
		ofn.lpstrFilter       = szFilter;
		ofn.nFilterIndex      = 1;
		ofn.lpstrFile         = szFileAll;
		ofn.nMaxFile          = sizeof(szFileAll);
		ofn.lpstrFileTitle    = szFileOne;
```

```
		ofn.nMaxFileTitle     = sizeof(szFileOne);
		ofn.lpstrInitialDir   = NULL;


	hToolBar=CreateToolbarEx(hdWnd,WS_CHILD|WS_VISIBLE|WS_DLGFRAME|TBSTYLE_TOOLTIPS,ID_TOOLBAR1,0,HI
NST_COMMCTRL,IDB_STD_SMALL_COLOR,but,3,0,0,0,0,sizeof(TBBUTTON));

		hStatus=CreateStatusWindow(WS_CHILD|WS_VISIBLE," ",hdWnd,120);
		parts[0]=50; parts[1]=150; parts[2]=230; parts[3]=380; parts[4]=440; parts[5]=570; parts[6]=660; parts[7]=820;
		SendMessage(hStatus,SB_SETPARTS,8,(LPARAM)parts);
		SendMessage(hStatus,SB_SETTEXT,0,(LONG)"Hash:");
		SendMessage(hStatus,SB_SETTEXT,2,(LONG)"Size (byte):");
		SendMessage(hStatus,SB_SETTEXT,4,(LONG)"Time (s):");
		SendMessage(hStatus,SB_SETTEXT,6,(LONG)"Speed (b/s):");
		InvalidateRect(hdWnd,NULL,TRUE);

		for (int i=0; i<4; i++) {  _ui64toa_s(Key[i],buff,75,16); strcat(keybuff1,buff); strcat(keybuff1,"-"); }
		for (int i=0; i<8; i++) { _ui64toa_s(Key2[i],buff,75,16); strcat(keybuff2,buff); strcat(keybuff2,"-"); }
		for (int i=0; i<8; i++) { _ui64toa_s(Key3[i],buff,75,16); strcat(keybuff3,buff); strcat(keybuff3,"-"); }
		for (int i=0; i<4; i++) { _ui64toa_s(Key4[i],buff,75,16); strcat(keybuff4,buff); strcat(keybuff4,"-"); }
		SetWindowText(hEdit2,keybuff1);
		SendMessage(hStatus,SB_SETTEXT,1,(LONG)"Кватерніон 256");
	return 0;

	case WM_NOTIFY:
		LPTOOLTIPTEXT Tstr;
		Tstr = (LPTOOLTIPTEXT)lParam;
		if (Tstr->hdr.code!=TTN_NEEDTEXT) return 0;
		switch (Tstr->hdr.idFrom)
		{	case ID_OPENFILE:
				Tstr->lpszText="Відкрити файл";
				break;
			case ID_SAVEFILE:
				Tstr->lpszText="Вказати шлях для збереження файлу";
				break;
			case ID_MYHELP:
				Tstr->lpszText="Інформація про програму";
				break;
		}
		return 0;

	case WM_COMMAND:
		switch(LOWORD(wParam))
		{
			case IDC_RADIO1:
				type=1;
				SendMessage(hStatus,SB_SETTEXT,1,(LONG)"Кватерніон 256");

				SetWindowText(hEdit2,keybuff1);
				hashbuff[0]=0;
				EnableWindow(hButton2, TRUE);
				return 1;

			case IDC_RADIO4:
				type=4;
				SendMessage(hStatus,SB_SETTEXT,1,(LONG)"Кватерніон 128");

				SetWindowText(hEdit2,keybuff4);
				hashbuff[0]=0;
				EnableWindow(hButton2, TRUE);
				return 1;

			case ID_OPENFILE:
				ofn.Flags=OFN_ALLOWMULTISELECT|OFN_EXPLORER|OFN_CREATEPROMPT;
				ofn.lpstrTitle="Панель відкриття файлів";
				szFileOne[0]='\0'; szFileAll[0]='\0';
				if (GetOpenFileName(&ofn))
				{
					for (int i=0;i<256;i++)
					{
						if (szFileAll[i]=='\0' && szFileAll[i+1]=='\0') break;
						if (szFileAll[i]=='\0') szFileAll[i]=';';
					}
```

