КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра математичної інформатики

Кваліфікаційна робота
на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:
ХЕШ-ФУНКЦІЇ ТА ЇХ ЗАСТОСУВАННЯ

Виконав студент 4-го курсу
Нікіта КОШМАН _____
(підпис)

Науковий керівник:
професор, доктор фіз.-мат. наук
Анатолій АНІСІМОВ _____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри математичної інформатики

«____»_____2023 р., протокол № ____

Завідувач кафедри

Терещенко В. М. _____

(підпис)

Київ - 2023

РЕФЕРАТ

Обсяг роботи 55 сторінок, 7 ілюстрацій, 24 джерела посилань, 3 додаток.

ХЕШУВАННЯ, БЛОКЧЕЙН, ХЕШ-ФУНКЦІЯ, GOLANG, SHA256.

Об’єктом роботи є хеш-функції, застосування їх у сучасному світі та розроблення блокчейн технологія.

Метою роботи є створення блокчейн технології, яку можна бути використовувати в комерційних цілях .

Інструменти, що були використані під час розроблення: інтегроване середовище розробки: Microsoft Visual Studio Code; хеш-функція SHA256.

Результати роботи: розроблена програма хешування даних, перенесення даних в блоки, для подальшого з’єднання в блокчейн. Програма були успішно протестовані у реальних умовах, де вони довели свою працездатність та ефективність.

Розроблений блокчейн може використовуватись для забезпечення безпеки та автентичності електронних документів, захисту паролів, контролю цілісності даних, а також у сферах фінансів, логістики, ліцензування та управління правами.

ЗМІСТ

ЗМІСТ……... 3
ВСТУП……. 5
СКОРОЧЕННІ ТА УМОВНІ ПОЗНАЧКИ……... 7
ОСНОВНА ЧАСТИНА……... 8
1. ТЕОРИТИЧНА ЧАСТИНА ……...……. 8
1.1. Основні поняття та терміни ……..... 8
1.2. Основи побудови хеш-функції……..... 13
1.3. Типи хеш-функцій …….……... 15
1.3.1 Хеш-функція на основі ділення……..……. 15
1.3.2 Мультиплиплікативна схема хешування…… 17
1.3.3 Хешування рядків змінної довжини……... 18
1.4.Криптографічні хеш-функції та їх застосування …….. 20
1.5.SHA256…….. 22
1.6.Атаки на хеш-функції та заходи щодо їх захисту…… 24
2. ЗАСТОСУВАННЯ ХЕШ-ФУНКЦІЙ…….. 30
2.1.Аутентифікація повідомлень……. 30
2.2.Цифровий підпис…… 32
2.3.Захист паролів…… 33
2.4.Блокчейн технології……... 34
3. ПРАКТИЧНА ЧАСТИНА…… 38

3.1 Загальна ідея…….…….. 38

3.2 Опис проекту блокчейн ..…….. 39

3.3 Деталі реалізації проекту…….. 40

3.4 Результат роботи програми……. 45

ВИСНОВКИ…….. 47

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ……..... 50

ДОДАТОК……. 51

ВСТУП

Оцінка сучасного стану об’єкта розробки. Хеш-функції є важливим інструментом для безпеки інформації та застосовуються в багатьох галузях, таких як криптографія, бази даних, комп’ютерна мережа, ігри та пошук інформації. Вони забезпечують цілісність даних, аутентифікацію користувачів, захист від фальсифікації та багато інших функцій. На сьогоднішній день висока ефективність застосування хеш-функцій має велике значення для безпеки та ефективності різноманітних систем. Однак, не дивлячись на широке використання, безпека хеш-функцій залишається однією з найбільш важливих проблем криптографії, і підвищення ефективності їх застосування може мати значний вплив на безпеку та ефективність різних систем.

З ростом кількості інформації, яка зберігається та передається через мережу Інтернет, зростає потреба у захисті цієї інформації від несанкціонованого доступу та модифікації. Хеш-функції є ефективним інструментом для захисту даних від цих загроз, тому дослідження їх властивостей та застосування є важливим завданням у галузі інформаційної безпеки. З урахуванням широкого використання хеш-функцій у сучасному світі, важливо досліджувати їх властивості та потенційні слабкі місця, щоб розробляти нові технології та алгоритми, які можуть забезпечувати більш високий рівень безпеки. Також важливо вивчити та описати практичні застосування хеш-функцій в сфері інформаційної безпеки та інших сферах, де вони можуть знайти своє застосування.

Актуальність роботи та підстави для її виконання. У сучасному цифровому світі, де обмін даними стає все більш інтенсивним, забезпечення безпеки та цілісності даних є критично важливим завданням. Хеш-функції, завдяки своїй властивості незмінності та стійкості до змін, стали незамінним

інструментом для забезпечення безпеки і ідентифікації даних. Актуальність цієї роботи полягає в розробці та дослідженні нових підходів до хеш-функцій, зокрема в контексті реалізації блокчейн-технологій, які потребують швидкої та надійної обробки великих обсягів даних.

Мета й завдання роботи. Мета кваліфікаційної роботи полягає є розробка та реалізація блокчейн технології з використанням мови програмування Golang. Основні завдання роботи включають:

- Вивчення теоретичних аспектів хеш-функцій та їх ролі в забезпеченні безпеки даних.
- Аналіз існуючих хеш-функцій та їхніх переваг та недоліків.
- Розробка та реалізація власної блокчейн технології на мові програмування Golang.
- Тестування та оцінка ефективності розробленої блокчейн технології.

Об’єкт, методи й засоби розроблення. Об’єктом роботи є є хеш-функції та їх застосування в контексті блокчейн-технологій. Консольний додаток написан на мові програмування Golang.

Використовувались такі інструментальні засоби, бібліотеки та технології: Microsoft Visual Studio Code, SHA256, BargedDB,crypto , HTTP.

Можливі сфери застосування.Хеш-функції мають широке спектр застосувань у різних галузях. Вони використовуються в криптографії, цифрових підписах, безпеці мереж та зберіганні даних. Також хеш-функції знаходять своє застосування у блокчейн-технологіях, де вони гарантують безпеку, цілісність та недублюваність блоків даних.

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

HMAC-механізм перевірки цілісності інформації, що передається або зберігається в ненадійному середовищі.

SHA-2 — безпечний алгоритм хешування, версія 2

CRC - циклічний надлишковий код

ГОСТ 34.11 – криптографічний стандарт обрахування хеш-функцій

MD5 – 128-бітовий алгоритм хешування

1. ТЕОРИТИЧНА ЧАСТИНА

1.1 Основні поняття та терміни

Одними із ключових аспектів криптографії та інформаційної безпеки є основні поняття та терміни хеш-функції. Дослідження цих понять допоможе зрозуміти принципи роботи хеш-функцій та їх використання в різних сферах, включаючи криптографію, бази даних, електронну підпис та багато інших.

Хешування — це перетворення вхідного масиву даних довільної довжини у вихідний бітовий рядок фіксованої довжини. Такі перетворення також називаються хеш-функціями, або функціями згортання, а їхні результати називають хешем, хеш-кодом, хеш-сумою, або дайджестом повідомлення.

Хешування знаходить широке застосування у різних сферах інформаційної технології. Вони використовуються для пошуку дублікатів, створення унікальних ідентифікаторів, побудови асоціативних масивів та зберігання паролів з високим рівнем безпеки. Крім того, хеш-функції використовуються для контрольного підсумовування даних з метою виявлення помилок та змін, а також для створення електронних підписів для підтвердження автентичності повідомлень.

