Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

Бакалаврська дипломна робота на тему:

«Засіб гешування даних на основі кватерніонів»

Виконав: студент 2-го курсу групи 1БС-21мс
спеціальності 125 – Кібербезпека

_____Куцурук В. В.

Керівник: завідувач каф. ЗІ, д.т.н., професор

_____Лужецький В. А.
«19» червня 2023 р.

Рецензент: доцент кафедри ПЗ, к.т.н.

_____ Хошаба О.М
«19» червня 2023 р.

Допущено до захисту
Завідувач кафедри ЗІ:
д.т.н., проф.
_____Лужецький В. А.
«19» червня 2023 р.

Вінниця ВНТУ – 2023 рік

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

ЗАТВЕРДЖУЮ
Зав. кафедри ЗІ д.т.н., проф.
_____ В. А. Лужецький
«20» березня 2023 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Куцуруку Віталію Вікторовичу

1. Тема роботи: «Засіб гешування даних на основі кватерніонів»
   Керівник роботи: Лужецький Володимир Андрійович, зав. кафедри ЗІ д.т.н., затверджені наказом ректора ВНТУ від 20 березня 2023 року № 67
2. Строк подання студентом роботи 19.06.2023 р.
3. Вихідні дані до роботи:
   - Обсяг даних, що підлягають гешуванню – не більше 2^{64} біт.
   - Довжина геш значення – 256.
   - Зв'язування даних на основі моделі кватерніона
4. Зміст розрахунково-пояснювальної записки: Вступ. Огляд підходів до побудови геш-функцій. Розроблення методу гешування даних. Розроблення алгоритму гешування даних. Розроблення програмного засобу для гешування даних. Тестування програмного засобу. Висновки. Список використаних джерел. Додатки.
5. Перелік ілюстративного матеріалу: Підходи до побудови геш-функцій (плакат А4). Структура геш-функцій (плакат А4). Метод гешування даних (плакат А4). Метод побудови геш-функцій (плакат А4). Схема роботи програми (плакат А4). Результати тестування програмного засобу (плакат А4).

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Лужецький В. А., д.т.н., проф., зав кафедри ЗІ	20.03.23	19.06.23
2	Лужецький В. А., д.т.н., проф., зав кафедри ЗІ	20.03.23	19.06.23
3	Лужецький В. А., д.т.н., проф., зав кафедри ЗІ	20.03.23	19.06.23

7. Дата видачі завдання 21 березня 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів Дипломної кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання. Вступ	20.03.23-26.03.23	
2	Пошук літературних джерел за темою БДР	27.03.23-09.04.23	
3	Аналіз літературних джерел за напрямком бакалаврської кваліфікаційної роботи	10.04.23-23.04.23	
4	Розробка рішень, моделей, алгоритмів	24.04.23-21.05.23	
5	Практична реалізація, моделювання, експериментування, результати	22.05.23-24.05.23	
6	Оформлення пояснювальної записки, підготовка ілюстративного матеріалу	25.05.23-26.05.23	
7	Попередній захист БДР	27.05.23-29.05.23	
8	Виправлення зауважень, перевірка БДР на плагіат	28.05.23-14.06.23	
9	Представлення БДР до захисту, рецензування	15.06.23-19.06.23	
10	Захист БДР	20.06.23-23.06.23	

Студент _____ Куцурук В.І

Керівник роботи _____ Лужецький В. А

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 56 сторінок формату А4, на яких є 25 рисунків, список використаних джерел містить 18 найменувань.

У бакалаврській диломній роботі розглядається питання засобу гешування даних на основі кватерніонів та розробка програми для їх обчислення. Основний напрямок даної роботи це проектування, аналіз та розробка методу гешування, що буде ефективно приймати та гешувати дані користувача. Під час виконання бакалаврської кваліфікаційної роботи було проаналізовано аналоги методів гешування, що розробляється, виявлено їх недоліки, з урахуванням яких створено власний метод гешування на основі кватерніонів. Розроблено метод гешування та алгоритми для множення кватерніонів що надходять.

Для розробки програмного засобу використано мову програмування Python та інтегроване середовище розробки PyCharm.

Ключові слова: геш-функція, методи гешування, кватерніони, множення кватерніонів.

ABSTRACT

The bachelor's thesis consists of 56 A4 pages with 25 figures and a list of references containing 18 titles.

The bachelor's thesis deals with the issue of a data hashing tool based on quaternions and the development of a program for their calculation. The main focus of this work is the design, analysis and development of a hashing method that will efficiently accept and hash user data. In the course of the bachelor's thesis, analogues of the developed hashing methods were analysed, their shortcomings were identified, and the authors created their own quaternion-based hashing method. A hashing method and algorithms for multiplying incoming quaternions were developed.

The Python programming language and the PyCharm integrated development environment were used to develop the software tool.

Keywords: hash function, hashing methods, quaternions, quaternion multiplication.

ЗМІСТ

ВСТУП

У сучасному цифровому світі обмін і збереження великого обсягу даних стало невід'ємною складовою багатьох сфер нашого життя. Однак, разом зі зростанням обсягу даних з'явилися також проблеми забезпечення їх безпеки та конфіденційності. Особливо важливою є необхідність захисту особистих даних, комерційної інформації, медичних записів та інших конфіденційних даних від несанкціонованого доступу.

Одним із підходів до забезпечення безпеки даних є їх гешування. Гешування є процесом перетворення вхідного повідомлення (дані) фіксованої довжини, яке називається гешем, за допомогою криптографічно стійкої геш-функції. Геші мають декілька важливих властивостей, зокрема, вони є незворотними і нечутливими до навіть найменших змін у вихідних даних. Це дозволяє використовувати Геші для перевірки цілісності даних, порівняння даних та швидкого пошуку.

Однак, у зв'язку зі зростанням обчислювальної потужності сучасних комп'ютерів, стало відомо, що деякі геш-функції можуть бути піддані атакам з використанням методу підбору (brute force) або застосування криптоаналітичних алгоритмів. Тому дедалі більше уваги приділяється розробці нових методів гешування, які були б стійкими до таких атак.

Один із цікавих підходів до гешування даних полягає у використанні кватерніонів. Кватерніони є розширенням комплексних чисел і мають свої особливі властивості, які знаходять застосування в різних галузях, включаючи комп'ютерну графіку, комп'ютерне бачення та криптографію. Використання кватерніонів для гешування даних відкриває нові можливості у плані стійкості до атак, швидкості обчислень та збереження даних.

Бакалаврська кваліфікаційна робота включає аналіз існуючих методів гешування, вивчення основ кватерніонної алгебри, розробку методу та алгоритму гешування з використанням кватерніонів та експериментальне порівняння його ефективності з існуючими методами гешування.

Метою бакалаврської кваліфікаційної роботи є пришвидшення процесу гешування даних зі збереженням існуючого рівня криптографічної стійкості геш-функцій. Розробка методу гешування даних з використанням кватерніонів.

Для досягнення мети необхідно:

- проаналізувати відомі методи створення геш-функцій;
- розробити метод гешування даних на основі кавтерніонів;
- розробити програмний засіб для реалізації методу гешування.

Наукова новизна роботи полягає у використанні моделі кватерніона для роботи зі складовими даних у процесі гешування. Практичну цінність роботи складає програмний засіб для пришвидшеного гешування даних.

1 ОГЛЯД ПІДХОДІВ ДО ПОБУДОВИ ГЕШ-ФУНКЦІЙ

1.1 Підходи до побудови геш-функцій

Геш-функція - це функція, яка отримує на вхід будь-який вхідний рядок довільної довжини і перетворює його в вихідний рядок, обмежений за обсягом і з фіксованою довжиною, який зазвичай називається геш-значенням. Геш-функції застосовуються в багатьох галузях, включаючи криптографію, пошук інформації та баз даних [1].

Значення функції гешування генерується функцією H виду:

$$h = H(M)$$

, де:

M – повідомлення фіксованої довжини,

H(M) – значення функції гешування довільної довжини.

Функції гешування H повинні мати такі властивості:

- На виході функції має бути обов’язково значення з фіксованою довжиною;
- може бути застосована до блоку даних з будь-якою довжиною;
- значення функції H(x) повинно обчислювати відносно легко і застосований алгоритм обчислення повинний бути практичним;
- функція гешування повинна бути одностороння. В такому випадку неможливо обчислити x, для якого H(x) = h;
- функція гешування повинна мати стійкість до пошуку першого прообразу (колізії першого ряду). Для будь-якого блоку x повинно неможливо обчислити y, для якого H(x) = H(y);
- також геш-функції повинні мати сильну опірну колізію або стійкість до колізій.

Ефективність геш-функції – це імовірність (P) того, що з появою помилки в даних значення функції гешування залишить попереднім.

Якщо значення геш-функції має n-розрядів, то

$P = 2^{-n}$ для довільного випадку даних

$P < 2^{-n}$ для форматованих даних

Ефективність геш-функції залежить від декількох факторів, таких як швидкодія геш-функції, рівень випадковості згенерованих геш-значень та стійкість до злому. Швидкодія геш-функції є важливим фактором, оскільки деякі застосування можуть вимагати швидкого гешування великих обсягів даних. Швидкодія зазвичай залежить від конкретної реалізації геш-функції та може бути різною для різних алгоритмів [1].

Геш-функції зазнають таких атак:

– Атака методом «Груба сила» або атака на знаходження випадкового прообразу, може бути виконана для знаходження прообразу за заданим геш-значенням або для знаходження прообразу, що дає задане геш-значення;
– атаки методом парадоксу «Дня народження», виконується для знаходження двох різних повідомлень з однаковим геш-значенням;
– атака з корекцією блоку, використовується у випадку коли порушник має повідомлення і хоче змінити в ньому один або більше блоків без зміни геш-значення;
– атака фіксованою точкою може застосовуватися за умови, що циклова функція f має одну або декілька фіксованих точок.

Загальний принцип побудови геш-функції, що на рис. 1.1 полягає в тому, щоб приймати вхідні дані будь-якої довжини та повертати геш-значення фіксованої довжини. Геш-функція призначена для забезпечення надійного збереження інформації, перевірки цілісності даних та шифрування [2].

Дані, які надходять (повідомлення, файл) розглядаються як послідовні m бітові блоки типу M_0, M_1, …. , M_{L-1}. Усі значення отриманні на виході геш-функції перед цим обробляються послідовно блок за блоком. Проміжні та остаточне значення функції мають бітове представлення.

Однією з головних вимог до геш-функції є відсутність можливості відновлення вихідних даних за геш-значенням. Це забезпечує криптографічну стійкість та випадковості функції гешування. Крім того, геш-функції повинні мати високу

швидкість обчислення, щоб бути ефективними у різних застосуваннях. На рис. 1.2 наведено конструкцію Меркля-Демгарда [2].

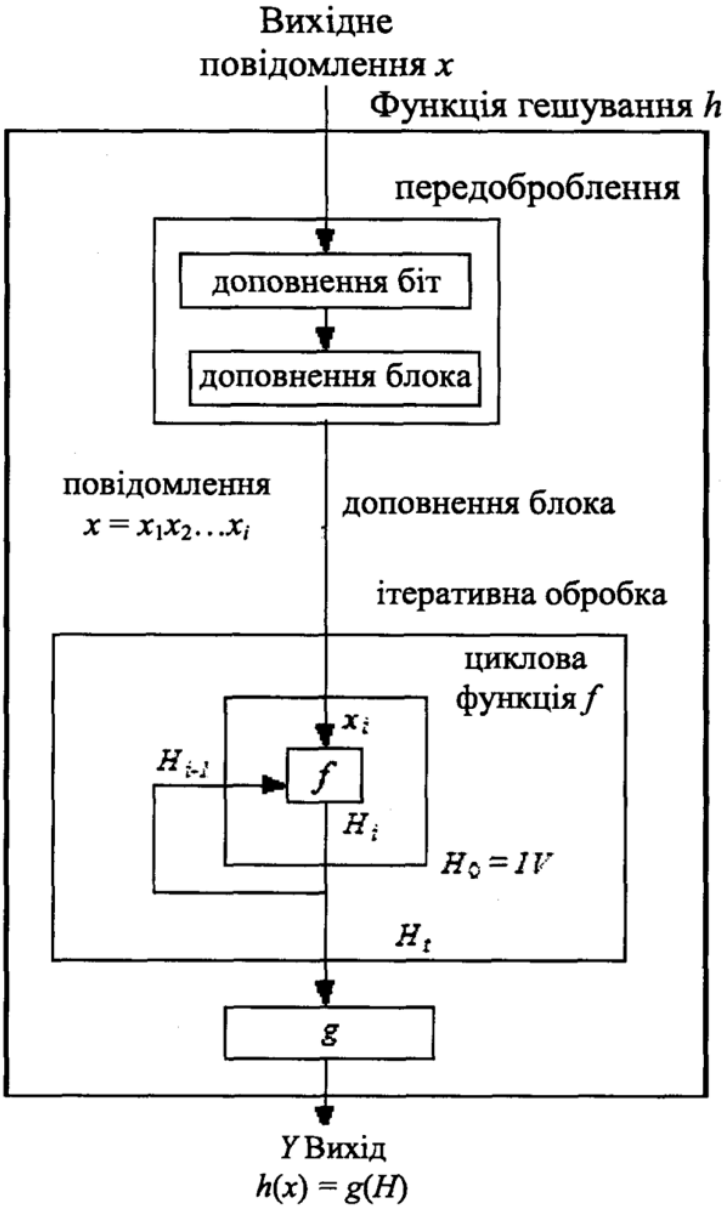


Рисунок 1.1 – Ітеративна модель функції гешування

При побудові якісних геш-функцій дуже важливою є інформація про розподіл ключів даної функції. Правильним підходом при побудові є підбір геш-функції так, щоб функція не корелювала із закономірностями, яким можуть підкорятися існуючі дані. Іноді до геш-функцій можуть висувати більш суворі вимоги, ніж за простого рівномірного гешування.