```
                }
                if((filer = fopen(szFileAll, "rb+"))==NULL)
                {
                        MessageBox(hWnd, "Файл не вдалось відкрити!","Error",
MB_OK|MB_ICONHAND);
                        puts("Error");
                }
                else
                {
                        EnableWindow(hButton2, TRUE);
                        EnableWindow(hButton1, FALSE);
                        fseek(filer,0,SEEK_END);
                        size = ftell(filer);
                        fseek(filer,0,0);

                        _ui64toa_s(size,st,65,10);
                        SendMessage(hStatus,SB_SETTEXT,3,(LONG) st);
                        SetWindowText(hEdit1,szFileOne);
                        fclose(filer);
                }
                break;


        case ID_MYHELP:
                DialogBox(hInst,MAKEINTRESOURCE(IDD_DIALOG6),hWnd,DlgProc6);
                break;

        case ID_SAVEFILE:
                DialogBox(hInst,MAKEINTRESOURCE(IDD_DIALOG1),hWnd,DlgProc1);
                break;

        case ID_SAVEHASH:
                files=fopen(adressf, "a");
                if((files = fopen(adressf, "a"))==NULL)
                {
                        MessageBox(hWnd, "Файл не вдалось відкрити!","Error",
MB_OK|MB_ICONHAND);
                        puts("Error");
                }
                else
                {
                        fputs(szFileOne,files);
                        fputs("***",files);
                        GetDlgItemText(hdWnd,IDC_EDIT3,hashbuff,150);
                        fputs(hashbuff,files);
                        fputs("******",files);
                        fprintf(files, "%s\n", " ");
                        fclose(files);
                }
                EnableWindow(hButton1, FALSE);
                EnableWindow(hButton2, FALSE);
                break;

        case ID_HASH:
                SetWindowText(hEdit3,"\0");
                if((filer = fopen(szFileAll, "rb"))==NULL)
                {
                        MessageBox(hWnd, "Файл не вдалось відкрити!","Error",
MB_OK|MB_ICONHAND);
                        puts("Error");
                }
                else
                {
                        if(type!=0)
                        {
                        switch(type)
                        {
                        case 1:
                                {
                                        start=clock();
                                        KvatHash256(filer);
                                        for (int i=0; i<4; i++)
                                        {
```

```
                                        _ui64toa_s(s[i],buff,65,16);
                                        if (strlen(buff)<16)
                                        for (int g=0; g<(16-strlen(buff)); g++)
                                                {
                                                        strcat(hashbuff,"0");
                                                }
                                        strcat(hashbuff,buff);
                                        strcat(hashbuff,"-");
                                }
                        }
                        break;
        case 2:
                        {
                                start=clock();
                                OktavHash256(filer);
                                for (int i=0; i<8; i++)
                                {
                                        _ui64toa_s(s2[i],buff,65,16);
                                        if (strlen(buff)<8)
                                        for (int g=0; g<(8-strlen(buff)); g++)
                                                {
                                                        strcat(hashbuff,"0");
                                                }
                                        strcat(hashbuff,buff);
                                        strcat(hashbuff,"-");
                                }
                        }
                        break;

        case 3:
                        {
                                start=clock();
                                OktavHash512(filer);
                                for (int i=0; i<8; i++)
                                {
                                        _ui64toa_s(s3[i],buff,65,16);
                                        if (strlen(buff)<16)
                                        for (int g=0; g<(16-strlen(buff)); g++)
                                                {
                                                        strcat(hashbuff,"0");
                                                }
                                        strcat(hashbuff,buff);
                                        strcat(hashbuff,"-");
                                }
                        }
                        break;