Хеш-таблиця - це структура даних, яка реалізує інтерфейс асоціативного масиву, що дозволяє зберігати пари "ключ-значення" та виконувати три основні операції: додавання нової пари, пошук та видалення пари за заданим ключем. В основі хеш-таблиці лежить масив, який формується у певному порядку за допомогою хеш-функції.

1) функція має бути простою з обчислювальної точки зору;

2) функція повинна розподіляти ключі у хеш-таблиці найбільш рівномірно;

3) функція не повинна відображати будь-який зв'язок між значеннями ключів у зв'язок між значеннями адрес;

4) функція повинна мінімізувати кількість колізій.

Хеш-таблиці повинні задовольняти наступні вимоги:

1) Операції у хеш-таблиці починаються з обчислення хеш-функції від ключа, і отримане хеш-значення використовується як індекс для доступу до відповідного елемента у внутрішньому масиві.

2) Коефіцієнт заповнення хеш-таблиці, який визначається як відношення кількості елементів до кількості можливих значень хеш-функції, є важливим параметром. Він впливає на середній час виконання операцій у таблиці.

3) Операції пошуку, вставки та видалення мають виконуватися у середньому за час O(1), що означає постійний час. Проте ця оцінка не враховує можливі апаратні витрати на перебудову індексу хеш-таблиці при збільшенні її розміру або додаванні нових пар.

4) Механізм вирішення колізій є необхідною складовою хеш-таблиці.

Хешування є цінним інструментом у разі необхідності зберегти широкий діапазон можливих значень в обмеженому обсязі пам'яті та забезпечити швидкий доступ до даних. Хеш-таблиці широко використовуються в базах даних, а також в мовних процесорах, таких як компілятори і асемблери, де вони підвищують швидкість обробки таблиць ідентифікаторів. Прикладами хешування в повсякденному житті включають розподіл книг у бібліотеці за тематичними каталогами, упорядкування

словників за першими буквами слів, шифрування спеціальностей у вузах та багато іншого.

Хеш-функція – це математична функція, яка перетворює числове вхідне значення в інше стиснене числове значення. Вхід у хеш-функцію є довільної довжини, але вихід завжди має фіксовану довжину. Значення, що повертаються хеш-функцією, називаються дайджестом повідомлення або просто хеш-значеннями. Хеш-функція застосовують для перевірки цілісності повідомлень та файлів, генерація і перевірки цифрового підпису, перевірки пароля та ідентифікатор файлу або даних.

Основні поняття та терміни, що пов’язані з хеш-функціями, включають:

1) Хеш-значення: Фіксований вихідний код, отриманий внаслідок застосування хеш-функції до вхідних даних. Хеш-значення зазвичай представлено у вигляді фіксованої довжини бітів та відображає великий обсяг вхідних даних у вигляді випадкової послідовності бітів фіксованої довжини.

2) Колізія: випадок, коли дайджест одного повідомлення є ідентичним до дайджесту іншого. Це пов’язано з тим, що при визначеній довжині дайджесту, існує невизначена кількість можливих повідомлень. Колізії є небажаною, оскільки вони можуть призвести до помилок у програмах, які використовують хеш-функції.

3) Дайджест: вихідний результат роботи хеш-функції. Це послідовність бітів фіксованого розміру, яка представляє унікальний відбиток даних, що були введені до хеш-функції.

4) Солі: довільний рядок, який додається до вхідних даних перед застосуванням хеш-функції. Використання солі може унеможливити атаки на хеш-функції, такі як словникові атаки.

5) Хешуванням даних: процес, при якому функція хешу покриває

дані довільної довжини до фіксованої довжини.

Хеш-функціям притаманні наступні ознаки:

1) Дайджест фіксованої довжини. Кожна хеш функція може приймати на вході повідомлення довільної довжини, та на виході видавати дайджест однієї, чітко визначеної довжини.

2) Дайджест може мати будь-яку довжину. Якщо розглядати на прикладах найпопулярніших імплементацій хеш-функцій (SHA-1, MD-5, SHA-2), то довжина блоку становить зазвичай 160, 256 та 512 біт. Однак варто відзначити, що одна хеш-функція може працювати у різних режимах, маючи різну довжину дайджесту.

З попередньої властивості випливає те, що вихідні дані зазвичай містять меншу кількість інформації, отже хеш-функції також називають функціями стиснення.

Для використання хеш-функції як методу захисту інформації, вона повинна відповідати певним вимогам:

1) Дайджести хеш-функції мають бути рівномірно розподілені, тобто хеш значення має генеруватися у вихідному діапазоні з однаковою рівномірністю. З цього випливає, що діапазоном є усі значення дайджестів. Кожний біт дайджесту повинен генеруватися з вірогідністю, що наближається до 50%.

2) Хеш-функція повинна приймати дані будь-якого формату, зазвичай алгоритми хешування приймають масив байтів, що забезпечує їх універсальність.

3) Алгоритм хешування не повинен використовувати генератори псевдовипадкових чисел, або певні вхідні дані, які неможливо отримати при повторному використанні алгоритму. Оскільки хеш-функція є детерміністичною, вхідне повідомлення завжди генерує один дайджест.

Також не повинно мати значення адреси 156 пам’яті об’єкта. Припускається використання зерна генерації або вектору ініціалізації.

Хеш-функція широко використовується у програмному забезпеченні для швидкого пошуку даних в хеш-таблицях, які є структурами даних. За допомогою хеш-функції, записи з однаковими значеннями можуть бути груповані разом, що дає оптимізацію таблиць та баз даних. У великих файлах такий підхід шукає дублікати даних ефективно. Криптографічні хеш-функції дозволяють перевірити, які саме вхідні дані породжують заданий хеш, що є корисним для перевірки цілісності переданих даних та надання автентифікації повідомлень за допомогою HMACs. У випадку, коли вхідні дані невідомі, значення хеш-функції не дозволяє легко відновити вхідні дані, що є важливою характеристикою криптографічних хеш-функцій.

У випадку, коли кількість можливих значень хеш-функції менша, ніж кількість варіантів значень вхідного масиву, то існує можливість, що два різні масиви можуть мати однаковий хеш-код, тобто виникає колізія. Такі колізії є досить поширеними, оскільки існує велика кількість різних масивів з різним вмістом, які можуть мати однаковий хеш-код. Ймовірність виникнення колізій грає важливу роль у визначенні якості хеш-функцій.

На сьогоднішній день існує широкий спектр різних алгоритмів хешування, які відрізняються за розрядністю, криптостійкістю, обчислювальною складністю та іншими характеристиками. Вибір конкретної хеш-функції залежить від особливостей задачі, яку потрібно вирішити. Контрольна сума або CRC можуть служити простими прикладами хеш-функцій, які використовуються, наприклад, для обчислення контрольної суми даних.