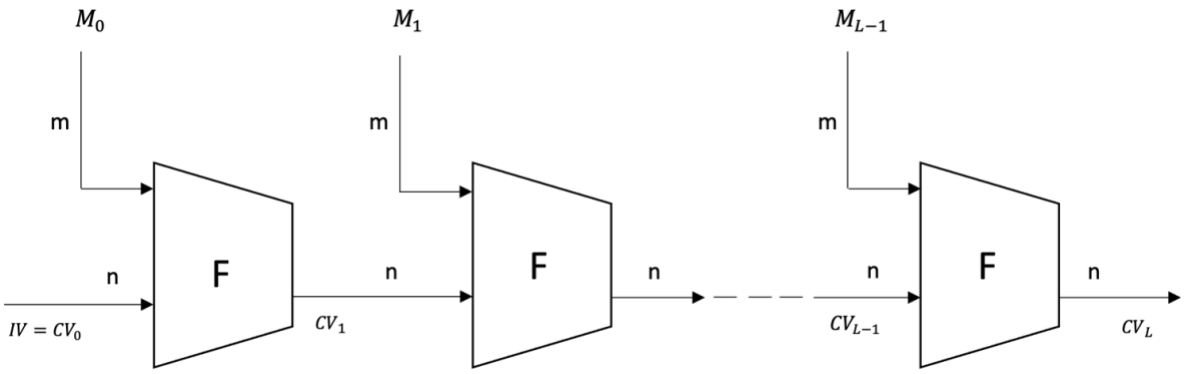


Рисунок 1.2 – Конструкція Меркля-Демгарда

Використані такі позначення:

— IV – початкове значення

— L – кількість блоків що вводиться

— n – довжина геш-коду

— m – довжина блоку що вводиться

— F – алгоритм ущільнення

— M_i – і-й блок, що вводиться

— CV_i – проміжне геш-значення

В більшості геш-функцій на основі блокових шифрів вхідний текст розбивається на блоки по 512 бітів (64 байти). Якщо вхідний текст не є кратним 512 бітам, його потрібно доповнити до більшого кратного значення за допомогою спеціальних алгоритмів доповнення [3].

Найбільш поширеними алгоритмами доповнення є доповнення нулями та доповнення бітів. В алгоритмі доповнення нулями до вхідного тексту додається додаткові нульові біти, щоб його довжина стала кратною 512 бітам. В алгоритмі доповнення бітів до вхідного тексту додається один біт зі значенням 1, за яким слідують нульові біти, аж доки довжина тексту не буде кратною 512 бітам.

Щодо алгоритмів шифрування, то вони можуть бути різними, залежно від конкретної функції гешування. Наприклад, для SHA-1 використовується комбінація різних операцій з бітами, таких як зсуви, логічні операції та операції з криптографічними ключами. Для SHA-2, який включає SHA-256, SHA-384 та SHA-

512, використовуються більш складні алгоритми, такі як блочні шифри, які включають в себе операції з замінами та перестановками бітів.

Значення біта 1 зазвичай використовується для відмітки кінця повідомлення, щоб не допускати дублювання і забезпечити унікальність геш-коду для кожного повідомлення [3].

Доповнення нулями, або zero-padding - це метод додавання нульових бітів до повідомлення таким чином, щоб його довжина стала кратною заданому розміру блоку (у випадку функцій гешування, зазвичай, 512 бітів). Це необхідно, оскільки функції гешування приймають лише повідомлення фіксованої довжини, і якщо повідомлення не кратне розміру блоку, його необхідно доповнити до необхідної довжини.

Доповнення нулями можна виконати, наприклад, шляхом додавання одного біту зі значенням "1", за яким слідують нульові біти до досягнення необхідної довжини, або ж додавання відразу всіх нульових бітів до досягнення необхідної довжини.

Важливо зазначити, що при доповненні нулями до повідомлення може виникнути проблема з безпекою, оскільки зловмисник може змінити доповнення, що призведе до зміни значення геш-функції. Тому деякі алгоритми гешування, такі як SHA-3, використовують інші методи доповнення, які не створюють таких проблем з безпекою.

Доповнення одиничним бітом та нулями (one-padding) є одним з методів доповнення повідомлення в алгоритмах гешування, таких як SHA-1 та MD5. Цей метод полягає у додаванні одиничного біту до повідомлення, а потім заповнення решти блоку нулями до досягнення кратності 512 бітам.

Наприклад, якщо довжина повідомлення складає 456 бітів, то до повідомлення буде додано одиничний біт, а потім заповнено нулями до довжини 512 бітів (тобто додано 55 нульових бітів).

Цей метод забезпечує включення більшої кількості інформації з повідомлення в геш-функцію, тому що додавання одиничного біту може розширити множину можливих гешів. Однак, в порівнянні з доповненням нулями, цей метод вимагає

додаткового обчислювального зусилля для вставки одиничного біту і перевірки цілості повідомлення.

У доповненні бітами довжини повідомлення, до повідомлення додається спочатку бітове подання довжини повідомлення в бінарному форматі. Якщо довжина повідомлення складається з n біт, то додається n-бітне бінарне число, яке представляє довжину повідомлення.

Для прикладу, якщо довжина повідомлення становить 124 біти, то до повідомлення додається бітове подання числа 124 в бінарному форматі, тобто "01111100". Потім додаються стільки нульових бітів, скільки потрібно, щоб загальна довжина повідомлення стала кратною 512 бітам [3].

Існують такі підходи до побудови геш-функцій, зображені на рис. 1.3:

– На основі будь-якої складної математичної задачі. Дані математичні задачі можна поділити на два види, а саме на основі еліптичних кривих та на основі операційного піднесення до степеня;
– на основі алгоритмів блокового шифрування. Даний підхід побудови поділяється містить в собі режим зчеплення блоків зашифрованого тексту;
– розроблені з нуля.

Рисунок 1.3 – Підходи до побудови геш-функцій

1.2 Геш-функції на основі блокових шифрів

Побудова геш-функцій на основі блокових шифрів - це метод, що поєднує в собі блокові шифри та геш-функції, щоб забезпечити інтегритет повідомлення та захист від фальсифікації даних. Блочні шифри перетворюють блоки вхідних даних в блоки випадкових даних фіксованої довжини, які називаються геш-значеннями .

Один з найпоширеніших методів, який складається з фази гешування та фази побудови коду аутентифікації повідомлення. У фазі гешування вхідне повідомлення зберігається у буфері та гешується за допомогою обраної геш-функції. Далі, після додавання ключа та можливої додаткової інформації, геш розбивається на блоки та проходить через фазу, в якій застосовується блоковий шифр для створення коду аутентифікації повідомлення.

Інший метод - це побудова геш-функції на основі режиму шифрування відкритого тексту ECB. Повідомлення розбивається на блоки, які шифруються за допомогою блокового шифру, а потім об'єднуються та гешуються за допомогою обраної геш-функції [4].

Загалом, побудова геш-функцій на основі блокових шифрів є досить складним процесом, оскільки потрібно враховувати різні атаки та забезпечити відповідний рівень безпеки. Тому перед використанням геш-функції на основі блокових шифрів необхідно детально розглянути метод та виконати відповідні тестування на стійкість до атак. Загальні схеми ітерації геш-функцій на основі блокового шифрування мають вигляд дивитися рис. 1.4-1.7.

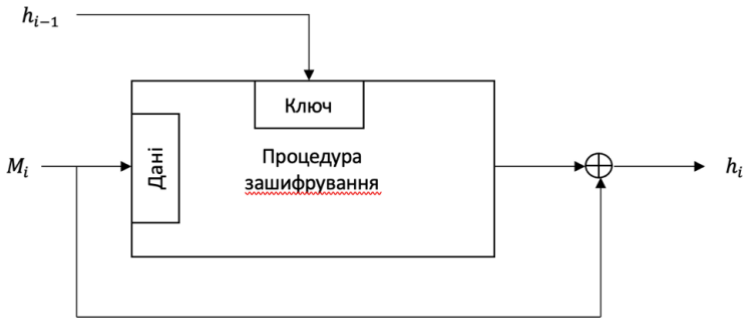


Рисунок 1.4 – Схема 1 ітерації безпечної геш-функції

$$h_i = E_{h_{i-1}}(M_i) \oplus M_i$$

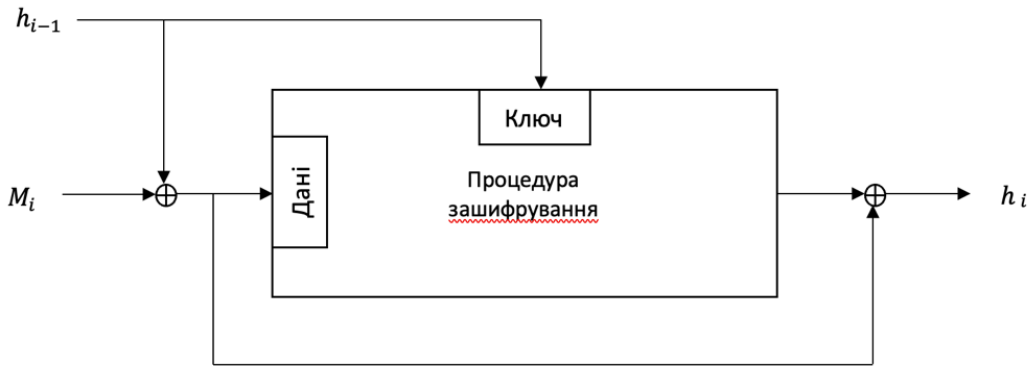


Рисунок 1.5 – Схема 2 ітерації безпечної геш-функції

$$h_i = E_{h_{i-1}}(M_i \oplus h_{i-1}) \oplus M_i \oplus h_{i-1}$$

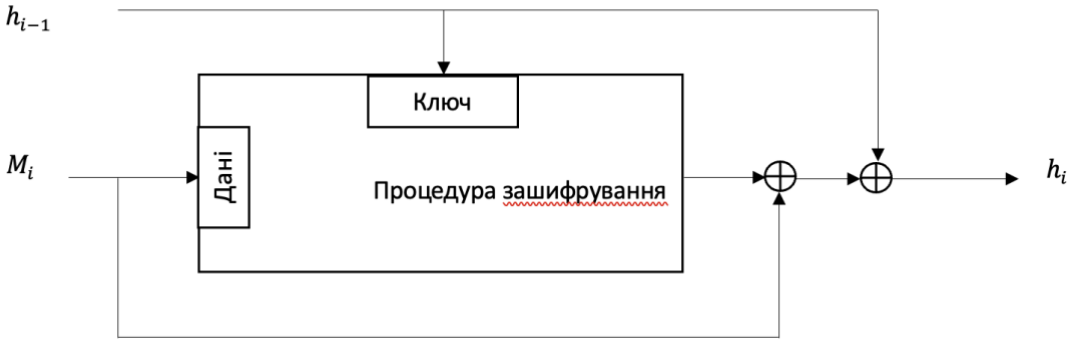


Рисунок 1.6 – Схема 3 ітерації безпечної геш-функції

$$h_i = E_{h_{i-1}}(M_i) \oplus h_{i-1}$$

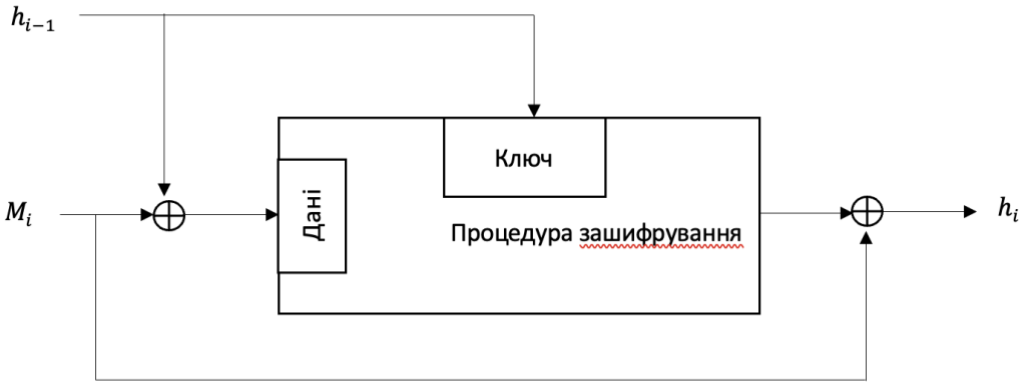


Рисунок 1.7 - Схема 4 ітерації безпечної геш-функції

$$h_i = E_{h_{i-1}}(M_i \oplus h_{i-1}) \oplus M_i$$

З практичної точки зору, побудова геш-функції на основі використання n-розрядного блокового шифру є ефективна програмна чи апаратна реалізація блокового шифру, що застосовується в системі захисту. Це забезпечує велику функціональність криптоалгоритмів і зниження витрат на їх розробку.

В ISO/IEC 10118-2 визначають 2 функції гешування:

– Функція гешування однократної довжини;
– функція гешування подвійної довжини.

ISO/IEC 10118-2 є міжнародним стандартом, який визначає методи гешування повідомлень з використанням криптографічних хеш-функцій. Цей стандарт описує три рівні безпеки, які відповідають різним розмірам вихідних значень геш-функцій.

ISO/IEC 10118-2 включає в себе такі геш-функції, як MD2, MD4, MD5, SHA-1, SHA-224, SHA-256, SHA-384 та SHA-512. Кожна з цих функцій має свої особливості, наприклад, розмір вихідного значення та спосіб обчислення [5].

Стандарт ISO/IEC 10118-2 також включає в себе вимоги до безпеки геш-функцій, такі як опірність до колізій та протиаварійність. Він також вказує на те, як використовувати геш-функції в різних криптографічних протоколах, таких як SSL, TLS, IPSec та інші.

Один з найважливіших аспектів стандарту ISO/IEC 10118-2 полягає в тому, що він рекомендує використовувати ключі з великою довжиною в бітовому виразі. Це забезпечує високий рівень безпеки при застосуванні геш-функцій для захисту від колізій та атак.

Загалом, стандарт ISO/IEC 10118-2 є важливим інструментом для забезпечення безпеки в криптографії та захисту від різних видів атак [5].

Циклова функція є блоковим шифром і характеризується такими основними параметрами:

– n - довжина вхідного блоку даних;
– k - довжина ключа;
– $m \leq n$ - довжина вихідного блоку даних.

У більшості випадків ключ (k) має таку саму довжину, що і блок (n), який обробляється, тобто n розрядів. Але в деяких випадках функції гешування можуть використовувати ключі більшої довжини.

Також з параметрами n, k, m обраховується також характеристика швидкості. Кількість операцій при блоковому шифруванні, яку необхідно виконати для формування геш-значення довжиною, що дорівнює довжині блоку шифрування.

В основу побудови функцій гешування однократної довжини покладено схеми Матіаса Майєра Озіса та Девіса Майєра. Загальним для цих схем обчислень є те, що рядок даних X, що гешується, перед цим розбивається на n-розрядні блоки $X_1, X_2, \dots, X_t$. Останній блок може доповнюватися необхідною кількістю символів [6].

Циклічна функція Матіаса Майєра Озіса визначається аналітичним співвідношенням рисунок 1.6:

$$H_i = EgH_{i-1}X_i \oplus X_i$$

Довжина блоку баних дорівнює довжині блоку шифру. Довжина геш-значення також приймається такою, що дорівнює довжині блоку блокового шифру, тобто m = n. Функція усікання забезпечує взяття найменших значущих k бітів рядка H_t.