        case 4:
                        {
                                start=clock();
                                KvatHash128(filer);
                                for (int i=0; i<4; i++)
                                {
                                        _ui64toa_s(s4[i],buff,65,16);
                                        if (strlen(buff)<8)
                                        for (int g=0; g<(8-strlen(buff)); g++)
                                                {
                                                        strcat(hashbuff,"0");
                                                }
                                        strcat(hashbuff,buff);
                                        strcat(hashbuff,"-");
                                }
                        }
                        break;
        }

        end=clock();
        t=((float)end-start)/CLK_TCK;
        gcvt(t,20,time);
        SendMessage(hStatus,SB_SETTEXT,5,(LONG)time);
        speed=size/t;
        gcvt(speed,20,speeds);
        SendMessage(hStatus,SB_SETTEXT,7,(LONG)speeds);
```

```
                                        SetWindowText(hEdit3,hashbuff);
                                        EnableWindow(hButton1, TRUE);

                                        fclose(filer);
                                        s[4]=(0, 0, 0, 0);
                                        s2[8]=(0, 0, 0, 0, 0, 0, 0, 0);
                                        s3[8]=(0, 0, 0, 0, 0, 0, 0, 0);
                                        s4[4]=(0, 0, 0, 0);
                                        k=0;
                                        hashbuff[0]=0;

                                        }
                                }
                                break;

                        case ID_CLEAN:
                                SetWindowText(hEdit1,"\0");
                                SetWindowText(hEdit2,"\0");
                                SetWindowText(hEdit3,"\0");
                                SendMessage(hStatus,SB_SETTEXT,1,(LONG)" ");
                                SendMessage(hStatus,SB_SETTEXT,3,(LONG)" ");
                                SendMessage(hStatus,SB_SETTEXT,5,(LONG)" ");
                                SendMessage(hStatus,SB_SETTEXT,7,(LONG)" ");
                                EnableWindow(hButton1, FALSE);
                                EnableWindow(hButton2, FALSE);
                                break;

                        case ID_EXIT:
                                EndDialog(hdWnd,0);
                                return 1;
                }
                return 1;
  }
  return 0;
}

unsigned long long CharToNum (char *a1, int numsize)
{
        int a=0, b=0;
        unsigned long long a1n=0;
        for(int i=0; i<numsize; i++)
                {
                        a=a1[i];
                        if (a<0)
                        {
                                b=256+a1[i];
                        }
                        else if (a==0)
                        {
                                b=170;
                        }
                        else
                        {
                                b=a1[i];
                        }
                        a1n=a1n+NumOnNumMod64(b,pow(256.0,i));
                }
        return a1n;
}

unsigned long long NumPlusNum64 (unsigned long long a, unsigned long long b)
{
        unsigned long long y=0, y1=0, y2=0, y11=0, y12=0, y21=0, y22=0;
        y11=a>>32;
        y21=b>>32;
        y1=y11+y21;
        y12=a&0x00000000ffffffff;
        y22=b&0x00000000ffffffff;
        y2=y12+y22;