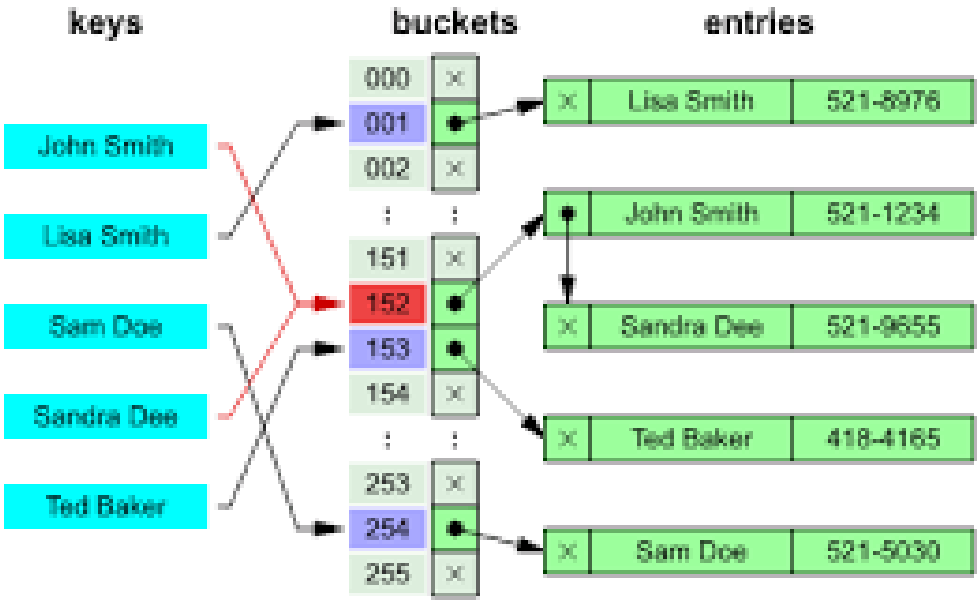

Рисунок 1.1 – Приклад колізії між «John Smith» та «Sandra Dee», яким відповідає одне значення

Хеш-функції можуть бути поєднані з різними технологіями, такими як рандомізація функцій, контрольні суми, відбитки пальців, контрольні цифри, шифри та коди для виправлення помилок. Часто вони помилково сплутуються між собою. Кожна з цих технологій має свою власну область застосування і може бути оптимізована та розроблена по-різному, відповідно до вимог конкретної задачі. Хоча ці поняття подібні одне до одного в певній мірі, вони мають різні характеристики і особливості.

1.2 Основи побудови хеш-функції

Одним із основних підходів до побудови хеш-функцій є ітеративна послідовна схема. У цій схемі основне перетворення здійснюється над блоком даних розміром *k* біт та призводить до результату фіксованої розрядності *n* біт. Значення n є довжиною вихідний хеш-функції, а *k* - довільне число, що перевищує n. Базове перетворення має мати всі

властивості хеш-функції, такі як незворотність і неможливість інваріантної зміни вхідних даних.

Процес хешування виконується з використанням проміжної змінної фіксованої розрядності n біт. Початкове значення цієї змінної вибирається довільно та є відомим для всіх сторонам, наприклад, може бути рівним нулю.

Вхідні дані розбиваються на блоки розміром (k-n) біт. На кожній ітерації хешування наступна (k-n)-бітна порція вхідних даних поєднується з проміжною змінною, отриманою на попередній ітерації, і проходить через базове перетворення. Кожна ітерація сприяє перемішування всього вхідного тексту з початковим значенням проміжної змінної. Значення проміжної змінної після останньої ітерації є остаточним значенням хеш-функції. Іноді над цим значенням виконуються додаткові перетворення, але якщо базове перетворення досить стійке, такі додаткові кроки можуть бути зайвими.

При проектуванні хеш-функції за ітеративною схемою виникають питання, пов'язані з обробкою даних, які не є кратними (k-n) бітами, а також включенням довжини документа в хеш-суму, якщо це потрібно. Існує два підходи до вирішення цих питань. Перший підхід включає початок документа фіксоване поле довжини (наприклад, 32 біта), в якому записується довжина тексту в двійковій формі. Потім об'єднаний блок даних доповнюється нулями до найближчого розміру кратного (k-n) біт. У другому підході документ доповнюється одним бітом "1" справа, та був доповнюється нулями до найближчого кратного розміру (k-n) біт. У цьому випадку поле довжини стає зайвим, оскільки після вирівнювання по межі блоків ні два різні документи не матимуть однакового результату.

Крім однопрохідних алгоритмів хешування, існують також багатопрохідні алгоритми. У цих алгоритмах вхідний блок даних повторюється кілька разів на етапі розширення, а потім доповнюється до найближчого розміру блоку.

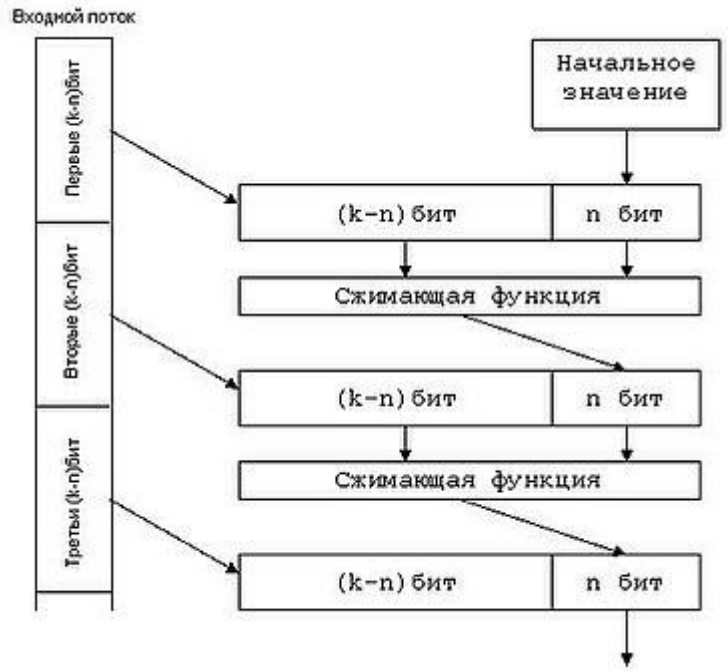

Рис. 1.2.1 – інтерактивна хеш-функція

1.3 Типи хеш-функцій

Гарна хеш-функція повинна відповідати двом властивостям:

- швидко обчислюватися;

- мінімізувати кількість колізій.

Функція з M є прикладом «поганої» хеш-функції, яка десятизначному натуральному числу k співставляє три цифри, що були обрані з середини двадцятизначного квадрата числа k.

$\forall k0 \leq h(k) < M$,

де $k0 = 0$, $M = 1000$;

$h(k)$ має якесь значення між нулем та не більше ніж 1000.

Якщо ключі не мають значної кількості нулів в кінці або початку, тоді цей підхід може бути застосований до реальних даних, де значення хеш-кодів повинні розподілятися рівномірно від "000" до "999".

Проте є інші прості та надійні методи, на яких базується багато хеш-функцій.

1.3.1 Хеш-функції на основі ділення

Перший метод полягає у тому, що як хеш застосовується залишок від ділення на M, де M — це кількість всіх можливих хеші:

$h(k) = K \ mod \ M$,

де $K = \boldsymbol{k_{n-1}k_{n-2}} \ldots \boldsymbol{k_0}$ – це бінарні ключі

Тому очевидно, що якщо M є парним числом, то значення функції також буде парним при парному K. Переміщення даних у файлах буде суттєвим, якщо як M, так і K є непарними числами. Не рекомендується використовувати систему числення комп'ютера як базу M, оскільки хеш-код буде залежати лише від кількох цифр числа K, і ці цифри будуть розташовуватися праворуч, що може призвести до значної кількості колізій. У практиці зазвичай використовують просте число M, що є повністю прийнятним в більшості випадків.

Варто згадати про метод хешування, який використовує ділення на поліном по модулю два. У цьому методі значення M також повинно бути степенем двійки, а поліноми представлені у вигляді бінарних ключів K. У цьому випадку хеш-код може бути визначений як коефіцієнти полінома, які отримуються шляхом залишку від ділення K на попередньо обраний поліном P степені m:

$h(x) = K(x) \ mod \ P(x)$,

де $h(x) = \boldsymbol{h_{m-1}h_{m-2}\ \ldots\ h_0}$.

Якщо обрати $P(x)$ правильно, то цей спосіб гарантує відсутність колізій між майже однаковими ключами.