Стійкість даних функцій базується на стійкості блокового шифру E, що використовується в їх основі. Також для таких алгоритмів не повинні реалізовуватися атаки, що засновані на використанні яких-небудь особливих властивостей і особливостей алгоритмів шифрування.

Функції гешування однократної довжини, зображена на рис. 1.9, використовуються для побудови однобічних функцій гешування на основі блокових шифрів з довжиною блоку n = 64 біта або побудови вільних від виникнення колізій функцій гешування на основі блокових шифрів з довжиною блоку n = 128 бітів.

Функції гешування з подвійною довжиною стандарту ISO/IEC 10118-2 визначає алгоритм MDC-2 як функцію гешування подвійної довжини. У MDC-2 на кожному кроці ітерації паралельно обчислюється відразу два блокових шифрування і таким чином, швидкість гешування дорівнює ½ [6].

Функція гешування подвійної довжини визначається аналітичним співвідношенням:

$$h(x) = H_t \parallel \widetilde{H_t}$$

1.3 Геш-функції на основі складнообчислювальної математичної задачі

Алгоритм RSA базується на складності факторизації великих цілих чисел. Зазвичай використовують дуже великі прості числа для створення ключів RSA, що забезпечує високу стійкість до атак на основі факторизації. Проте, з розвитком криптоаналізу та збільшенням обчислювальної потужності комп'ютерів, атаки на основі факторизації стали більш ефективними [7].

Найбільш відомою атакою на основі факторизації є метод Квантового пошуку Шора, який здатний розв'язувати задачу факторизації швидше, ніж класичні алгоритми. Інші методи атак на основі факторизації включають решіткові методи та методи поліноміального розкладу.

Для забезпечення стійкості RSA проти атак на основі факторизації використовуються ключі з довжиною від 2048 до 4096 біт, а також використовуються техніки захисту, такі як підсилення простоти чисел та використання апаратної підтримки для реалізації шифрування та дешифрування. Також в останні роки з'явилися альтернативні криптографічні протоколи, які базуються на інших математичних задачах, що не пов'язані з факторизацією, наприклад, задача дискретного логарифму та задача розкладу на множники в скінченних полях.

Виявлення великих простих чисел - відносно просте завдання, а проблема розкладання на множники, добуток двох таких чисел розглядається в обчислювальному відношенні важкооброблюваним. Рівест, Шамір і Адлеман, що базуються на труднощі цієї проблеми, розробили RSA загальне - ключову систему шифрування [7].

У той час як ціла проблема факторизації займала увагу відомих математиків подібно до Ферма та і Гауса більш ніж століття, тільки в минулих 20 роках був зроблений прогрес у вирішенні цієї проблеми. Є дві основні причини цього явища.

Спочатку, винахід RSA-системи шифрування в 1978 р. стимулював багато математиків до вивчення цієї проблеми. І швидкодіючі ЕОМ стали доступними до виконання та випробування складних алгоритмів. Фермат і Гаус мали невеликий стимул для винаходу алгоритму розкладання на множники решета поля цифр, так як цей алгоритм більш громіздкий, ніж випробувальний поділ з метою розкладання на множники цілих чисел вручну [8].

Є в основному два типи спеціалізованих та універсальних алгоритмів розкладання на множники. Спеціалізовані алгоритми розкладання на множники намагаються експлуатувати спеціальні особливості номера розкладається на множники. Поточні часи універсальних алгоритмів розкладання на множники залежить тільки від розміру n.

Геш-функції на основі складнообчислювальної математичної задачі використовують складну математичну задачу, яку важко вирішити в оберненому напрямку, для побудови геш-функції. Ці задачі часто пов'язані з теорією чисел або теорією графів і зазвичай використовуються в криптографії.

Один з найпопулярніших прикладів геш-функцій на основі складнообчислювальної математичної задачі - це RSA геш-функції. Вони використовують модульний арифметичний підхід, де вхідне повідомлення розбивається на блоки та кожен блок піддається обчисленню RSA геш-функції, використовуючи складну математичну задачу факторизації числа.

Загальним принципом побудови геш-функцій на основі складнообчислювальної математичної задачі є створення такої функції, яка була б складно вирішити в оберненому напрямку, тобто знайти вхідне повідомлення, знаючи вихідний геш-код. Це забезпечує надійний захист від зловмисних атак, таких як підробка даних або злам геш-функції.

При використанні геш-функції на основі складнообчислювальної математичної задачі, повідомлення представляється як послідовність m-розрядних блоків $M_0, M_1, M_2, \dots, M_{L-1}$.

Проблема дискретного логарифму в шифруванні виникає при використанні криптографічних систем, які базуються на складності обчислення дискретного логарифму в групах скінченного порядку, таких як група мультиплікативних залишків за модулем. Це включає в себе такі алгоритми, як Діффі-Хеллманове обмін ключами та RSA [9].

Проблема полягає в тому, що обчислення дискретного логарифму в загальному випадку вважається важкообчислюваною задачею. Це означає, що знайти дискретний логарифм a відносно заданого числа g в групі за модулем p є дуже складною задачею, коли значення p і g великі.

Ця складність полягає в тому, що існує лише експоненційна кількість можливих значень дискретного логарифму, тобто для n-бітового модуля можна мати максимум 2^n можливих значень дискретного логарифму. Тому, навіть при використанні найкращих на сьогодні алгоритмів, обчислення дискретного логарифму може зайняти дуже багато часу та ресурсів, залежно від величини чисел, які використовуються.

Ця складність полягає в тому, що існує лише експоненційна кількість можливих значень дискретного логарифму, тобто для n-бітового модуля можна мати максимум 2^n можливих Ця складність є важливим елементом багатьох криптографічних систем, таких як протокол Діффі-Хеллмана та алгоритм RSA, і вони залежать від того, щоб атакувач не міг ефективно знайти дискретний логарифм в достатньо короткий час.

Однак, з появою квантових комп'ютерів, які можуть ефективно розв'язувати проблему дискретного логарифму, ці криптографічні системи можуть стати небезпечними. Тому в даний час активно досліджуються інші методи шифрування,значень дискретного логарифму. Тому, навіть при використанні найкращих на сьогодні алгоритмів, обчислення дискретного логарифму може зайняти дуже багато часу та ресурсів, залежно від величини чисел, які використовуються.

Розглянемо на прикладі виконная даних операція піднесення до степення за модулем при побудові геш-функцій на основі складнообчислювальних математичних задач.

Спосіб 1:

$$H_i = {M_{i-1}}^{H_{i-1}} mod\ p$$

Спосіб 2:

$$H_i = {H_{i-1}}^{M_{i-1}} mod\ p$$

Спосіб 3:

$$H_i = g^{H_{i-1} \oplus M_{i-1}} mod\ p$$

g – примітивний елемент за модулем p

Геш-функції на основі складнообчислювальної математичної задачі є одним з основних способів захисту від зловмисних атак на дані. Ці геш-функції мають деякі переваги над іншими методами гешування, такими як геш-функції на основі блокових шифрів. Основна перевага полягає в тому, що складні математичні задачі важко розв'язати в оберненому напрямку, тобто знаходження вхідного повідомлення, знаючи вихідний геш-код [9].

Одним з прикладів геш-функцій на основі складнообчислювальної математичної задачі є геш-функції на основі дискретного логарифму. Ці геш-функції використовують великі прості числа та складну математичну задачу дискретного логарифму для побудови геш-функції. Іншим прикладом є геш-функції на основі еліптичних кривих, які використовують великі прості числа та еліптичні криві для побудови геш-функції.

Одним з недоліків геш-функцій на основі складнообчислювальної математичної задачі є їх повільна швидкість обробки в порівнянні з іншими геш-функціями, наприклад, з геш-функціями на основі блокових шифрів. Однак, ці геш-функції є більш надійними і забезпечують більшу безпеку даних, що робить їх популярними в криптографії та інших областях, де потрібен надійний захист від зловмисних атак на дані.

Геш-функції на основі еліптичних кривих (ECC) - це криптографічні геш-функції, які використовують операції на еліптичних кривих. Ці геш-функції зазвичай побудовані на основі алгоритмів гешування Меркла-Дамгарда або Белларда-Райпа, але використовують еліптичні криві для обчислення геш-значення.

Для побудови геш-функції на основі еліптичних кривих використовуються елементи поля Галуа, що утворюють еліптичну криву. Зазвичай використовуються криві з групою точок порядку простого числа, оскільки ці криві забезпечують достатньо високий рівень безпеки. Обчислення геш-значення полягає у вибірці початкового значення і додаванні до нього повідомлення в послідовностях блоків, застосуванні до них елементарної функції згортки, обчисленні точки на кривій, і повторенні цих кроків доки не буде отримано кінцеве значення гешу [10].

Геш-функції на основі еліптичних кривих відомі своєю високою швидкодією, а також стійкістю до більшості атак на геш-функції, включаючи диференціальний криптоаналіз та атаку з використанням колізій. Однак, для забезпечення безпеки цих геш-функцій необхідно вибирати відповідні параметри еліптичних кривих.

Геш-функції на основі дискретного логарифму використовуються в криптографії для створення безпечних геш-функцій. Дискретний логарифм - це математична операція, яка полягає в пошуку експоненти, до якої потрібно піднести певне число, щоб отримати інше число, в заданому простому полі.

Для побудови геш-функцій на основі дискретного логарифму використовуються так звані "пастки дискретного логарифму". Пастки дискретного логарифму - це числа, для яких знаходження дискретного логарифму є важким завданням. Такі числа можуть бути використані для побудови безпечних геш-функцій, так як вирахування дискретного логарифму з пастки є важким завданням.

Ефективними системами криптографічного захисту даних є асиметричні криптосистеми, які називають також криптосистемами з відкритим ключем. У таких системах для зашифрування даних використовується один ключ, а для розшифрування – інший ключ. Перший ключ є відкритим і може бути опублікований для використання всіма користувачами системи, які зашифровують дані. Розшифрувати дані за допомогою відкритого ключа неможливо [11].

Для розшифрування даних одержувач зашифрованої інформації використовує другий ключ, що є таємним. Зрозуміло, таємний ключ не може бути визначений, виходячи з відкритого ключа.

Існують ефективні алгоритми, що дозволяють досить швидко обчислити значення функції $f_{A,N}(x)$. Якщо $y = A^x$, то, природно, $x = \log_A(y)$. Тому, задачу обернення функції $f_{A,N}(x)$ називають задачею знаходження дискретного логарифма або задачею дискретного логарифмування. Задача дискретного логарифмування формулюється в такий спосіб.

Для відомих цілих A, N, y знайти ціле число x, що $A\ mod(N) = y$. Алгоритм обчислення дискретного логарифма за прийнятний час поки не знайдена, тому модульна експонента вважається односпрямованою функцією [11].

1.4 Геш-функції побудовані з нуля

Геш-функції можна розробляти з нуля, без використання готових блокових шифрів або складних математичних задач. Одним з таких методів є конструкція Меркла-Дамгарда. Основна ідея полягає у тому, що повідомлення поділяється на блоки фіксованої довжини, які послідовно обробляються за допомогою функції зв'язку. Функція зв'язку бере на вхід блок повідомлення та попереднє значення геш-функції і повертає нове значення геш-функції. Для того, щоб забезпечити відповідну безпеку, функція зв'язку повинна бути такою, що неможливо знайти два різних вхідних блоки, які дають одне і те ж значення геш-функції. Один з найпопулярніших варіантів функції зв'язку - це конструкція на основі блокових шифрів. За допомогою фінальної функції обчислюється кінцеве значення геш-функції [12].

Інший метод побудови геш-функцій з нуля - метод роздільної зміни (separate chaining). У цьому методі кожен можливий значення геш-функції відображається в список зв'язаних блоків, які містять ключі з таким же значенням геш-функції. Якщо при додаванні нового ключа до таблиці виникає колізія, тобто ключ вже існує в таблиці, то новий ключ додається до списку відповідного значення геш-функції. Цей метод є простим та ефективним, але може страждати від проблеми "загальної забрудненості", коли під час використання таблиці з багатьма ключами в одному списку накопичується занадто багато елементів.

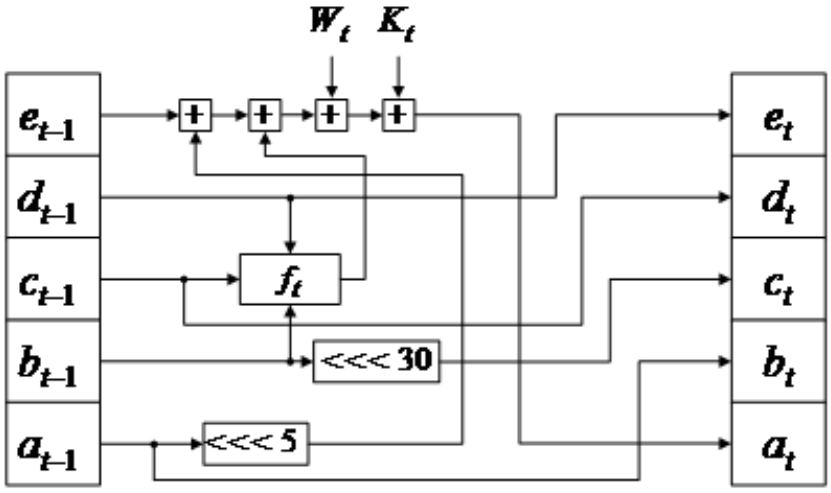


Рисунок 1.8 - Приклад геш-функції SHA-1

Проте, з часом виявилися певні уразливості в геш-функції SHA-1, дивитися рис. 1.8, тому вона була замінена на більш безпечну версію - SHA-2, що на рис. 1.9, яка також використовує дискретний логарифм для створення геш-функції. Зокрема, у SHA-2 використовуються різні типи пасток дискретного логарифму для забезпечення ще більшої безпеки [12].