        if (y1>0x00000000ffffffff)
        {
```

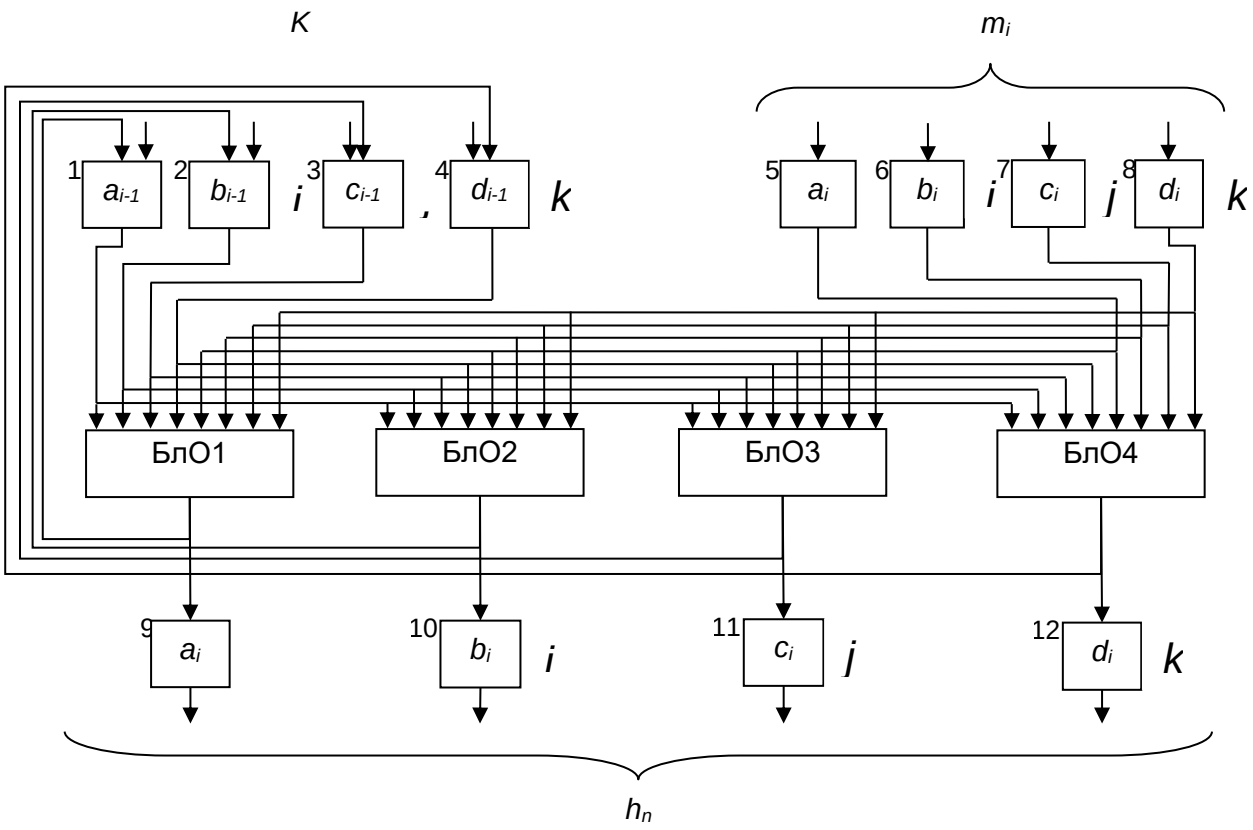
```
                y1=fmod(y1,4294967295.0);
                y1=y1<<32;
                y=y1+y2;
        }
        else
        {
                y=a+b;
        }
        return y;
}

unsigned long long NumOnNumMod64 (unsigned long long x, unsigned long long z)
{
        unsigned long long a=0, b=0, c=0, d=0;
        unsigned long long x1=0, x2=0, x3=0, x4=0;
        unsigned long long x11=0, x21=0, x31=0;
        unsigned long long x12=0, x22=0, x32=0;
        unsigned long long y=0, y1=0, y2=0;

        a=x>>32;
        b=x&0x00000000ffffffff;
        c=z>>32;
        d=z&0x00000000ffffffff;

        x4=a*c;
        x3=b*c;
        x2=a*d;
        x1=b*d;

        x11=x1>>32;
        x12=x1&0x00000000ffffffff;
        x21=x2>>32;
        x22=x2&0x00000000ffffffff;
        x31=x3>>32;
        x32=x3&0x00000000ffffffff;

        y1=x22+x32;
        if (y1>0x00000000ffffffff)
        {
                x21=x21+1;
                y1=fmod(y1,4294967295.0);
        }
        y1=y1+x11;
        if (y1>0x00000000ffffffff)
        {
                x21=x21+1;
                y1=fmod(y1,4294967295.0);
        }
        y1=y1<<32;

        y1=y1+x12;
        y2=x4+x21+x31;

        y=NumPlusNum64(y1,y2);
        return y;
}

unsigned long long NumOnNumMod32 (unsigned long long a, unsigned long long b)
{
        unsigned long long z=0, y=0, y1=0, y2=0;
        z=a*b;
        if (z==0)
                y=a+b;
        else
        {
                y1=z>>32;
                y2=z&0x00000000ffffffff;
                y=y1+y2;
                y=fmod(y,4294967295.0);
        }
        return y;
}
```