1.3.2 Мультиплікативна схема хешування

Обрання деякої цілої константи *A*, взаємно простої з ω , тобто кількості можливих варіантів значень у вигляді машинного слова є другим методом. Тоді виникає можливість взяти хеш-функцію виду:

$$h(K) = [M \lfloor \frac{\boldsymbol{A}}{\boldsymbol{\omega}} * K \rfloor],$$

У цьому випадку M представляє собою степінь двійки, а h(K) формується зі старших бітів правої половини добутку A * K в двійковій системі числення на комп'ютері. Перевагами цих двох методів є використання того факту, що реальні ключі не є випадковими. Наприклад, якщо ключі утворюють арифметичну прогресію, мультиплікативний метод відображення перетворить цю арифметичну прогресію на наближену арифметичну прогресію різних хеш-значень. Це знизить кількість колізій в порівнянні з випадковим розподілом ключів.

Один з варіантів цього методу - це хешування за допомогою чисел Фібоначчі, яке ґрунтується на властивостях золотого перетину. У цьому випадку A вибирається як найближче ціле число до $\boldsymbol{\varphi^{-1}} * \omega$, де φ - золотий перетин, а ω - просте число, взаємно просте з ω.

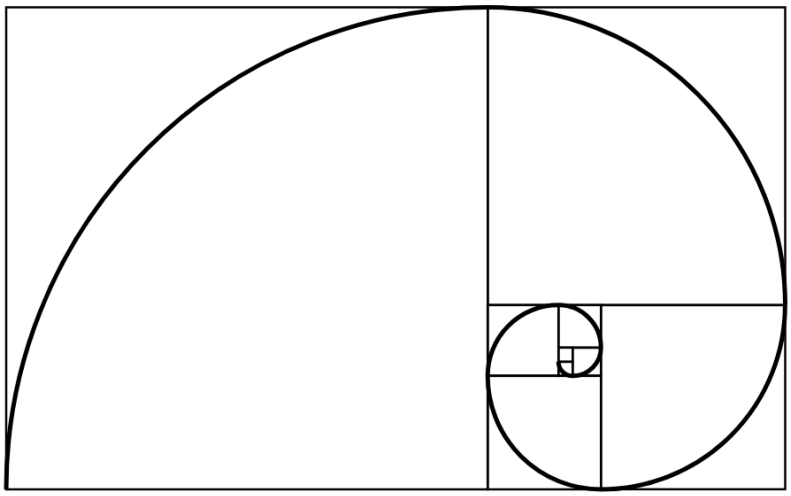

Рисунок 1.3.2.1 – Спіраль Фібоначчі, апроксимація золотої спіралі, що утворена дугами та проведені через протилежні кути квадратів Фібоначчі

1.3.3 Хешування рядків змінної довжини

Методи, які були описані вище, є корисними, коли ми маємо справу з ключами, складених з кількох слів або мають змінну довжину. Крім того, за допомогою операції додавання по модулю ω або операції "додавання по модулю 2", ми можемо комбінувати слова в один ключ. Одним з алгоритмів, які працюють за таким принципом є хеш-функція Пірсона.

Хешування Пірсона - це алгоритм, який був запропонований Пітером Пірсоном для процесорів з 8-бітними регістрами. Основним завданням алгоритму є швидке обчислення хеш-коду для рядка будь-якої довжини. Як вхідні дані цей алгоритм отримує слово W, яке складається з n символів, кожен розміром 1 байт. Результатом роботи алгоритму є значення хеш-коду, яке знаходиться в діапазоні від 0 до 255. Це значення хеш-коду залежить від кожного символу вхідного слова.

Алгоритм варто описати таким псевдокодом, який використовує

таблицю перестановок T та отримує на вхід рядок W.

```
function pearsonHash(W):
    T = initializePermutationTable()  // Ініціалізуємо таблицю перестановок

    hashValue = 0
    for i = 0 to length(W) - 1:
        character = W[i]
        index = character  // Отримуємо числове представлення символу

        // Застосовуємо перестановку з таблиці T
        hashValue = T[(hashValue + index) mod 256]

    return hashValue
```

Рисунок 1.3 – Псевдокод для хешування рядків змінної довжини

Цей псевдокод використовує цикл для обходу кожного символу в рядку W. Кожен символ перетворюється на числове представлення (наприклад, ASCII-код символу), яке використовується як індекс для отримання значення з таблиці перестановок T. Значення з таблиці T додається до поточного хеш-значення, і результат обмежується в межах 0-255 за допомогою операції модулю. Кінцеве хеш-значення повертається з функції.

Функцію **initializePermutationTable()** слід імплементувати таким чином, щоб вона повертала таблицю перестановок T, яка може бути використана алгоритмом хешування Пірсона.

Перевагою алгоритму слід вважати:

1) простоту обчислення;

2) той факт, що не має таких вхідних даних, для яких імовірність колізії найбільша;

3) можливість модифікації в ідеальну хеш-функцію.

Запропонуємо обчислення як альтернативний спосіб хешування K ключів, котрі складаються з l символів:

$$h(K) = (\boldsymbol{h_1}(\boldsymbol{x_1}) + \boldsymbol{h_2}(\boldsymbol{x_2}) + \cdots + \boldsymbol{h_1}(\boldsymbol{x_1}))mod\ M,$$

де $K = \boldsymbol{x_1 x_2} \ldots \boldsymbol{x_1}$.

1.4 Криптографічні хеш-функції та їх застосування

Серед великої кількості хеш-функцій прийнято виділяти криптографічно стійкі, що використовуються в криптографії, оскільки на них накладаються додаткові вимоги. Для того, щоб хеш-функція вважалася криптографічно стійкою, вона має відповідати трьом основним вимогам, на яких засновано більшість застосувань хеш-функцій в криптографії:

1) бути незворотньою: для заданого значення хеш-функції m має бути обчислювально неможливо знайти блок даних *X*, для якого *H(X) = m*;

2) бути стійкою до колізій першого роду: для заданого повідомлення *M* має бути обчислювально неможливим підібрати інше повідомлення *N*, для якого *H(N) = H(M)*;

3) мати стійкість до колізій другого роду: має бути обчислювально неможливим підібрати пару повідомлень (M, M'), що мають однаковий хеш.

Викладені вимоги залежні один від одного:

1) оборотна функція нестійка до колізій першого та другого роду;

2) функція, нестійка до колізій першого роду, як і нестійка до колізій другого роду; тоді зворотне невірно.

Важливо зазначити, що існує недоведена можливість створення незворотних хеш-функцій, які теоретично унеможливлюють обчислення

будь-якого початкового значення з відомого хеш-коду. Однак, знаходження зворотного значення застосовуваної хеш-функції є високозатратним обчислювальним завданням.

Атака "днів народження" дозволяє знаходити колізії в хеш-функціях, які приймають значення заданої довжини n бітів, в середньому з витратою приблизно $2^{n/2}$ обчислень хеш-функції. Якщо обчислювальна складність знаходження колізій для такої хеш-функції наближається до *2n/2*, то n-бітну хеш-функцію можна вважати криптостійкою.

Криптографічна хеш-функція працює за принципом: Повідомлення M має бути представлене у двійковій формі і розбите на окремі блоки M_i довжиною n біт кожний. Більшість хеш-функцій мають вигляд:

$h_i = H(M_i, h_{i-1})$, де

M_i – черговий блок повідомлення M;

h_{i-1} – хеш-значення усіх попередніх блоків M (має довжину також n біт)

При обчисленні хеш-значення для першого блоку M_1 використовується деяке початкове хеш-значення h_0 , яке можна вибрати випадковим IV або фіксованим. Хеш-значення, обчислене при використанні останнього блоку повідомлення, вважається хеш-значенням усього повідомлення M.