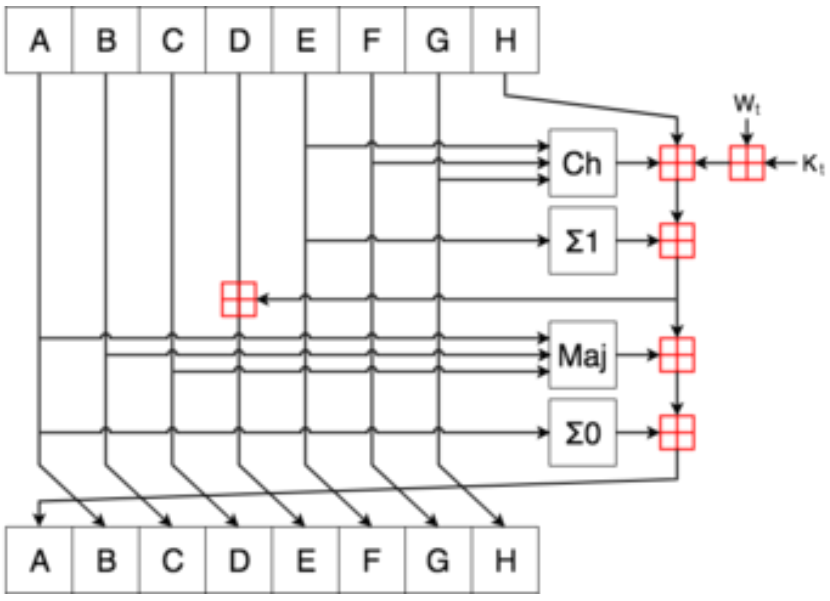


Рисунок 1.9 – Приклад геш-функції SHA-2

Метод лінійного пошуку (linear probing) є іншим методом побудови геш-функцій з нуля. У цьому методі, якщо при додаванні нового ключа в таблицю виникає колізія, то відбувається спроба збільшити індекс ключа

Однією з найпростіших функцій гешування розроблених з нуля є зв’язування всіх блоків операцією порозрядного виключного АБО:

$$C_i = b_{i1} \oplus b_{i2} \oplus \ldots \oplus b_{im}$$

, де C_i – і-й біт геш-коду,

m – число -бітових блоків вихідних даних,

b_{ij} – і-й біт у j-му блоці,

$\oplus$ - операція АБО.

Функції гешування розроблені з нуля спеціально створюють для задач гешування. Вони оптимізують тимчасові витрати на обчислення, що пов’язані, насамперед, при виконанні алгоритму зі звертанням до пам’яті. Функції цього класу використовують алгоритми обчислень, що вперше реалізовані в проекті алгоритму MD4.

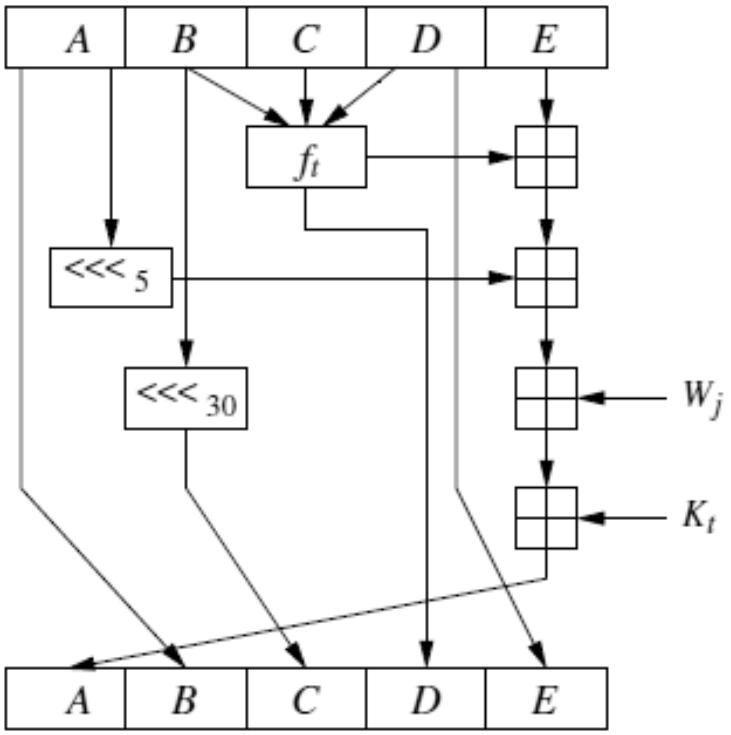


Рисунок 1.10 – Схема алгоритму гешування SHA-1

SHA-1 реалізує хеш-функцію, яка на рис. 1.10, побудовану на ідеї функції стиснення. Входом функції стиснення є блок повідомлення довжиною 512 біт і вихід попереднього блоку повідомлення. Виходом є значення всіх хеш-блоків до цього

моменту. Іншими словами хеш блоку M_i дорівнює $h_i = f(M_i, h_{i-1})$. Хеш-значенням всього повідомлення є виходом останнього блоку [12].

Вхідне повідомлення розбивається на блоки по 512 біт у кожному. Останній блок доповнюється до довжини кратної 512 біт. Спочатку додається 1, а потім нулі, щоб довжина блоку стала рівною (512 – 64 = 448) біт.

В останні 64 біта записується довжина вихідного повідомлення у бітах. Якщо останній блок має довжину понад 448, але менше 512 біт, то додаток виконується в такий спосіб:

– Спочатку додається 1, потім нулі аж до кінця 512-бітного блоку;
– після цього створюється ще один 512-бітний блок, який заповнюється аж до 448 біта нулями, після чого в 64 біта, що залишилися;
– записується довжина вихідного повідомлення в бітах.

Доповнення останнього блоку здійснюється завжди, навіть якщо повідомлення вже має потрібну довжину [12].

До геш-функцій розроблених з нуля відноситься удосконалена проста функція гешування. В даній функції початкова ініціалізація n-бітового значення функції гешування є нульовим значенням.

Обробка n-блоків відбувається послідовно за правилом, а саме:

– Виконання циклічного зсуву проміжного значення функції гешування вліво на один біт;
– додавання поточного блоку до проміжного значення функції гешування за допомогою функції АБО.

Keccak – це криптофункція, яка у 2012 році стала переможцем конкурсу криптографічних алгоритмів та була затверджена як новий федеральний стандарт обробки інформації США [13].

Keccak використовує інноваційний механізм губки для хешування тексту повідомлення. Він має середню швидкість 12,5 циклів на байт на процесорі Intel Core 2, а його простий дизайн добре підходить для апаратної реалізації.

Keccak може протистояти відомим атакам із мінімальною складністю 2n, де n – розмір хешу. Він має великий запас міцності. На сьогоднішній день сторонній криптоаналіз не показав серйозних недоліків у Кекчаку. Тим не менш, творці Keccak беруть участь у конкурсі Crunchy Crypto Contest, де задаються проблеми, визначаються правила та звіти про статус зіткнення та попереднього зображення Keccak Crunchy. Існує три частини алгоритму Кечкака, зображені на рис. 1.11.

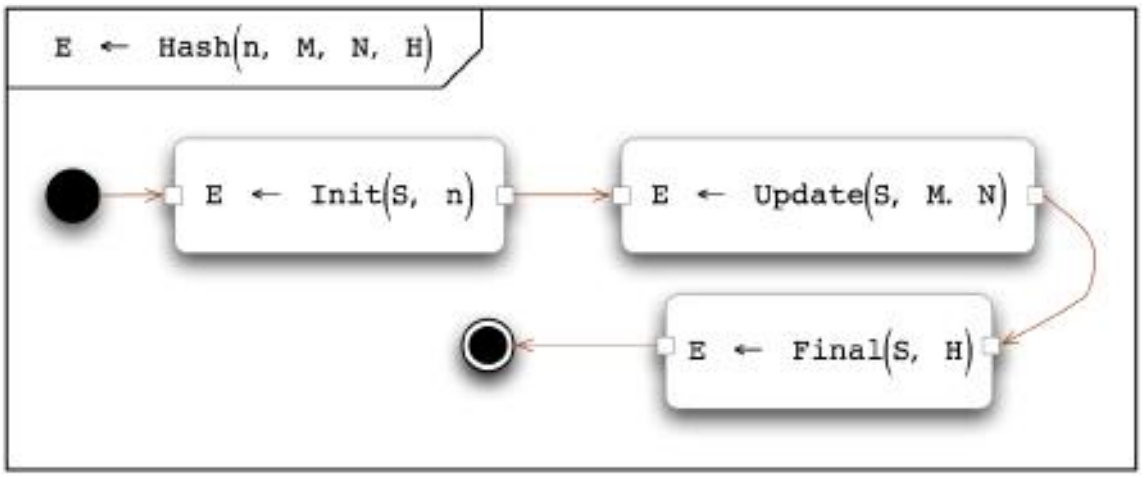


Рисунок 1.11 – Частини алгоритму Keccak

Функція Hash (n, M, N, H) є функцією введення. Він приймає чотири вхідні аргументи:

– Розмір хешу (n) у бітах;
– текст повідомлення (M);
– розмір повідомлення (N) у бітах;
– хеш-змінну (H).

Алгоритм стандарту Keccak має структурну конструкцію губки, як на рис. 1.12, де всі вступні дані спочатку ніби вбираються і підсумовуються по вузлу 2 з інваріантами стану, потім усередині криптографічної губки створюються багатосерійні перестановки і на виході отримується зашифрований результат [18].

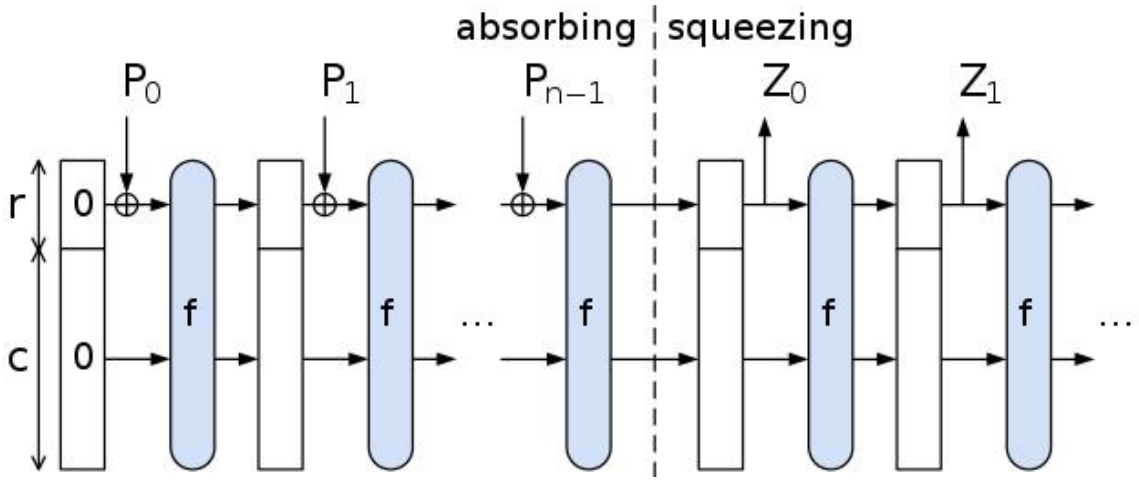


Рисунок 1.12 – Структура алгоритму Keccak

Щоб зашифрувати рядок, його слід розбити на безліч фрагментів певного розміру і на кожному раунді підмішати всі отримані частини не до повноцінного 1600-бітового стану, а лише до верхнього розміру (на малюнку – це проміжок «r»). І так відбувається на кожному раунді хешування. Завдяки тому, що використовується не цілий стан, а лише певна частина, не можна отримати інформацію про зв'язок даних на вході та виході.

2 РОЗРОБКА МЕТОДУ ГЕШУВАННЯ

2.1 Метод гешування на основі кватерніонів

Кватерніон - це гіперкомплексне число, яке можна представити у вигляді суми складових, де перша складова - це звичайне комплексне число, а решта трьох - уявні одиниці, які називають кватерніонними одиницями [14].

Кожен кватерніон також складається з дійсної та уявної частин. Однак для кватерніонів використовується не одна уявна одиниця, а цілих три - i, j, k.

Використовуючи ці уявні одиниці, будь-який кватерніон можна записати у вигляді:

$$q = a + i * b + j * c + k * d$$

Тут b, c, d і a - звичайні дійсні числа. Дійсною частиною кватерніона q є a, а уявною: ib + jc + kd.

У той же час кватерніон можна розглядати просто як чотиривимірний вектор і для цих векторів ввести операції покомпонентного додавання, віднімання і множення на дійсне число. Відповідно, сума двох кватерніонів q_1 та q_2 в цьому випадку буде визначатися формулою.

$$q_1 + q_2 = a_1 + a_2 + i * (b_1 + b_2) + j * (c_1 + c_2) + k * (d_1 + d_2)$$

Однак на відміну від векторів, для кватерніонів також можна ввести і операцію множення двох кватерніонів. Для цього достатньо розкрити дужки в відповідному добутку і скористатися властивостями уявних одиниць.

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

Тоді отримаємо:

$$q_1 * q_2 = (a_1a_2 - b_1b_2 - c_1c_2 - d_1d_2) + i * (a_1b_2 + a_2b_1 + c_1d_2 - c_2d_1) +$$
$$+ j * (a_1c_2 + a_2c_1 + b_2d_1 - b_1d_2) + k * (a_1d_2 + a_2d_1 + b_1c_2 - b_2c_1)$$

Для погіршення знаходження мультиколізій у геш-функцій, пропонується використовувати конструкцію ітеративного гешування з розпаралеленням обчислень і з подальшим зав’язуванням між собою проміжних результатів функції. Великою

перевагою конструкції гешування з розпаралеленням обчислень і зав'язуванням проміжних результатів є те, що обчислення в кожному каналі є незалежним. Така методика можливість виконати кожне обчислення окремо і отримувати незалежне геш-значення. Одночасне використання декількох блоків даних на кожному рівні ітерації і їх перемішування для свого каналу дозволяє отримати значення необхідної довжини, яке буде стійке до колізій і побудоване за короткий час [14].

Розглянемо метод для паралельного гешування, який грунтується на проміжних геш-значеннях і структурі даних, що описуються моделями кватерніонів.

В основі даної ідеї гешування даних лежить те, що проміжні геш-значення h_{i-1} і значення усіх блоків даних m_i, що підлягають гешуванню розрядністю n, діляться на 4 частини і мають представлення кватерніонів. Тому проміжне значення h_{i-1} представляється у вигляді:

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

Аналогічний вигляд має представлення блоку даних m_i:

$$m_i = a_2 + b_2 i + c_2 j + d_2 k,$$

Операція множення дає змогу забезпечити рівномірний вплив кожного біта кватерніона на вихідне геш-значення, доцільно її використовувати для домішування результату обробки блоків даних до геш-значення, отриманого на попередній ітерації. Враховуючи це, пропонується визначити значення геш-функції як добуток вхідних елементів за модулем 2^n.