```
void KvatHash128 (FILE *filer)
{
	static unsigned long long size=0;
	char *ars[4];
	for (int i=0; i<4; i++)
	{
		ars[i]=new char[5];
		//ars[i][5]='\0';
	}
	unsigned long long a1[4]={0,0,0,0};
	unsigned long long a2[4]={0,0,0,0};
	int j=0;

	fseek(filer,0,SEEK_END);
	size = ftell(filer);
	fseek(filer,0,0);

	while(size>0)
	{
		if(size>=16)
		{
			for (int i=0; i<4; i++)
				fread(ars[i], sizeof(char), 4, filer);
			size=size-16;
		}
		else if (size>=4)
		{
			while(size>=4)
			{
				fread(ars[j], sizeof(char), 4, filer);
				size=size-4;
				j++;
			}
		}
		else if (size<4)
		{
			fread(ars[j], sizeof(char), size, filer);
			size=0;
		}
		for (int i=0; i<4; i++)
		{
			a1[i]=CharToNum(ars[i],4);
		}

		if (k==0)
		{
			for (int i=0; i<4; i++)
			{
				a2[i]=Key4[i];
			}
			k++;
		}
		else
		{
			for (int i=0; i<4; i++)
			{
				a2[i]=s4[i];
			}
		}

		s4[0]=fmod((NumOnNumMod32(a1[0],a2[0])-NumOnNumMod32(a1[1],a2[1])-NumOnNumMod32(a1[2],a2[2])-
NumOnNumMod32(a1[3],a2[3])),4294967295.0);
		s4[1]=fmod((NumOnNumMod32(a1[0],a2[1])+NumOnNumMod32(a1[1],a2[0])+NumOnNumMod32(a1[2],a2[3])-
NumOnNumMod32(a1[3],a2[2])),4294967295.0);
		s4[2]=fmod((NumOnNumMod32(a1[0],a2[2])+NumOnNumMod32(a1[2],a2[0])+NumOnNumMod32(a1[1],a2[3])-
NumOnNumMod32(a1[3],a2[1])),4294967295.0);
		s4[3]=fmod((NumOnNumMod32(a1[0],a2[3])+NumOnNumMod32(a1[3],a2[0])+NumOnNumMod32(a1[1],a2[2])-
NumOnNumMod32(a1[2],a2[1])),4294967295.0);
	}
}

void KvatHash256 (FILE *filer)
{
```

```
	static unsigned long long size=0;
	char *ars[4];
	for (int i=0; i<4; i++)
	{
		ars[i]=new char[9];
	}
	unsigned long long a1[4]={0,0,0,0};
	unsigned long long a2[4]={0,0,0,0};
	int j=0;

	fseek(filer,0,SEEK_END);
	size = ftell(filer);
	fseek(filer,0,0);

	while(size>0)
	{
		if(size>=32)
		{
			for (int i=0; i<4; i++)
				fread(ars[i], sizeof(char), 8, filer);
			size=size-32;
		}
		else if (size>=8)
		{
			while(size>=8)
			{
				fread(ars[j], sizeof(char), 8, filer);
				size=size-8;
				j++;
			}
		}
		else if (size<8)
		{
			fread(ars[j], sizeof(char), size, filer);
			size=0;
		}
		for (int i=0; i<4; i++)
		{
			a1[i]=CharToNum(ars[i],8);
		}

		if (k==0)
		{
			for (int i=0; i<4; i++)
			{
				a2[i]=Key[i];
			}
			k++;
		}
		else
		{
			for (int i=0; i<4; i++)
			{
				a2[i]=s[i];
			}
		}