Основні властивості криптографічної хеш-функції:

1) Детермінованість – для однакових повідомлень M функція має повертати однакові хеш-значення h;

2) Односторонність – за значенням h неможливо відновити M;

3) Наявність лавинного ефекту – будь-які, навіть незначні, зміни у повідомленні M призводять до значних змін у хеш-значенні h;

4) Відсутність колізій (унікальність хеша) – ймовірність співпадіння хеш-значень двох різних повідомлень повинна бути надзвичайно малою;

5) Висока швидкість роботи.

Хеш-функція	Рік	Розробники	Довжина блоку	Довжина дайджесту	Кількість раундів
MD5	1992	Ronald Rivest	512	128	64
RIPEMD	1992	The RIPE Consortium	512	128	48
SHA-1	1995	NSA	512	160	80
SHA-256	2002	NSA	512	256	64
SHA-512	2002	NSA	1024	512	80
Whirlpool	2004	Vincent Rijmen, Paulo Barreto	512	512	10
BLAKE-256	2008	Jean-Philippe Aumasson, Luca Henzen, Willi Meier, Raphael C.-W. Phan	512	256	14
Купина	2014	ПАТ «Інститут інформаційних технологій»	512	від 8 до 512 біт	10 або 14

Таблиця 1.3 - Порівняння деяких хеш-функції

Важливою властивістю криптографічних хеш-функцій є їхня здатність до лавинного ефекту. Це означає, що навіть незначна зміна вхідного значення суттєво змінює вихідний хеш. Крім того, хеш-функція повинна бути розроблена таким чином, щоб навіть зі значення хешу не було можливості витягти інформацію про окремі біти вхідного значення. Ця вимога гарантує криптографічну стійкість хеш-алгоритмів, особливо коли використовуються для хешування паролів користувачів з метою отримання ключа.

1.4 SHA256

Хеш-функцію SHA256 це одностороння функція, яка призначена для створення фіксованої довжини цифрових відбитків. Вона зводить вхідні дані різних розмірів (до 2,31 ексабайт або 2^64 біт) до цифрового відбитку фіксованої довжини 256 біт (32 байти). SHA-256 є окремим представником криптографічного сімейства алгоритмів SHA-2 (Secure Hash Algorithm Version 2), які були розроблені та опубліковані Національним бюро з питань стандартів (NIST) у США у 2002 році.

Хеш-функції з сімейства SHA-2 базуються на структурі Меркле-Дамгарда. Після доповнення, вхідне повідомлення розділяється на блоки, кожен з яких складається з 16 слів. Алгоритм обробляє кожен блок повідомлення через цикл з 64 ітераціями. На кожній ітерації два слова піддаються перетворенню, використовуючи інші слова для задання функції перетворення. Результати обробки кожного блоку складаються, утворюючи значення хеш-функції. Так як блоки не можуть бути оброблені паралельно, ініціалізація внутрішнього стану використовує результат обробки попереднього блоку.

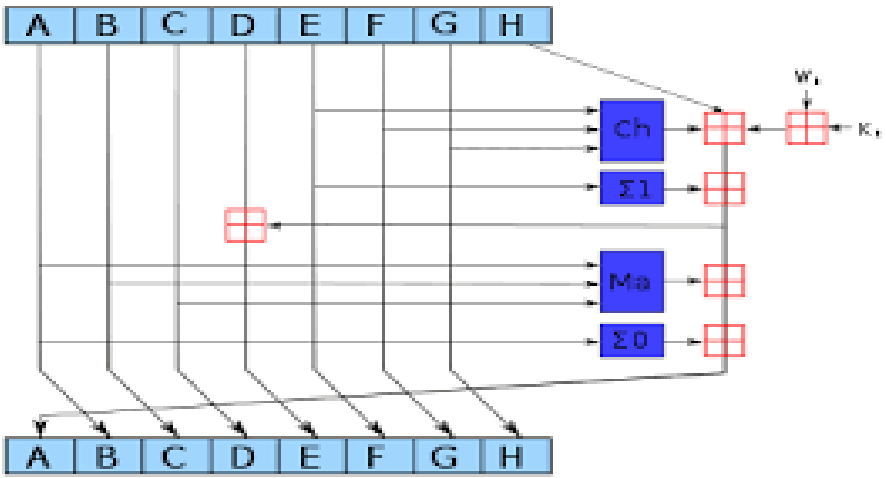

Рисунок 3.4.1 – Одна ітерація обробки блоку даних

Алгоритм використовує такі бітові операції:

- ‖ — Конкатенація,
- + — Додавання,
- And — Побітове «І»,
- Or — Побітове «АБО»,
- Xor — Виключне «АБО»,
- Shr (Shift Right) — Логічний зсув вправо,
- Rotr (Rotate Right) — Циклічний зсув вправо.

Характеристики SHA-256:

- довжина дайджесту повідомлення є 256 біт;
- довжина внутрішнього стану є 256 біт;
- довжина блоку становить 512 біт;
- максимальна довжина повідомлення складає 2 64 - 1 біт;
- довжина слова є 32 біт; - кількість ітерацій в циклі 64 біт.

Наразі відомі методи для побудови колізій до 31 ітерації. Рішення про те, що SHA-3 буде базуватися на зовсім іншому алгоритмі було прийнято опираючись на огляд на алгоритмічної схожості SHA-2 з SHA-1 і наявності в останньої потенційних вразливостей.

1.5 Атаки на хеш-функції та заходи щодо їх захисту

Хеш-функції є важливими криптографічними інструментами, які використовуються для забезпечення цілісності даних та безпеки інформації. Однак, хеш-функції також можуть стати об'єктом атак, де зловмисники намагаються зламати їх або знайти вразливості. В цьому розділі ми розглянемо деякі типи атак на хеш-функції та заходи, які можна прийняти для їх захисту.

Атаки на хеш-функції - це спроби зламати хеш-функцію або знайти

вразливості в її роботі з метою отримання небажаних результатів. В залежності від конкретної атаки, вона може бути спрямована на розкриття вихідного повідомлення, знаходження колізій (різних вхідних повідомлень, що дають однаковий хеш), знайомство зі структурою хеш-функції тощо.

Основні види атак на хеш-функції включають:

1) Переборний пошук (Brute-force attack): атака, при якій зловмисник перебирає всі можливі вхідні значення для хеш-функції з метою знайти відповідні колізії або вихідні значення. Переборний пошук може призвести до витрати великої кількості обчислювальних ресурсів та часу для зламування хеш-функції. Може бути використаний для зламування хеш-функцій, які використовуються для збереження паролів.

2) Колізійна атака (Collision attack): полягає в спробі знайти два різних вхідних повідомлення, які мають однаковий хеш. Зловмисник шукає колізію для того, щоб підробити повідомлення, підмінити дані або порушити цілісність системи. Наслідком того, що зловмиснику вдасться знайти колізію може бути порушення безпеки 26 системи, підробки даних або виконання шкідливого коду. Наприклад, колізійна атака може використовуватися для підробки цифрових підписів або зламування хеш-функцій, використовуваних для збереження паролів. У 2005 році була знайдена колізія в хеш-функції MD5, що дозволило зловмисникам підробити цифрові підписи і зламати систему SSL/TLS. У 2017 році була представлена атака на хеш-функцію SHA-1, яка дозволила знайти колізію і підробити цифровий підпис.