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

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

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

Кожен з добутків 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:

$$(p_1 + p_2) mod 2^n$$

З урахуванням, цього результат множення кватерніонів з ущільненням можна представити в такому вигляді:

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

, де

$$P_{a_1 a_2} = (p_{1,a_1 a_2} + p_{2,a_1 a_2}) mod 2^n,\ P_{b_1 b_2} = (p_{1,b_1 b_2} + p_{2,b_1 b_2}) mod 2^n,$$
$$P_{c_1 c_2} = (p_{1,c_1 c_2} + p_{2,c_1 c_2}) mod 2^n,\ P_{d_1 d_2} = (p_{1,d_1 d_2} + p_{2,d_1 d_2}) mod 2^n,$$
$$P_{a_1 b_2} = (p_{1,a_1 b_2} + p_{2,a_1 b_2}) mod 2^n,\ P_{b_1 a_2} = (p_{1,b_1 a_2} + p_{2,b_1 a_2}) mod 2^n,$$
$$P_{c_1 d_2} = (p_{1,c_1 d_2} + p_{2,c_1 d_2}) mod 2^n,\ P_{d_1 c_2} = (p_{1,d_1 c_2} + p_{2,d_1 c_2}) mod 2^n,$$
$$P_{a_1 c_2} = (p_{1,a_1 c_2} + p_{2,a_1 c_2}) mod 2^n,\ P_{c_1 a_2} = (p_{1,c_1 a_2} + p_{2,c_1 a_2}) mod 2^n,$$
$$P_{b_1 d_2} = (p_{1,b_1 d_2} + p_{2,b_1 d_2}) mod 2^n,\ P_{d_1 b_2} = (p_{1,d_1 b_2} + p_{2,d_1 b_2}) mod 2^n,$$
$$P_{a_1 d_2} = (p_{1,a_1 d_2} + p_{2,a_1 d_2}) mod 2^n,\ P_{d_1 a_2} = (p_{1,d_1 a_2} + p_{2,d_1 a_2}) mod 2^n,$$
$$P_{b_1 c_2} = \big(p_{1,b_1 c_2} + p_{2,b_1 c_2}\big) mod 2^n,\ P_{c_1 b_2} = \big(p_{1,c_1 b_2} + p_{2,c_1 b_2}\big) mod 2^n.$$

2.2 Приклад реалізації методу гешування

Для множення кватерніонів використовується геш-код та блок даних довжиною 256 біт. Довжина добутку коефіцієнтів кватерніона становить 128 біт, а довжина коефіцієнтів кватерніонів – 128 біт. Додавання і віднімання добутків проводиться за модулем 2^{64} [15].

Для прикладу гешування повідомлення, розглянемо множення кватерніонів, які будуть наведені нижче. Для цього використаємо двійкові коди $h_{i-1} = 0010.0011.0101.1000$ та $m_i = 0011.0111.1011.1101$.

В нашому прикладі h_{i-1} - це проміжне геш-значення, а m_i – повідомлення, яке буде гешуватися.

Для початку розіб’ємо складові добутку окремо і будемо розглядати кожен з них. Перший складовий - $\left(P_{a_1a_2} - P_{b_1b_2} - P_{c_1c_2} - P_{d_1d_2}\right)$. Тепер представимо складові проміжного геш-значення та повідомлення в десятковій системі числення.

$$h_{i-1} = a_1 + b_1 i + c_1 j + d_1 k$$

Розклянемо покрокове множення кватерніонів. Обчислення першої складової добутку кватерніонів.

$\left(P_{a_1a_2} - P_{b_1b_2} - P_{c_1c_2} - P_{d_1d_2}\right) mod 16$

$p_{1,a_1a_2} = 0, p_{2,a_1a_2} = 6$

$P_{a_1a_2} = \left(p_{1,a_1a_2} + p_{2,a_1a_2}\right) mod 16 = (0 + 6) \bmod 16 = 6$

$p_{1,b_1b_2} = 16, p_{2,b_1b_2} = 5$

$P_{b_1b_2} = \left(p_{1,b_1b_2} + p_{2,b_1b_2}\right) mod 16 = (16 + 5) \bmod 16 = 6$

$p_{1,c_1c_2} = 3, p_{2,c_1c_2} = 7$

$P_{c_1c_2} = \left(p_{1,c_1c_2} + p_{2,c_1c_2}\right) mod 16 = (3 + 7) \bmod 16 = 10$

$p_{1,d_1d_2} = 6, p_{2,d_1d_2} = 8$

$P_{d_1d_2} = \left(p_{1,d_1d_2} + p_{2,d_1d_2}\right) mod 16 = (6 + 8) \bmod 16 = 14$

Відповідно до виконаних обчислень, перша складова кватерніона добутку дорівнює $\left(P_{a_1a_2} - P_{b_1b_2} - P_{c_1c_2} - P_{d_1d_2}\right) mod 16 = (6 - 6 - 10 - 14) mod 16 = 8$. Подальші обчислення складових кватерніона проводимо за таким самим принципом.

Обчислення другої складової добутку кватерніона, а саме: $i\left(P_{a_1b_2} + P_{b_1a_2} + P_{c_1d_2} - P_{d_1c_2}\right)$.

$p_{1,a_1b_2} = 0, p_{2,a_1b_2} = 14$

$P_{a_1b_2} = (0 + 14) mod 16 = 14$

$p_{1,b_1a_2} = 0, p_{2,b_1a_2} = 9$

$P_{b_1a_2} = (0 + 9) mod 16 = 9$

$p_{1,c_1d_2} = 4, p_{2,c_1d_2} = 1$

$P_{c_1d_2} = (4 + 1) mod 16 = 5$

$p_{1,d_1c_2} = 5, p_{2,d_1c_2} = 8$

$P_{d_1c_2} = (5 + 8)mod16 = 13$

Після виконання множення отримуємо результат, що друга складова кватерніона має вигляд $i\left(P_{a_1b_2} + P_{b_1a_2} + P_{c_1d_2} - P_{d_1c_2}\right)mod16 = i(14 + 9 + 5 - 13)mod16 = 15i.$

Обчислюємо третю складову добутку кватерніона, $j\left(P_{a_1c_2} + P_{c_1a_2} + P_{b_1d_2} - P_{d_1b_2}\right).$

$p_{1,a_1c_2} = 1\,, p_{2,a_1c_2} = 6$

$P_{a_1c_2} = (\,1 + 6)mod16 = 7$

$p_{1,c_1a_2} = 0, p_{2,c_1a_2} = 15$

$P_{c_1a_2} = (0 + 15)mod16 = 15$

$p_{1,b_1d_2} = \ 2, p_{2,b_1d_2} = 7$

$P_{b_1d_2} = (2 + 7)mod16 = 9$

$p_{1,d_1b_2} = \ 3, p_{2,d_1b_2} = 8$

$P_{d_1b_2} = (3 + 8\,)mod16 = 11$

З отриманих обчислень третя складова кватерніона $j\left(P_{a_1c_2} + P_{c_1a_2} + P_{b_1d_2} - P_{d_1b_2}\right)$ буде мати вигляд, $j(7 + 15 + 9 - 11)mod16 = 4j.$

Обчислюємо четверту складову кватерніона, $k\left(P_{a_1d_2} + P_{d_1a_2} + P_{b_1c_2} - P_{c_1b_2}\right).$ Так само окремо обчислюємо його складові і потім знаходимо загальне значення.

$p_{1,a_1d_2} = \ 1, p_{2,a_1d_2} = 10$

$P_{a_1d_2} = (1 + 10)mod16 = 11$

$p_{1,d_1a_2} = \ 1, p_{2,d_1a_2} = 8$

$P_{d_1a_2} = (1 + 8\,)mod16 = 9$

$p_{1,b_1c_2} = \ 2, p_{2,b_1c_2} = 1$

$P_{b_1c_2} = (2 + 1)mod16 = 3$

$p_{1,c_1b_2} = \ 2, p_{2,c_1b_2} = 3$

$P_{c_1b_2} = (2 + 3)mod16 = 5$

Четверта складова $\left(P_{a_1 d_2} + P_{d_1 a_2} + P_{b_1 c_2} - P_{c_1 b_2}\right)k$ кватерніона дорівнює, $k(26 + 9 + 3 - 5)mod16 = 1k.$

Після всіх виконаних операцій маємо добуток $h_{i-1} * m_i = 8 + 15i + 4j + + 1k$, якому відповідає такий двійковий код $h_i = 1000.1111.0100.0001.$

Для геш-функцій важливим є так званий лавинний ефект. Тобто зміна одного біта в даних, що підлягають гешуванню має забезпечувати зміну великої кількості бітів геш-значення (понад 0.5 довжини геш-значення).

Покажемо як змінюється біти геш-значення, якщо в повідомленні блок 1101 замінити на 1111.

Нове півідомлення матиме вигляд:

$$m_i = 0011.0111.1011.1111$$

Тепер після змін маємо нові значення $a_2,\ b_2,\ c_2,\ d_2,$ відповідно $3, 7, 11, 15.$ Далі виконуємо такі ж самі дії як попередньо знаходили кожну складову кватерніона.

Перша складова кватерніона:

$p_{1,a_1 a_2} = 0, p_{2,a_1 a_2} = 6$

$P_{a_1 a_2} = \left(p_{1,a_1 a_2} + p_{2,a_1 a_2}\right)mod16 = (0 + 6)mod16 = 6$

$p_{1,b_1 b_2} = 1, p_{2,b_1 b_2} = 5$

$P_{b_1 b_2} = \left(p_{1,b_1 b_2} + p_{2,b_1 b_2}\right)mod16 = (1 + 5)\text{mod}16 = 6$

$p_{1,c_1 c_2} = 3, p_{2,c_1 c_2} = 7$

$P_{c_1 c_2} = \left(p_{1,c_1 c_2} + p_{2,c_1 c_2}\right)mod16 = (3 + 7)\text{mod}16 = 10$

$p_{1,d_1 d_2} = 7, p_{2,d_1 d_2} = 8$

$P_{d_1 d_2} = \left(p_{1,d_1 d_2} + p_{2,d_1 d_2}\right)mod16 = (7 + 8)\text{mod}16 = 15$

$\left(P_{a_1 a_2} - P_{b_1 b_2} - P_{c_1 c_2} - P_{d_1 d_2}\right)mod16 = (6 - 6 - 10 - 15)\text{mod}16 = 7$

Друга складова кватерніона:

$i\left(P_{a_1 b_2} + P_{b_1 a_2} + P_{c_1 d_2} - P_{d_1 c_2}\right).$

$p_{1,a_1 b_2} = 0, p_{2,a_1 b_2} = 14$

$P_{a_1 b_2} = (0 + 14)mod16 = 14$

$p_{1,b_1 a_2} = 0, p_{2,b_1 a_2} = 9$

$P_{b_1a_2} = (0 + 9)mod16 = 9$

$p_{1,c_1d_2} = 64, p_{2,c_1d_2} = 11$

$P_{c_1d_2} = (64 + 11\ )mod16 = 15$

$p_{1,d_1c_2} = 5, p_{2,d_1c_2} = 8$

$P_{d_1c_2} = (5 + 8)mod16 = 13$

$i\left(P_{a_1b_2} + P_{b_1a_2} + P_{c_1d_2} - P_{d_1c_2}\right)mod16 = i(14 + 9 + 15 - 13)mod16 = 9i$

Третя складова кватерніона:

$j\left(P_{a_1c_2} + P_{c_1a_2} + P_{b_1d_2} - P_{d_1b_2}\right).$

$p_{1,a_1c_2} = 1\ , p_{2,a_1c_2} = 6$

$P_{a_1c_2} = (\ 1 + 6)mod16 = 7$

$p_{1,c_1a_2} = 0, p_{2,c_1a_2} = 15$

$P_{c_1a_2} = (0 + 15)mod16 = 15$

$p_{1,b_1d_2} = 32, p_{2,b_1d_2} = 13$

$P_{b_1d_2} = (32 + 13)mod16 = 15$

$p_{1,d_1b_2} = 3, p_{2,d_1b_2} = 8$

$P_{d_1b_2} = (3 + 8\ )mod16 = 1$

$\left(P_{a_1c_2} + P_{c_1a_2} + P_{b_1d_2} - P_{d_1b_2}\right) = (7 + 15 + 15 - 1)mod16 = 4j$

Четверта складова кватерніона:

$\left(P_{a_1d_2} + P_{d_1a_2} + P_{b_1c_2} - P_{c_1b_2}\right)k$

$p_{1,a_1d_2} = 16, p_{2,a_1d_2} = 14$

$P_{a_1d_2} = (16 + 14)mod16 = 15$

$p_{1,d_1a_2} = 1, p_{2,d_1a_2} = 8$

$P_{d_1a_2} = (1 + 8\ )mod16 = 9$

$p_{1,b_1c_2} = 2, p_{2,b_1c_2} = 1$

$P_{b_1c_2} = (2 + 1)mod16 = 3$

$p_{1,c_1b_2} = 2, p_{2,c_1b_2} = 3$

$P_{c_1b_2} = (2 + 3)mod16 = 5$

$$\left(P_{a_1 d_2} + P_{d_1 a_2} + P_{b_1 c_2} - P_{c_1 b_2}\right)k = k(15 + 9 + 3 - 5) mod 16 = 6k$$

Виконавши заміну біта в повідомлені, ми можемо проаналізувати, що наш новий добуток кватерніона змінився і тепер має інший вигляд:

$$h_{i-1} * m_i = 7 + 9i + 4j + 6k$$

Відповідно зі зміною добутку, змінився і геш, який після усіх операцій має результат:

$$h_i = 0111.1001.1010.0110$$

Старе геш-значення, має вигляд:

$$h_i = 1000.1111.0100.0001$$

Зміна навіть одного біта в геш значенні може призвести до значних змін в результаті геш-функції. Основні впливи зміни біта в повідомленні на геш-значення такі:

– Унікальність: зміна одного біта в геш-значенні може призвести до отримання зовсім іншого геш-значення. В ідеальному випадку, зміна будь-якого біта вхідного повідомлення повинна призвести до повної зміни геш-значення;
– розподіленість: криптографічна геш-функція розподіляє геш-значення рівномірно по всьому діапазону можливих значень. Зміна біта в повідомленні, може вплинути на розподіл геш-значень, зміщуючи їх у конкретний сегмент діапазону;
– колізії: колізія в геш-функції виникає, коли два різних вхідних повідомлення мають одне й те ж геш-значення. Зміна навіть одного біта може призвести до повної зміни геш-значення і уникнення колізій.

Загалом, зміна біта в повідомленні може мати великий вплив на унікальність, розподіленість і колізії геш-функції. Це важливо враховувати при роботі з геш-функціями в криптографії, безпеці даних та інших сферах, де геш-функції використовуються [16].

2.3 Опис алгоритму для гешування

Після виконання множення кватерніонів, створимо блок-схему алгоритму множення. Дана схема, що на рис. 2.1 буде описувати роботу майбутнього програмного засобу для виконання множення кватерніонів та допоможе при його створенні.

Попереднє множення, виконувалося в однопоточному процесі, програмний засіб буде виконувати це паралельно, що дає змогу пришвидшити роботу програми.