		s[0]=NumOnNumMod64(a1[0],a2[0])-NumOnNumMod64(a1[1],a2[1])-NumOnNumMod64(a1[2],a2[2])-
NumOnNumMod64(a1[3],a2[3]);
		s[1]=NumOnNumMod64(a1[0],a2[1])+NumOnNumMod64(a1[1],a2[0])+NumOnNumMod64(a1[2],a2[3])-
NumOnNumMod64(a1[3],a2[2]);
		s[2]=NumOnNumMod64(a1[0],a2[2])+NumOnNumMod64(a1[2],a2[0])+NumOnNumMod64(a1[1],a2[3])-
NumOnNumMod64(a1[3],a2[1]);
		s[3]=NumOnNumMod64(a1[0],a2[3])+NumOnNumMod64(a1[3],a2[0])+NumOnNumMod64(a1[1],a2[2])-
NumOnNumMod64(a1[2],a2[1]);
	}
}
```

ДОДАТОК В
ІЛЮСТРАТИВНА ЧАСТИНА

СХЕМА ОДНІЄЇ ІТЕРАЦІЇ АЛГОРИТМУ ГЕШУВАННЯ



АЛГОРИТМ РОБОТИ ПРОГРАМИ

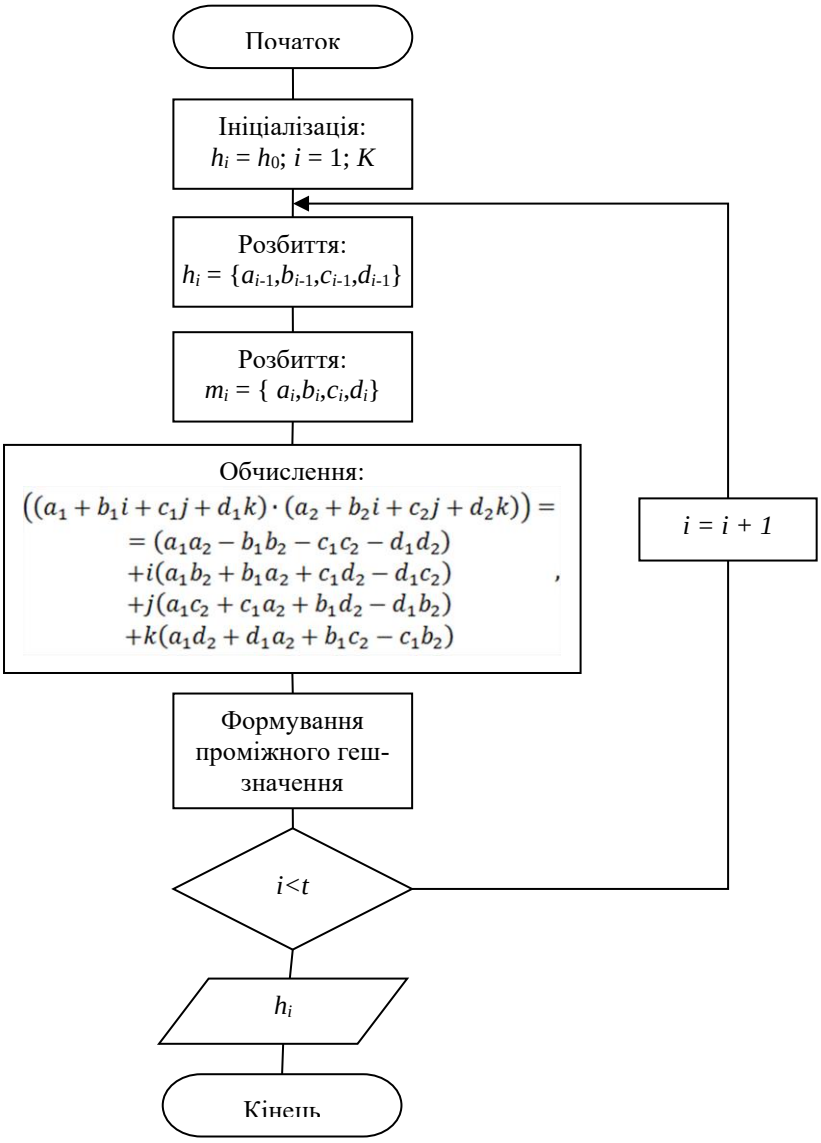
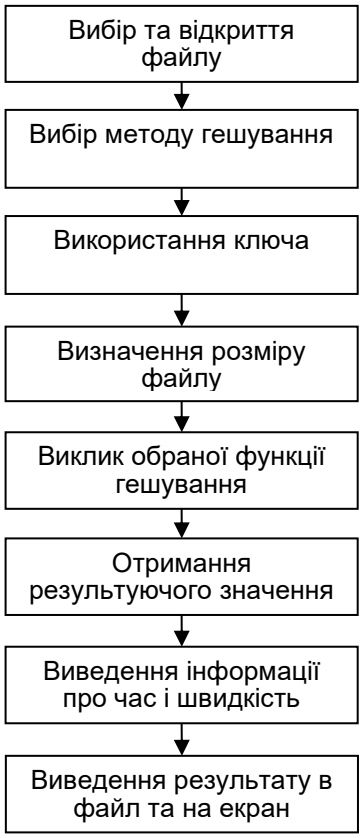
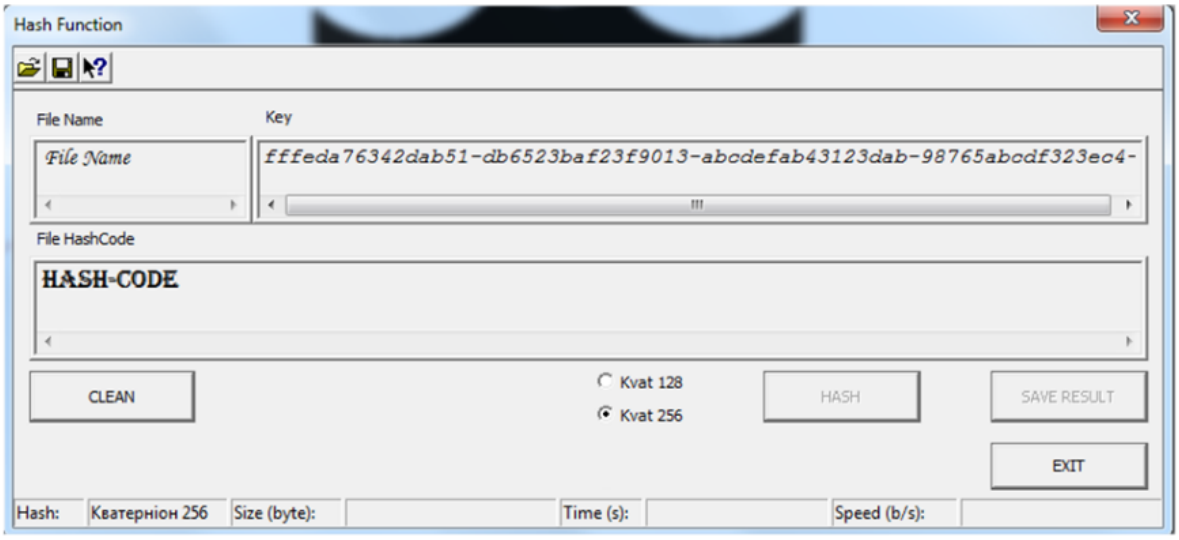


СХЕМА ФУНКЦІОНУВАННЯ ПРОГРАМИ



ОСНОВНЕ ДІАЛОГОВЕ ВІКНО ІНТЕРФЕЙСУ ПРОГРАМИ



ПОРІВНЯЛЬНІ ОЦІНКИ

Геш	Обчислювальна складність ітерації (операцій)
Keccak	320
Skein	243
Кватерніон	44

Геш	Коефіцієнт пришвидшення 1 та 4 процесори
Keccak	7,3 / 29,2
Skein	5,5 / 22

ВИГЛЯД РЕЗУЛЬТАТУ ГЕШУВАННЯ У ФАЙЛІ