3) Проміжна колізія (Second preimage attack): виникає, коли зловмисник намагається знайти друге повідомлення, яке має той самий хеш, що і вхідне повідомлення. Це відрізняється від колізійної атаки, де зловмисник шукає будь-яке різне повідомлення з однаковим хешем. Проміжна колізія може використовуватися для підробки повідомлень або

підробки даних, оскільки зловмисник може замінити вхідне повідомлення на друге повідомлення з таким самим хешем.

4) Обернена атака (Preimage attack): також відома як пряма атака, полягає в спробі знайти вихідне повідомлення, використовуючи відомий хеш. Зловмисник намагається обернути процес хешування шляхом перебору можливих вхідних повідомлень. Якщо зловмиснику вдасться знайти вихідне повідомлення, використовуючи хеш, це може дозволити йому розкрити конфіденційну інформацію, порушити цілісність даних або зламати систему, що ґрунтується на хеш-функції. Може бути використана для зламування хеш-функцій, використованих для збереження паролів. Зловмисник, маючи хеш пароля, може використати обернену атаку, щоб відновити сам пароль. Ця атака також може бути використана для підробки цифрових підписів. Зловмисник може шукати вхідне повідомлення, яке має той самий хеш, що і підписане повідомлення, з метою створення підробленого підпису.

Забезпечення безпеки хеш-функцій є критичним аспектом криптографії і захисту даних. Оскільки атаки на хеш-функції можуть мати серйозні наслідки для цілісності та безпеки системи, необхідно приймати ефективні заходи для їх запобігання. Ось деякі важливі заходи щодо захисту від атак на хеш-функції:

1) Використання сучасних та стійких хеш-функцій: Важливо вибирати хеш-функції, які вважаються криптографічно стійкими і мають відповідні сертифікати безпеки. Наприклад, сімейства хеш-функцій SHA-2 (SHA-256, SHA-512) та SHA-3 є рекомендованими для застосування в сучасних системах.

2) Додавання солей (salting): Для збереження паролів або генерації цифрових підписів, варто використовувати солі. Сіль - це випадкова додаткова інформація, яка додається до вхідного повідомлення перед хешуванням. Вона підвищує стійкість проти атак перебором та зламу

паролів.

3) Використання ключових похідних функцій: Ключові похідні функції, такі як PBKDF2, bcrypt або scrypt, використовуються для генерації хешів паролів з великою кількістю ітерацій та додатковими параметрами. Це ускладнює атаки перебором та зламу паролів, оскільки вимагає багато обчислювальних ресурсів.

4) Використання ключових хеш-функцій: Для додаткового рівня безпеки можна використовувати ключові хеш-функції, які вимагають наявності секретного ключа для обчислення хеша. Ключові хеш-функції дозволяють захистити дані, забезпечуючи конфіденційність ключа. Це дозволяє збільшити складність атак, оскільки для отримання правильного хешу потрібно знати відповідний ключ.

5) Використання різних хеш-функцій для різних цілей: Розумне використання різних хеш-функцій для різних цілей може знизити ризик уразливості. Наприклад, для зберігання паролів можна використовувати одну хеш-функцію, а для перевірки цілісності даних - іншу. Це ускладнює роботу зловмисників, оскільки вони повинні адаптувати свої атаки до кожного типу хеш-функції.

6) Регулярне оновлення хеш-функцій: З урахуванням появи нових вразливостей і атак, важливо вносити оновлення у використовувані хеш-функції. Слід слідкувати за рекомендаціями від організацій криптографічної безпеки і вчасно оновлювати застосовані хеш-функції на більш стійкі і безпечні варіанти.

7) Аудит безпеки хеш-функцій: Регулярний аудит безпеки хеш-функцій дозволяє виявляти потенційні вразливості і проблеми з їх використанням. Варто проводити перевірки безпеки, оцінювати ризики і вживати відповідних заходів для захисту від нових видів атак.

Загальним підходом до захисту від атак на хеш-функції є поєднання розумного вибору стійких хеш-функцій, використання додаткових заходів

без пеки, таких як солі та ключові похідні функції, регулярного оновлення та аудиту безпеки. Перевірка і вдосконалення безпеки хеш-функцій є постійним процесом, оскільки криптографічні атаки розвиваються і появляються нові методи зламу.

Крім цього, важливо приділяти увагу правильному використанню хеш-функцій. Наприклад, уникати використання власних алгоритмів хешування, оскільки вони можуть мати невідомі вразливості. Також слід уникати залежності безпеки від секретності алгоритму, а замість цього ставити на відкритий алгоритм і забезпечення безпеки через секретні ключі та параметри.

Важливо також звернути увагу на правильне зберігання хешів та захисту від несанкціонованого доступу до них. Застосування механізмів контролю доступу, шифрування та фізичного захисту даних можуть бути важливими елементами захисту хеш-функцій.

Крім перерахованих заходів, слід враховувати також нові розробки та рекомендації в галузі криптографії. Вчасне оновлення і використання нових стандартів та протоколів може забезпечити вищий рівень безпеки для захисту від атак на хеш-функції.

Узагалі, ефективний захист від атак на хеш-функції вимагає комплексного підходу, який включає вибір стійких хеш-функцій, використання додаткових заходів безпеки, регулярне оновлення, аудит безпеки та використання сучасних рекомендацій в галузі криптографії. Тільки таким чином можна забезпечити наді йний рівень захисту від атак на хеш-функції. Застосування сучасних, стійких хеш-функцій у поєднанні з додатковими заходами безпеки, такими як солі, ключові похідні функції та ключові хеш-функції, допомагає підвищити відпорність до атак.

Важливо також враховувати зміни в криптографічному середовищі

та приймати відповідні заходи з оновлення хеш-функцій, якщо поточні алгоритми стають застарілими або виявляються вразливими. Регулярний аудит безпеки хеш-функцій дозволяє виявляти потенційні вразливості та негайно реагувати на них. Необхідно також враховувати контекст використання хеш-функцій і приділяти увагу не тільки алгоритмам, але й застосуванню правильних протоколів і методологій. Збереження хешів у безпечному середовищі, контроль доступу до них та застосування фізичного захисту даних є важливими елементами безпеки.

2. ЗАСТОСУВАННЯ ХЕШ-ФУНКЦІЙ

2.1 Аутентифікація повідомлень

Розглянемо вимоги, яким повинна відповідати хеш-функція для того, щоб вона могла використовуватися в якості аутентифікатора повідомлення. Розглянемо дуже простий приклад хеш-функції . Потім проаналізуємо кілька підходів до побудови хеш-функції . Хеш-функція H, яка використовується для аутентифікації повідомлень, повинна мати наступні властивості:

1. Хеш-функція *H* повинна застосовуватися до блоку даних будь-якої довжини.

2. Хеш-функція *H* створює вихід фіксованої довжини.

3. *H (M)* відносно легко (за поліноміальний час) обчислюється для будь-якого значення *M* .

4. Для будь-якого даного значення хеш-коду *h* обчислювально неможливо знайти *M* таке, що *H (M) = h* .

5. Для будь-якого даного *a* обчислювально неможливо знайти , що *H (b) = H (a)* .

6. Обчислювально неможливо знайти довільну пару (*a, b*) таку, що *H (b) = H (a)* .

Перші три властивості вимагають, щоб хеш-функція створювала хеш-код для будь-якого повідомлення.

Четверте властивість визначає вимога однобічності хеш-функції : легко створити хеш-код по цим повідомленням, але неможливо відновити повідомлення з даного хеш-коду . Це властивість важливо, якщо аутентифікація з використанням хеш-функції включає секретне значення. Саме секретне значення може не надсилатися, проте, якщо хеш-функція не є

односторонньою, противник може легко розкрити секретне значення наступним чином. При перехопленні передачі атакуючий отримує повідомлення *M* і хеш-код $C = H(S_{AB} || M)$. Якщо атакуючий може інвертувати хеш-функцію , то, отже, він може отримати $S_{AB} || M = H^{-1}(C)$. Так як атакуючий тепер знає і M і $S_{AB} || M$, отримати S_{AB} зовсім просто.