Дана блок-схема описує алгоритм, який виконує наступні кроки:

- початок – початок виконання алгоритму;
- введення змінних та їх ініціалізація – користувач вводить значення чотирьох змінних a_1, b_1, c_1, d_1 для складових проміжного геш-значення;
- введення повідомлення для гешування – користувач вводить значення чотирьох змінних a_2, b_2, c_2, d_2 для складових блоку даних;
- обчислення 1 складової кватерніона;
- обчислення 2 складової кватерніона;
- обчислення 3 складової кватерніона;
- обчислення 4 складової кватерніона;
- отримання результату після обчислення усіх чотирьох складових кватерніона – вивід результату після завершення обчислення усіх 4 складових кватерніон;
- отримання результату після обчислення складових блоку даних;
- кінець – завершення програми.

При реалізації алгоритму гешування на основі моделі кватерніонів, для отримання 256-бітного геш-значення, функції будуть обробляти 4 канали вхідних даних розрядністю в 64 біти кожний. Обробка вхідного значення величиною 32 байти буде виконуватись за один цикл. Для обчислення одного проміжного геш-значення за алгоритмом що пропонується, потрібно виконати 16 операцій множення і 28 операцій додавання на кожній ітерації.

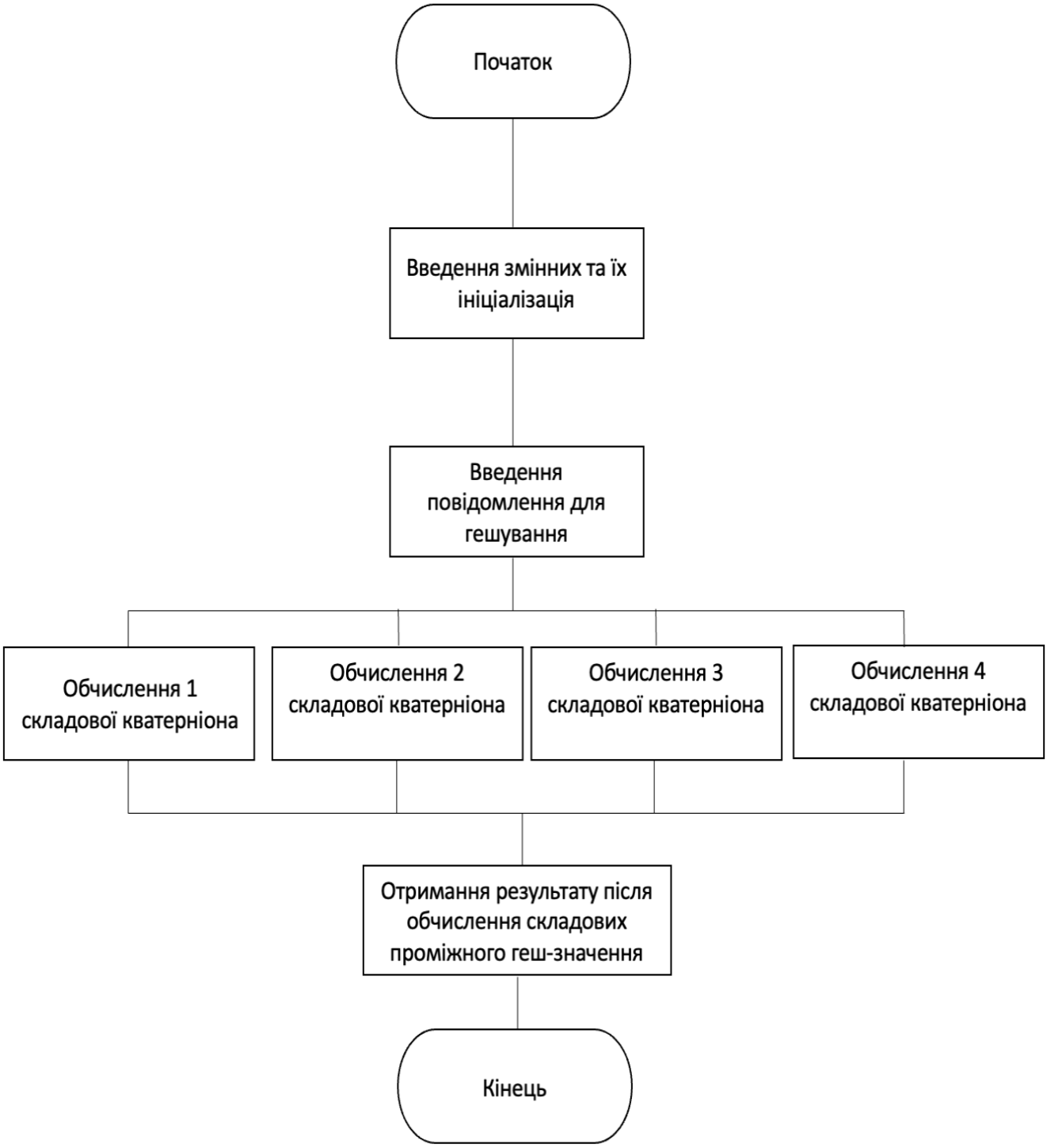


Рисунок 2.1 – Блок схема паралельного алгоритму множення

В результаті для обчисленні геш-значення на один байт вхідних даних розроблюваний алгоритм виконує 44/32 = 1.38 операцій, для геш-функції на основі моделі кватерніона, тут число 44 – це число складових операцій при обчисленні добутку кватерніона.

Для порівняння, один з найбільш швидких фіналістів конкурсу на SHA-3, Skein, показував результат в 6.1 циклів на байт в 64-розрядному варіанті коду для гешу довжиною 512 біт, результати порівняння в табл. 2.1 [16].

Таблиця 2.1 – Порівняльні характеристики алгоритмів

Алгоритм	Довжина геш-значення (біт)	Циклів/байт
Запропонований алгоритм	256	1.38
Skein	256	6.1
BMW	256	7.32
LAKE	256	8.19
BLAKE256	256	8.4
Keccak	256	10
RIPEMD256	256	11.1
SHA256	256	15.0

Як видно з таблиці, запропонований алгоритм прискореного гешування даних має явну перевагу в швидкодії над відомими алгоритмами, оскільки потребує меншої кількості виконуваних операцій на один байт вхідного повідомлення.

3 ПРОГРАМНИЙ ЗАСІБ ДЛЯ ГЕШУВАННЯ ДАНИХ НА ОСНОВІ КВАТЕРНІОНІВ

3.1. Інтегроване середовище розробки PyCharm

PyCharm є одним з найзручніших і популярних інтегрованих середовищ розробки (IDE) для мови Python [17].

Його можна отримати у двох варіантах:

- PyCharm Community - це безкоштовна версія з відкритим вихідним кодом. Вона надає зручний набір інструментів для розробки проектів Python, допомагаючи розширити можливості вашого програмування. За допомогою PyCharm Community ви можете легко створювати, змінювати і відлагоджувати програми Python, реалізовувати різні алгоритми та робити експерименти з кодом;
- PyCharm Professional - це платна версія з тріальним періодом, яка має додаткові функціональні можливості. Вона дозволяє працювати не тільки з Python, але і з іншими суміжними мовами програмування, такими як веб-програмування. Крім того, PyCharm Professional надає доступ до безлічі фреймворків, які допомагають розробникам створювати складні та масштабні проекти.

Обидві версії PyCharm надають зручну інтерфейс, автодоповнення коду, можливість відлагодження програм, керування версіями за допомогою систем контролю версій та багато іншого. Вони спрощують процес розробки, дозволяючи вам більш швидко та ефективно програмувати.

Отже, якщо ви шукаєте потужне та зручне середовище розробки для Python, PyCharm може стати відмінним вибором, незалежно від того, для яких завдань ви його використовуєте.

Під час першого запуску PyCharm ви зустрінетеся з декількома кроками, які допоможуть налаштувати і настроїти ваше середовище розробки. По-перше, вам буде запропоновано прийняти угоду користувача, яку ви повинні прочитати та прийняти, щоб продовжити використання програми.

Далі з'явиться вікно з питанням щодо анонімної передачі даних про використання продукту. Ви можете вибрати, чи надсилати такі дані, відповідаючи на це питання згідно з вашими вподобаннями та політикою конфіденційності.

Після цього ви побачите вітальне вікно, яке пропонує вам створити новий проєкт, зображено на рис. 3.1. У цьому вікні ви можете вказати назву проекту, вибрати розташування файлів проекту на вашому комп'ютері та вибрати інтерпретатор Python, який буде використовуватися в проекті.

Ці кроки першого запуску допоможуть вам налаштувати PyCharm згідно з вашими потребами і почати роботу з розробкою проектів Python у зручному середовищі.

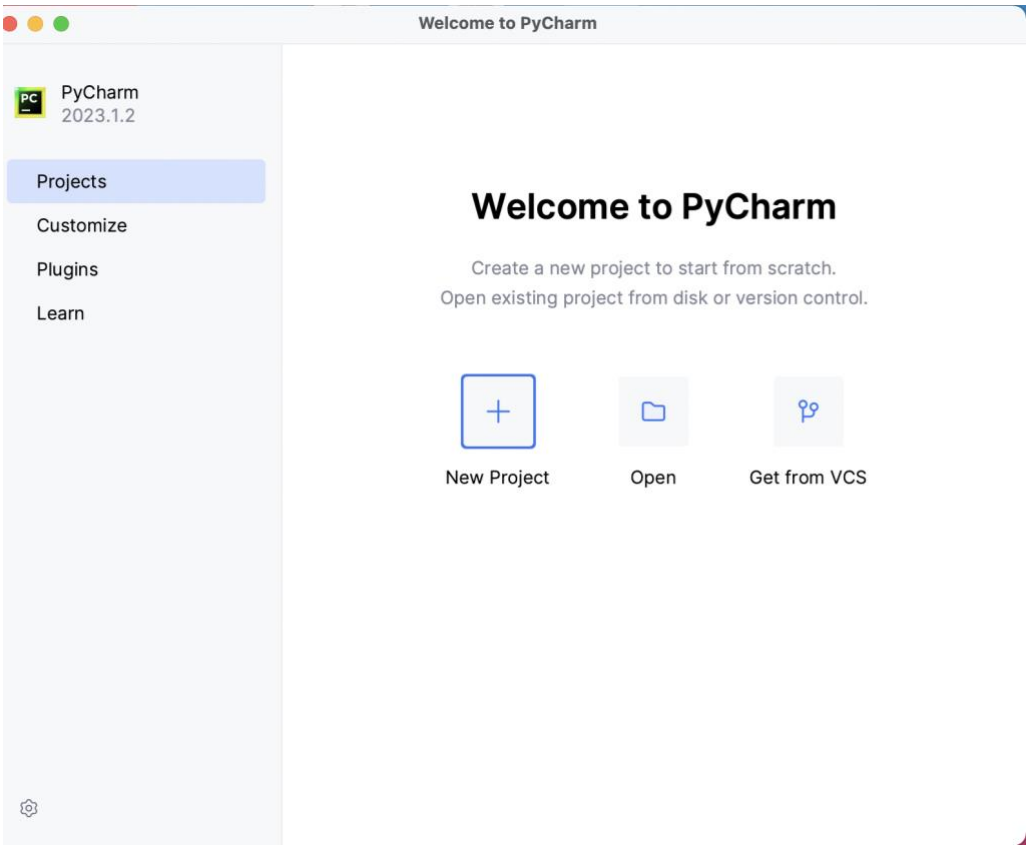


Рисунок 3.1 – Початкове вікно запуску PyCharm

Під час створення нового проєкту у PyCharm, в запропонованому діалоговому вікні, що зображене на рис. 3.2, вказуємо рядок, що представляє адресу нового каталогу для проєкту. Можна ввести власний шлях до каталогу або погодитися зі значенням за замовчуванням. Цей шлях вказує на місце, де будуть зберігатися файли вашого проєкту.

Крім того, в цьому ж діалоговому вікні є можливість вибрати, власне віртуальне оточення для проєкту. Віртуальне оточення - це ізольоване середовище, в якому будуть встановлені пакети та залежності, специфічні для проєкту. Використання віртуальних оточень рекомендується для керування залежностями та запобігання конфліктів між різними проектами [17].

Ці параметри, такі як адреса каталогу та створення віртуального оточення, дозволяють налаштувати власне середовище розробки в PyCharm, забезпечуючи зручність та гнучкість для вашого проєкту.

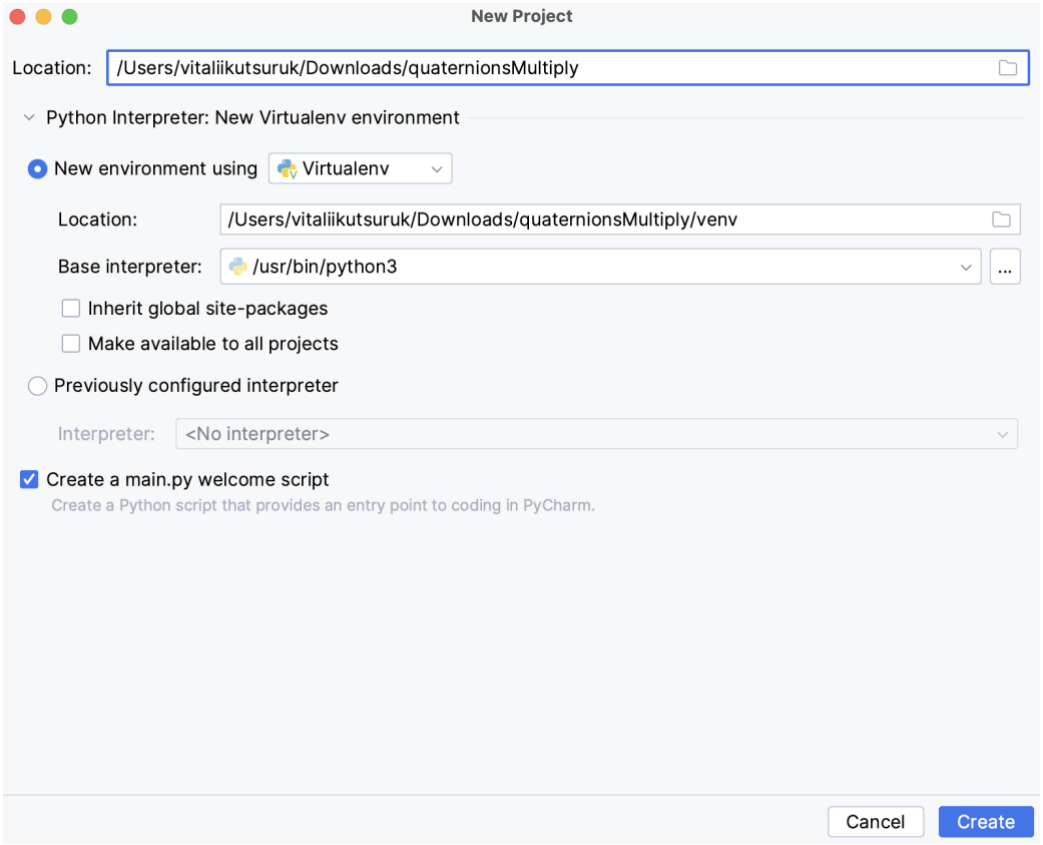


Рисунок 3.2 – Створення проекту в середовищі PyCharm

Після створення назви проекту та місця розташування для нього, середовище розробки генерує стартовий вигляд програми, як на рис. 3.3. Ми можемо запустити отриманий шаблон, щоб впевнитись, що все створено вірно або видалити все і почати створювати власний проект з нуля.