П'яте властивість гарантує, що неможливо знайти інше повідомлення, чиє значення хеш-функції збігалося б із значенням хеш-функції даного повідомлення. Це запобігає підробку аутентифікатора при використанні зашифрованого хеш-коду . В даному випадку противник може читати повідомлення і, отже, створити його хеш-код . Але так як противник не володіє секретним ключем , він не має можливості змінити повідомлення так, щоб одержувач цього не виявив. Якщо ця властивість не виконується, атакуючий має можливість виконати наступну послідовність дій: перехопити повідомлення і його зашифрований хеш-код , обчислити хеш-код повідомлення, створити альтернативне повідомлення з тим же самим хеш-кодом , замінити вихідне повідомлення на підроблене. Оскільки хеш-коди цих повідомлень збігаються, одержувач не виявить підміни.

Шосте властивість захищає проти класу атак, відомих як атака " день народження ".

Для аутентифікація повідомлень використовують наступні алгоритми:

- MD5 – алгоритм отримує на вході повідомлення довільної довжини і створює в якості виходу дайджест повідомлення довжиною 128 біт
- SHA-1 - алгоритм отримує на вході повідомлення максимальної довжини 2 64 біт і створює в якості виходу дайджест повідомлення довжиною 160 біт.
- ГОСТ 34.11 - довжина хеш-коду дорівнює 256 бітам. Алгоритм розбиває повідомлення на блоки, довжина яких також дорівнює 256 бітам.

Крім того, параметром алгоритму є стартовий вектор хешування H - довільне фіксоване значення довжиною також 256 біт.

2.2 Цифровий підпис

Аутентифікація має на меті захистити комунікацію між двома учасниками від впливу зовнішньої третьої сторони. Проте, проста аутентифікація не забезпечує захист між самими учасниками, і між ними можуть виникати різні форми конфліктів.

У випадку, коли взаємодіючі сторони не довіряють одна одній, потрібні більш надійні методи, ніж проста аутентифікація на основі загального секрету. Одним з можливих рішень для такої ситуації є використання цифрового підпису. Цифровий підпис повиннен мати наступні властивості:

1. Забезпечувати можливість перевірити автора повідомлення, дату та час створення підпису.

2. Гарантувати, що вміст повідомлення не змінювався під час створення підпису.

3. Підпис має бути перевіреним третьою стороною для вирішення спорних ситуацій.

Таким чином, функція цифрового підпису включає функцію аутентифікації. На підставі цих властивостей можна сформулювати наступні вимоги до цифрового підпису :

1. Підпис має бути двійкового зразком, який залежить від підписується повідомлення.

2. Підпис має використовувати деяку унікальну інформацію

відправника для запобігання підробки або відмови.

3. Створювати цифровий підпис має бути відносно легко.

4. Повинно бути обчислювально неможливо підробити цифровий підпис як створенням нового повідомлення для існуючої цифрового підпису , так і створенням помилкової цифрового підпису для деякого повідомлення.

5. Цифровий підпис повинна бути досить компактними і не займати багато пам'яті.

Сильна хеш-функція , зашифрована закритим ключем відправника, задовольняє перерахованим вимогам.

Існує кілька підходів до використання функції цифрового підпису . Всі вони можуть бути розділені на дві категорії: прямі і арбітражні .

Види шифрувань

- Симетричне шифрування, арбітр бачить повідомлення
- Симетричне шифрування, арбітр не бачить повідомлення
- Шифрування відкритим ключем, арбітр не бачить повідомлення

2.3 Захист паролів

Захист паролів є важливим аспектом безпеки в сучасних інформаційних системах. Використання хеш-функцій є одним з найпоширеніших методів для збереження паролів у безпечному вигляді. Хеш-функції перетворюють паролі на унікальні хеш-значення, які зберігаються у базі даних замість самого пароля. Це забезпечує кілька переваг у контексті захисту паролів:

1) Хешування паролів: Під час реєстрації або зміни паролів

користувачів, їхні паролі не зберігаються в чистому вигляді. Замість цього, використовуються хеш-функції, що перетворюють паролі на фіксований розмір хеш-значень. Це унеможливлює пряме відновлення початкового пароля з хеш-значення.

2) Захист від розкриття паролів: Якщо база даних з хешами паролів стає доступною зловмисникам, хеш-значення самі по собі недостатньо для відновлення паролів. Розкриття паролів потребує великих обчислювальних зусиль та витрат часу на перебір можливих комбінацій.

3) Сіль (Salt): Додавання солі до паролів перед хешуванням забезпечує ще більший рівень безпеки. Сіль - це додаткова випадкова послідовність, яка додається до паролів перед їх хешуванням. Кожен користувач може мати унікальну сіль. Використання солі перешкоджає використанню заздалегідь підготовлених таблиць радужних хешів (rainbow tables) для швидкого відновлення паролів.

4) Криптографічно стійкі хеш-функції: Важливо використовувати криптографічно стійкі хеш-функції, такі як SHA-2, SHA-3 або bcrypt. Ці алгоритми володіють властивостями, що ускладнюють злам паролів, включаючи опікання, велику випадковість хеш-значень та можливість обчислення хешу великих обсягів даних.

Застосування хеш-функцій для захисту паролів є важливим етапом у забезпеченні безпеки інформаційних систем. Комбінація хешування, використання солі та використання сучасних криптографічно стійких алгоритмів допомагає знизити ризик розкриття паролів та забезпечити високий рівень безпеки парольних систем.

2.4 Блокчейн технології

Блокчейн є книгою децентралізованих даних, обмін якими виконується безпечними каналами. Технологія блокчейн дозволяє групі обраних учасників обмінюватись даними. Хмарні сервіси блокчейну дають

можливість легко збирати та інтегрувати дані транзакцій із кількох джерел, а також обмінюватися ними. Дані розбиваються на загальні блоки, які зчеплюються один з одним за допомогою унікальних ідентифікаторів, що мають форму криптографічних хеш-функцій.

Блокчейн забезпечує цілісність даних з єдиним джерелом достовірної інформації, усуваючи дублювання даних та підвищуючи їхню безпеку.

У системі блокчейна неможливі шахрайство та маніпуляція даними, оскільки змінити їх можна лише з дозволу кворуму сторін. Книжкою блокчейна можна обмінюватися, але її не можна змінити. Якщо хтось спробує змінити дані, всі учасники отримають повідомлення про це і знатимуть, хто це був.

Блокчейн можна як запис історії транзакцій. Кожен блок послідовно зчепляється з попереднім блоком і записується в незмінній формі в мережі пірінгу. Криптографічний траст та технологія гарантування застосовують унікальний ідентифікатор (або цифровий відбиток пальця) до кожної транзакції.

Такий ланцюг блоків забезпечує довіру, підзвітність, прозорість та безпеку. Все це дозволяє компаніям різних видів та торговим партнерам обмінюватися даними та отримувати доступ до них. Такий феномен називається довірою третіх сторін з урахуванням консенсусу.

Усі учасники ведуть закодований запис кожної транзакції в децентралізованому, високомасштабованому та гнучкому механізмі, який неможливо анулювати. Для блокчейна не потрібні додаткові витрати чи посередники. Наявність децентралізованого, єдиного джерела достовірної інформації забезпечує зниження витрат за виконання довірених комерційних операцій між сторонами, які можуть повною мірою довіряти одне одному. В ексклюзивному блокчейні, що використовується більшістю підприємств,

учасникам дозволено входити до складу мережі, при цьому кожен із них веде закодований запис кожної транзакції.