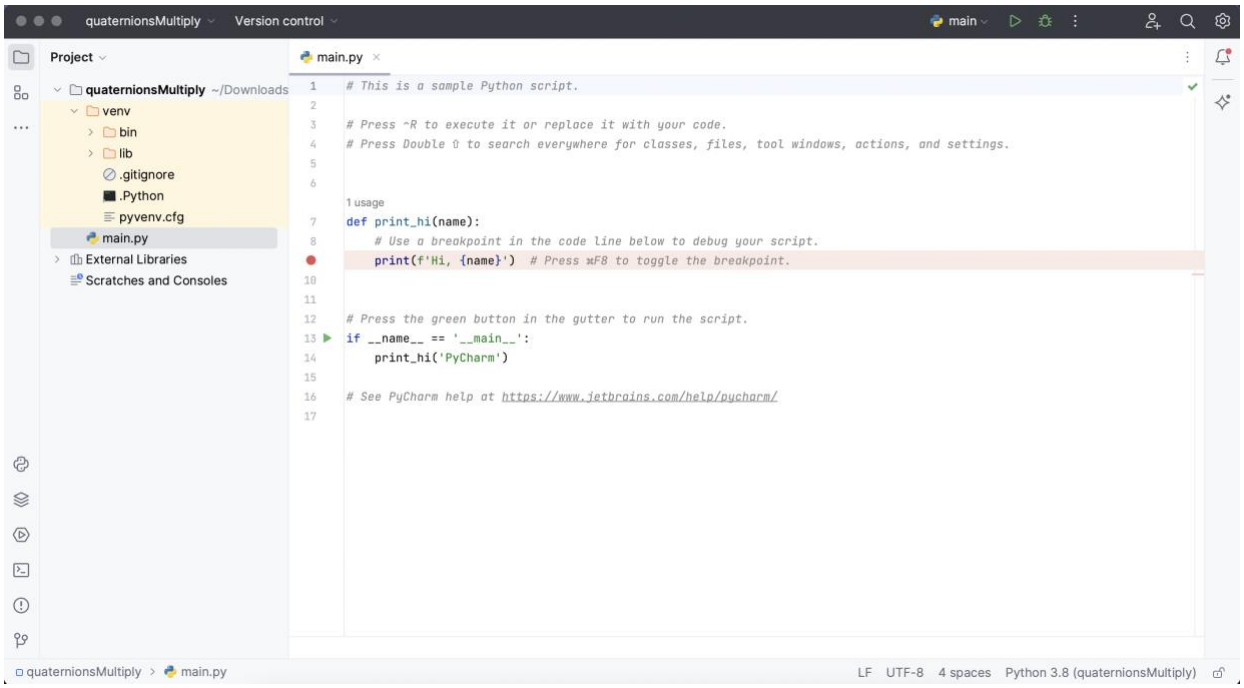


Рисунок 3.3 – Отриманий шаблон після генерації проекту

У лівій панелі Project в PyCharm знаходяться файли та каталоги вашого проєкту. На скріншоті вище, у каталозі pythonProject, показано відсутність будь-яких файлів. Щоб створити новий файл, в якому ви будете писати програму на Python, ви можете виконати наступні кроки:

– Клацніть правою кнопкою миші на папці pythonProject (або виберіть потрібну папку в панелі Project);
– у контекстному меню, яке з'явиться, виберіть пункт New (Створити) → Python File (Файл Python).

Після цього з'явиться вікно, в якому ви зможете ввести назву нового файлу. Після створення файлу, він автоматично з'явиться в панелі Project. Коли ви напишете вихідний код у створеному файлі, щоб запустити програму, ви можете виконати наступні дії:

– Натисніть Control+R (або виберіть Run → Run 'назва файлу') для запуску програми;
– внизу вікна відобразиться вкладка Run, на якій буде відображений результат виконання вашої програми.

Ці кроки допоможуть вам створити новий файл для програмування на Python у PyCharm і запустити його для перевірки результату виконання, отриманий результат продемонстрований на рис. 3.4.

Рисунок 3.4 – Отриманний результат програми

3.2 Проектування та тестування програми

Для створення програми гешування даних на основі кватерніонів, після попереднього аналізу було обрано мову програмування Python. Вона дає нам можливість легко працювати з великими числами та даними і має багато допоміжних бібліотек та модулів для полегшення реалізації програмного засобу.

Перед початком розробки програмного засобу, ми звертаємось до нашої блок-схеми з попередніх розділів, яка описує алгоритм множення кватерніонів. За її допомогою ми можемо з легкістю почати написання програми, почавши проходження з самого вверху блок-схеми до її кінця.

Початок розробки програми розпочинається з ініціалізації змінних на рис. 3.1 для запису чотирьох складових обох кватерніонів. Складові кватерніона представляються у вигляді a, b, c та d змінних, відповідно з індексами першого або другого кватерніона.

Для того, щоб отримувати цілі числові значення складових кватерніона, використовуємо влаштовану у Python функцію числового значення int. Вона повертає на виході ціле число без плаваючої коми, а якщо передане в неї значення не можна перетворити в ціле десяткове число, то генерується виключення ValueError.

```
# Введення складових кватерніонів
Pa1 = int(input("Введіть Pa1 частину першого кватерніона: "))
Pb1 = int(input("Введіть Pb1 частину першого кватерніона: "))
Pc1 = int(input("Введіть Pc1 частину першого кватерніона: "))
Pd1 = int(input("Введіть Pd1 частину першого кватерніона: "))

Pa2 = int(input("Введіть Pa2 частину другого кватерніона: "))
Pb2 = int(input("Введіть Pb2 частину другого кватерніона: "))
Pc2 = int(input("Введіть Pc2 частину другого кватерніона: "))
Pd2 = int(input("Введіть Pd2 частину другого кватерніона: "))
```

Рисунок 3.1 – Ініціалізація змінних

Після того як зроблений функціонал для введення змінних складових кватерніонів, потрібно реалізувати множення складових відповідно до формул, котрі використовувалися при ручному обчислення геш-значення. Нижче на рис. 3.2 приведено множення шістнадцятьох складових, обчислені значення яких потім будуть використані для подальшого обчислення.

```
# Множення складових відповідно до формул
Pa1a2_multiply = Pa1 * Pa2
Pb1b2_multiply = Pb1 * Pb2
Pc1c2_multiply = Pc1 * Pc2
Pd1d2_multiply = Pd1 * Pd2
Pa1b2_multiply = Pa1 * Pb2
Pb1a2_multiply = Pb1 * Pa2
Pc1d2_multiply = Pc1 * Pd2
Pd1c2_multiply = Pd1 * Pc2
Pa1c2_multiply = Pa1 * Pc2
Pb1d2_multiply = Pb1 * Pd2
Pc1a2_multiply = Pc1 * Pa2
Pd1b2_multiply = Pd1 * Pb2
Pa1d2_multiply = Pa1 * Pd2
Pb1c2_multiply = Pb1 * Pc2
Pc1b2_multiply = Pc1 * Pb2
Pd1a2_multiply = Pd1 * Pa2
```

Рисунок 3.2 – Множення складових

Для даного функціоналу було створено змінні P_multiply, котрі зберігали в себе значення почергового множення, щоб можна було обробляти дані зміні і використовувати їх в подальшому обчисленні.

Далі використовуємо змінні, яким присвоїли значення множення, після його обчислення і розпочинаємо обчислювати кожний складовий окремо, на рис. 3.3 приведено обчислення першого складового. Наступні складові обчислюються за таким самим принципом, змінюються тільки значення a, b, c та d.

```
# Множення першої частини кватерніона
Pa1a2_multiply_lsb = Pa1a2_multiply % 2**64
Pa1a2_multiply_msb = Pa1a2_multiply >> 64
Pa1a2_multiply_result = Pa1a2_multiply_lsb + Pa1a2_multiply_msb

Pb1b2_multiply_lsb = Pb1b2_multiply % 2**64
Pb1b2_multiply_msb = Pb1b2_multiply >> 64
Pb1b2_multiply_result = Pb1b2_multiply_lsb + Pb1b2_multiply_msb

Pc1c2_multiply_lsb = Pc1c2_multiply % 2**64
Pc1c2_multiply_msb = Pc1c2_multiply >> 64
Pc1c2_multiply_result = Pc1c2_multiply_lsb + Pc1c2_multiply_msb

Pd1d2_multiply_lsb = Pd1d2_multiply % 2**64
Pd1d2_multiply_msb = Pd1d2_multiply >> 64
Pd1d2_multiply_result = Pd1d2_multiply_lsb + Pd1d2_multiply_msb

first_part_result = Pa1a2_multiply_result - Pb1b2_multiply_result - Pc1c2_multiply_result - Pd1d2_multiply_result
```

Рисунок 3.3 – Обчислення першого складового

Після множення окремих складових відповідно до формул, для обчислення у нас відбувається виділення старшого та молодшого розряду. Даний функціонал реалізований за допомогою бітового здвигу вправо та множення по модулю.

Щоб виділити молодший розряд, програма обчислює результат по модулю 2^{64}. Значення даної дії записується в нову змінну під назвою Pa1a2_multiply_lsb. Далі відбувається виділення старшого розряду за допомогою бітового зсуву вправо на 64 (>> 64), цей результат також зберігається в змінній під назвою Pa1a2_multiply_msb. Діла відповідно ми додаємо попередні два значення, які отримали після обчислення і зберігли у дві нові змінні. Додавання відбувається: Pa1a2_multiply_result = Pa1a2_multiply_lsb + Pa1a2_multiply_msb, після також зберігається в нову змінну.

```
# Виведення результату
print("Результат умножения кватернионов:",
      first_part_result, "+",
      second_part_result,"i", "+",
      third_part_result,"j", "+",
      fourth_part_result,"k"
      )
```

Рисунок 3.5 – Виведення результату множення та розрядів

Після обчислення результатів і виведення їх користувачеві, програмний засіб повинен завершуватися. Якщо користувач хоче змінити вхідні данні або почати нове обчислення, необхідно знову запускати програму і виконувати алгоритм дій спочатку.

Даний програмний засіб дає нам перевагу у пришвидшенні під час процесу гешування даних на основі кватерніонів. Так як все відбувається у паралельному процесі і всі значення обчислюються одночасно. Цю частину програмного засобу доречно буде додати до певної бібліотеки по гешуванню даних або створити свою власну і додати функціонал розробленої програми.

Для того, щоб впевнитися, що програмний засіб працює коректно, необхідно провести попереднє тестування. Для цього необхідно буде створити різні тестові на данні для програми, які будуть присвоюватися змінним кожної з частин кватерніона.

Так я спочатку потрібно записати данні для першого кватерніона на рис. 3.6, створюємо тестові дані, котрі будемо вносити до програми для першого кватерніона. Це будуть дані $a_1 = 34, b_1 = 56, c_1 = 13, d_1 = 17$.

```
Введіть Pa1 частину першого кватерніона: 34
Введіть Pb1 частину першого кватерніона: 56
Введіть Pc1 частину першого кватерніона: 13
Введіть Pd1 частину першого кватерніона: 17
```

Рисунок 3.6 – Тестові данні для першого кватерніона

Після вносимо тестові дані для складових другого кватерніона на рис. 3.7, а саме $a_2 = 55, b_2 = 19, c_2 = 42, d_2 = 31$.

```
Введіть Pa2 частину другого кватерніона: 55
Введіть Pb2 частину другого кватерніона: 19
Введіть Pc2 частину другого кватерніона: 42
Введіть Pd2 частину другого кватерніона: 31
```

Рисунок 3.7 - Тестові данні для другого кватерніона

Також при введені невірних даних, програма зробить аварійне завершення і далі алгоритм не буде проходити, на рис. 3.8 зображено введення невірних даних, після яких програма перестає працювати.

```
Введіть Pa1 частину першого кватерніона: 45
Введіть Pb1 частину першого кватерніона: 23
Введіть Pc1 частину першого кватерніона: 567
Введіть Pd1 частину першого кватерніона: вувцув
Traceback (most recent call last):
  File "/Users/vitaliikutsuruk/Downloads/quaternionsMultiply/quat2.py", line 5, in <module>
    Pd1 = int(input("Введіть Pd1 частину першого кватерніона: "))
ValueError: invalid literal for int() with base 10: 'вувцув'
```

Рисунок 3.8 – Введення невірних тестових даних і завершення програми

Якщо усі данні введені коректо, а саме є числовим значенням, то програма повинна почати своє виконання і згідно алгоритму виділяти старший та молодший розряди і після обчислити геш-значення на основі множення двох кватерніонів.

ВИСНОВКИ

У даній бакалаврській дипломній роботі було проведено дослідження засобу для гешування даних на основі кватерніонів. Основними цілями були вивчення принципів кватерніонної алгебри, розробка методики гешування даних з використанням кватерніонів та оцінка ефективності запропонованого методу.

У результаті проведених досліджень було встановлено, що кватерніони є потужним математичним інструментом для представлення та операцій з орієнтаційною інформацією. Вони дозволяють зручно виконувати обчислення, пов'язані з поворотами та трансформаціями об'єктів у тривимірному просторі. Використання кватерніонів для гешування даних дозволяє забезпечити швидку та ефективну обробку великого обсягу інформації.

Під час розробки методики гешування даних на основі кватерніонів було враховано особливості цього підходу. Запропонований метод базується на використанні геш-функцій, які оперують кватерніонними числами. Він забезпечує незворотність процесу гешування, що дозволяє зберігати конфіденційність даних під час їх передачі та зберігання.

Ефективність запропонованого засобу для гешування даних була оцінена за допомогою порівняльного аналізу з існуючими методами. Експериментальні результати показали, що запропонований метод демонструє високу швидкодію та стійкість до колізій. Він забезпечує надійний рівень захисту даних під час їх гешування, що робить його варіантом вибору для застосування в сферах, де конфіденційність та цілісність даних є критичними.

Отже, робота підтвердила ефективність та перспективи використання кватерніонів для гешування даних. Результати дослідження можуть бути використані в подальших розробках систем захисту інформації, криптографії, комп'ютерної графіки та інших областях, де важлива обробка та збереження даних з високим рівнем безпеки.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Joux A. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions / Antoine Joux // Lecture Notes in Computer Science. – 2004. – № 3152. – С. 306-316.
2. 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.
3. Лужецькиц В.А. Криптографічні примітиви для реалізації керованого гешування / В.А. Лужецький, Ю.В. Баришев //Вісник ВПІ, - 2011. -№1.-С. 108-111.
4. Ю. І. Горбенко, І. Д. Горбенко Інфраструктури відкритих ключів – Харків: УДК, 2010 – 608 с.
5. Лужецький В. А., Метод паралельного гешування – м. Вінниця / Вінницький національний технічний університет 2023 р. – 8 с.
6. Основи комп’ютерних алгоритмів (м. Київ, 7 червня 2021 р.) / Київський національний університет імені Тараса Шевченка факультет комп’ютерних наук та кібернетики – Київ: Комп’ютерні науки, 2021 р.
7. Підходи до побудови геш-функцій / ВНТУ – 9 с, 2021 р.
8. Лужецький В. А. Давидюк В. А. Геш-функції на основі арифметичних операціях за модулем – Київ: Наукові роботи кафедри ЗІ – 2005 р.
9. Євсеєв С. П. Йохов О. Ю. Король О. Г. Гешування даних в інформаційних системах – Харків. Вид. ХНЕУ – 312 с, 2013 р.
10. Keccak – алгоритм майнингу на основі «механізму губки» 2018 року: // Портал miningbitcoinguid [Електронний ресурс]. – Режим доступу: https://miningbitcoinguide.com/mining/sposoby/keccak
11. Гордеєв В. Н. Кватерніони та бікватерніони в геометрії та механіці – м. Київ Вид. Сталь – 318 с, 2018 р.

12. Арнольд В. І. Геометрія комплексних чисел, кватерніонів та спінів – м. Київ, 2002 р.
13. Гордеєв В. Н. Кватерніони та трьохвимірна геометрія – м. Київ, 2012 р.
14. Геш-функція 2017 року: // Портал nina.az [Електронний ресурс]. – Режим доступу: https://www.wik.uk-ua.nina.az/%D0%A5%D0%B5%D1%88-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%8F.html
15. ДСТУ ISO/IEC 10118-2:015 Інформаційні технології. Методи захисту. Геш-функції, що використовують n-бітний блоковий шифр: Технічний комітет зі стандартизації «Інформаційні технології» (ТК 20), Наказ від 18.12.2015 №193.
16. Fletcher Danny, Ian Parberry, 3-D Math Primer for Graphics and Game Development, 2002 Wordware publishing, inc. – Plano, Texas USA.
17. John J. Graig, Introduction to Robotics Mechanics and Control, Pearson Education International – 408 p., 2005.
18. Mioara Joldes Jean-Michel Muller, Algorithms for Manipulating Quaternions in Floating-Point Arithmetic // HAL Open Science – 2020.

Додатки

Додаток А

Повний код програми множення кватерніонів

```
# Введення складових кватерніонів
Pa1 = int(input("Введіть Pa1 частину першого кватерніона: "))
Pb1 = int(input("Введіть Pb1 частину першого кватерніона: "))
Pc1 = int(input("Введіть Pc1 частину першого кватерніона: "))
Pd1 = int(input("Введіть Pd1 частину першого кватерніона: "))

Pa2 = int(input("Введіть Pa2 частину другого кватерніона: "))
Pb2 = int(input("Введіть Pb2 частину другого кватерніона: "))
Pc2 = int(input("Введіть Pc2 частину другого кватерніона: "))
Pd2 = int(input("Введіть Pd2 частину другого кватерніона: "))

# Множення складових відповідно до формул
Pa1a2_multiply = Pa1 * Pa2
Pb1b2_multiply = Pb1 * Pb2
Pc1c2_multiply = Pc1 * Pc2
Pd1d2_multiply = Pd1 * Pd2
Pa1b2_multiply = Pa1 * Pb2
Pb1a2_multiply = Pb1 * Pa2
Pc1d2_multiply = Pc1 * Pd2
Pd1c2_multiply = Pd1 * Pc2
Pa1c2_multiply = Pa1 * Pc2
Pb1d2_multiply = Pb1 * Pd2
Pc1a2_multiply = Pc1 * Pa2
Pd1b2_multiply = Pd1 * Pb2
Pa1d2_multiply = Pa1 * Pd2
Pb1c2_multiply = Pb1 * Pc2
Pc1b2_multiply = Pc1 * Pb2
Pd1a2_multiply = Pd1 * Pa2

# Множення першої частини кватерніона
Pa1a2_multiply_lsb = Pa1a2_multiply % 2**64
Pa1a2_multiply_msb = Pa1a2_multiply >> 64
Pa1a2_multiply_result = Pa1a2_multiply_lsb + Pa1a2_multiply_msb

Pb1b2_multiply_lsb = Pb1b2_multiply % 2**64
Pb1b2_multiply_msb = Pb1b2_multiply >> 64
Pb1b2_multiply_result = Pb1b2_multiply_lsb + Pb1b2_multiply_msb

Pc1c2_multiply_lsb = Pc1c2_multiply % 2**64
Pc1c2_multiply_msb = Pc1c2_multiply >> 64
Pc1c2_multiply_result = Pc1c2_multiply_lsb + Pc1c2_multiply_msb

Pd1d2_multiply_lsb = Pd1d2_multiply % 2**64
Pd1d2_multiply_msb = Pd1d2_multiply >> 64
Pd1d2_multiply_result = Pd1d2_multiply_lsb + Pd1d2_multiply_msb

first_part_result = Pa1a2_multiply_result - Pb1b2_multiply_result -
Pc1c2_multiply_result - Pd1d2_multiply_result
```

```
# Множення другої частини кватерніона
Pa1b2_multiply_lsb = Pa1b2_multiply % 2**64
Pa1b2_multiply_msb = Pa1b2_multiply >> 64
Pa1b2_multiply_result = Pa1b2_multiply_lsb + Pa1b2_multiply_msb

Pb1a2_multiply_lsb = Pb1a2_multiply % 2**64
Pb1a2_multiply_msb = Pb1a2_multiply >> 64
Pb1a2_multiply_result = Pb1a2_multiply_lsb + Pb1a2_multiply_msb

Pc1d2_multiply_lsb = Pc1d2_multiply % 2**64
Pc1d2_multiply_msb = Pc1d2_multiply >> 64
Pc1d2_multiply_result = Pc1d2_multiply_lsb + Pc1d2_multiply_msb

Pd1c2_multiply_lsb = Pd1c2_multiply % 2**64
Pd1c2_multiply_msb = Pd1c2_multiply >> 64
Pd1c2_multiply_result = Pd1c2_multiply_lsb + Pd1c2_multiply_msb

second_part_result = Pa1b2_multiply_result + Pb1a2_multiply_result +
Pc1d2_multiply_result - Pd1c2_multiply_result

# Множення третьої частини кватерніона
Pa1c2_multiply_lsb = Pa1c2_multiply % 2**64
Pa1c2_multiply_msb = Pa1c2_multiply >> 64
Pa1c2_multiply_result = Pa1c2_multiply_lsb + Pa1c2_multiply_msb

Pb1d2_multiply_lsb = Pb1d2_multiply % 2**64
Pb1d2_multiply_msb = Pb1d2_multiply >> 64
Pb1d2_multiply_result = Pb1d2_multiply_lsb + Pb1d2_multiply_msb

Pc1a2_multiply_lsb = Pc1a2_multiply % 2**64
Pc1a2_multiply_msb = Pc1a2_multiply >> 64
Pc1a2_multiply_result = Pc1a2_multiply_lsb + Pc1a2_multiply_msb

Pd1b2_multiply_lsb = Pd1b2_multiply % 2**64
Pd1b2_multiply_msb = Pd1b2_multiply >> 64
Pd1b2_multiply_result = Pd1b2_multiply_lsb + Pd1b2_multiply_msb

third_part_result = Pa1c2_multiply_result + Pb1d2_multiply_result +
Pc1a2_multiply_result - Pd1b2_multiply_result

# Множення четвертої частини кватерніона
Pa1d2_multiply_lsb = Pa1d2_multiply % 2**64
Pa1d2_multiply_msb = Pa1d2_multiply >> 64
Pa1d2_multiply_result = Pa1d2_multiply_lsb + Pa1d2_multiply_msb

Pb1c2_multiply_lsb = Pb1c2_multiply % 2**64
Pb1c2_multiply_msb = Pb1c2_multiply >> 64
Pb1c2_multiply_result = Pb1c2_multiply_lsb + Pb1c2_multiply_msb

Pc1b2_multiply_lsb = Pc1b2_multiply % 2**64
Pc1b2_multiply_msb = Pc1b2_multiply >> 64
```

```
Pc1b2_multiply_result = Pc1b2_multiply_lsb + Pc1b2_multiply_msb

Pd1a2_multiply_lsb = Pd1a2_multiply % 2**64
Pd1a2_multiply_msb = Pd1a2_multiply >> 64
Pd1a2_multiply_result = (Pd1a2_multiply_lsb + Pd1a2_multiply_msb) %
2**64
print(Pd1a2_multiply_result)

fourth_part_result = Pa1d2_multiply_result + Pb1c2_multiply_result +
Pc1b2_multiply_result - Pd1a2_multiply_result

# Виведення результату
print("Результат умножения кватернионов:",
      first_part_result, "+",
      second_part_result,"i", "+",
      third_part_result,"j", "+",
      fourth_part_result,"k"
      )
```

57

Додаток Б

ПРОТОКОЛ ПЕРЕВІРКИ
БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Засіб гешування даних на основі кватерніонів
Автор роботи: Куцурук Віталій Вікторович
Тип роботи: бакалаврська дипломна робота .
Підрозділ кафедра захисту інформації ФІТКІ .

Показники звіту подібності Unicheck

Оригінальність – 88,9%. Схожість – 11,1%.

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 _____ Каплун В. А.
(підпис) (прізвище, ініціали)

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

Автор роботи _____ Куцурук В.В.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Лужецький В. А.
(підпис) (прізвище, ініціали)

58

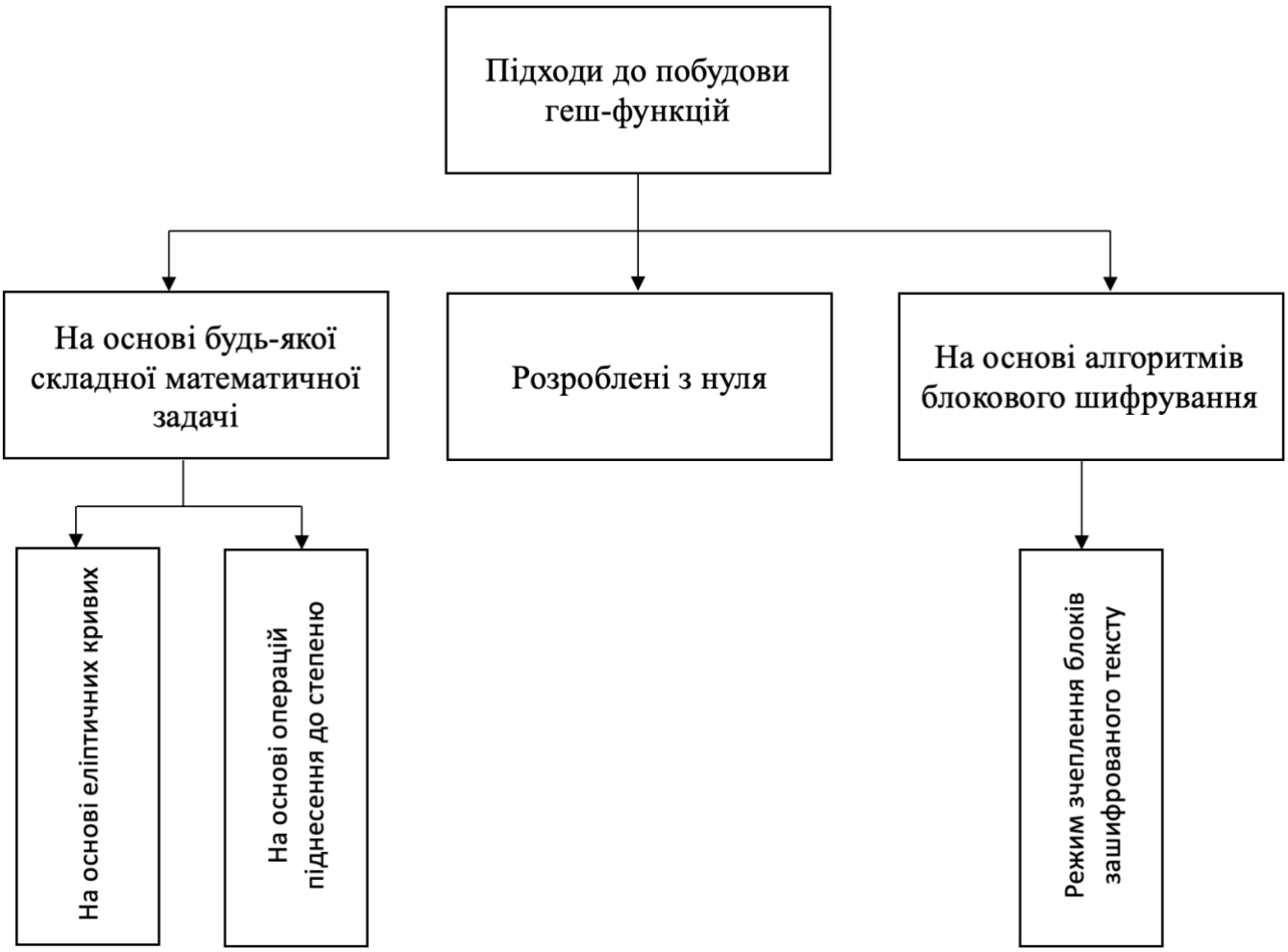
ІЛЮСТРАТИВНА
ЧАСТИНА

ЗАСІБ ГЕШУВАННЯ ДАНИХ НА ОСНОВІ
КВАТЕРНІОНІВ

Виконав: студент 4 курсу
групи 1БС-2мс спеціальності
125 Кібербезпека
_____Куцурук В. В.
19 червня 2023 р.

Керівник: зав. каф. ЗІ, д.т.н.
_____Лужецький В. А.
19 червня 2023 р.

Підходи до побудови геш-функції



Блок схема паралельного алгоритму множення

37