Ця унікальна технологія дає багато переваг будь-якій компанії або групі компаній, які потребують безпечного запису транзакцій у реальному часі, і якій можна поділитися. Немає одного місця, де зберігається вся інформація, що гарантує більш високу безпеку та доступність, оскільки немає ніякої центральної точки вразливості.

Ось кілька важливих визначень, які допоможуть краще зрозуміти блокчейн, його базову технологію та приклади використання:

1) Децентралізований траст: Основною причиною, через яку компанії віддають перевагу технології блокчейн перед іншими варіантами зберігання інформації, є забезпечення цілісності даних, незалежно від централізованого органу. Це називається децентралізованим трастом, що реалізується за рахунок надійних даних.

2) Блоки блокчейну: Назва «блокчейн» походить від того, що дані зберігаються в блоках, кожен з яких пов'язаний з попереднім блоком, утворюючи схожу на ланцюг структуру (chain англійською означає ланцюг). При роботі з технологією блокчейн можна тільки додавати (прикріплювати) нові блоки до вже наявного ланцюжка блоків. Після додавання блоку до блокчейна його не можна видалити або змінити.

3) Алгоритми консенсусу: Алгоритми, які визначають правила, що діють у системі блокчейну. Після того як сторони, що беруть участь в системі, визначать правила блокчейну, за їх дотриманням стежитимуть алгоритми консенсусу.

4) Вузли блокчейну: Блоки даних системи блокчейна зберігаються на вузлах - сховищах, що підтримують синхронізацію даних та їхній актуальний стан. Вузол може швидко визначити, чи змінився той чи інший

блок після додавання. Коли новий, повноцінний вузол приєднується до мережі блокчейна, він завантажує копію всіх блоків, що у мережі на даний момент. Після того, як новий вузол синхронізується з іншими вузлами та отримає актуальну версію блокчейна, він може отримувати будь-які нові блоки так само, як інші вузли мережі.

Є два основних типи вузлів блокчейну:

1) Повноцінні вузли зберігають повну копію блокчейну.

2) Полегшені вузли зберігають лише останні блоки і можуть вимагати більш старі блоки в міру необхідності.

3. ПРАКТИЧНА ЧАСТИНА

3.1 Загальна ідея

Загальна ідея програми полягає у створенні простої реалізації блокчейну мовою Golang, щоб показати основні концепції та компоненти, що лежать в основі цієї технології. Блокчейн як технологія має величезний потенціал для революції в різних сферах, починаючи від фінансів і ланцюгів поставок до голосування та управління правами доступу. Він пропонує децентралізовану, надійну та прозору систему, яка може усунути проблеми централізованих структур, таких як відсутність довіри, єдиної точки відмови та маніпуляції даними.

Мета цієї програми - познайомити розробників з основами блокчейну та дати їм можливість вивчити та експериментувати з його різними компонентами. Проект пропує реалізацію, яка може бути відправною точкою для подальшого дослідження та розробки складніших блокчейн-додатків.

Програма включає реалізацію блоків, ланцюжка блоків, алгоритму Proof-of-Work, транзакцій і гаманців. Блоки є основними будівельними блоками блокчейна та містять дані, такі як транзакції та хеш попереднього блоку. Ланцюжок блоків забезпечує зв'язок між блоками та підтримує безперервність ланцюжка. Алгоритм Proof-of-Work використовується для створення нових блоків шляхом вирішення обчислювально-складного завдання. Транзакції є передачу даних або цифрових активів між учасниками блокчейна, а гаманці служать для генерації та управління ключовими парами для підпису та перевірки транзакцій.

Ця програма також надає можливість взаємодії з блокчейном через HTTP-запити, що дозволяє розробникам створювати та перевіряти транзакції, додавати нові блоки в ланцюжок, а також отримувати інформацію про стан блокчейну.

Загальна ідея цієї програми полягає в тому, щоб зробити блокчейн доступним і зрозумілим для всіх, хто цікавиться цією захоплюючою технологією, і надихнути розробників на створення нових та інноваційних програм на основі блокчейну.

3.2 Опис роботи програми

При запуску програми створюється новий екземпляр блокчейна. Користувачеві надається командний рядок, де він може вводити різні команди для взаємодії з блокчейном. Доступні команди:

- addblock: Створює новий блок у блокчейні. Користувачеві пропонується ввести дані для нового блоку.
- printchain: Виводить інформацію про всі блоки в ланцюжку блоків.
- getbalance: Виводить баланс гаманця за вказаною адресою. Користувач вводить адресу гаманця для перевірки балансу.
- createwallet: Створює новий гаманець (пару публічного та приватного ключів) та виводить публічну адресу гаманця.
- listddresses: Виводить список усіх доступних адрес гаманців.

Після введення команди програма обробляє запит користувача і виконує відповідні операції:

- При використанні команди addblock створюється новий блок із введеними даними і додається в ланцюжок блоків.
- При використанні команди printchain, програма виводить інформацію про кожен блок у ланцюжку блоків, включаючи хеш, дані та хеш попереднього блоку.
- При використанні команди getbalance програма обходить всі блоки в ланцюжку і перевіряє кожну транзакцію на наявність заданої адреси входу або виходу, підраховуючи відповідні суми.
- При використанні команди createwallet, генерується новий набір публічного та приватного ключів для гаманця, який зберігається у файлі.

- При використанні команди Listeddresses програма виводить список всіх доступних адрес гаманців, отриманих зі збережених ключів.

Користувач може послідовно вводити команди та взаємодіяти з блокчейном, виконуючи різні операції, такі як додавання блоків, перевірка балансу та створення нових гаманців. Всі зміни, включаючи додавання нових блоків та створення нових гаманців, зберігаються у файлі, щоб при наступному запуску програми можна було відновити попередній стан блокчейна. Програма використовує хеш-функцію SHA256.

3.3 Деталі програмної реалізації

Программа розпочинаю роботу з файлу main.go(див. Додаток А). Він запускає командний рядок інтерфейсу (CLI) взаємодії з блокчейном. Давайте розглянемо докладніше деталі роботи коду:

1) Функція main() є точкою входу до програми. Вона викликається під час запуску програми.
2) Всередині функції main(), є відкладена інструкція defer os.Exit(0). Це гарантує, що програма завершиться з кодом 0 після виконання всіх операцій.
3) Створюється екземпляр структури cli.CommandLine {}. Структура CommandLine визначена в файлу cli і надає методи обробки команд у командному рядку.
4) Викликається метод Run() на створеному екземплярі структури CommandLine. Метод Run() запускає командний рядок інтерфейсу (CLI) та починає прослуховувати команди користувача.

Командний рядок інтерфейсу дозволяє користувачеві взаємодіяти з блокчейном, вводячи різні команди. Команди можуть включати додавання нових блоків, виведення інформації про блоки, перевірку балансу гаманця та створення нових гаманців.

Команди, введені користувачем, обробляються відповідними функціями всередині файлу cli(див. Додаток Б). Командний рядок інтерфейсу продовжує прослуховувати команди користувача доти, доки програма не буде завершена або команда не призведе до завершення програми. У цьому файлі реалізовані наступні функції:

1) printUsage(): Функція виводить інформацію про доступні команди та їх використання.

2) StartNode(nodeID, minerAddress string): Функція запускає вузол блокчейна. Вона приймає ідентифікатор вузла (nodeID) та адресу для майнера (minerAddress) як аргументи. Якщо вказана адреса майнера, то функція перевіряє його валідність та виводить повідомлення про включену генерацію нових блоків для зазначеної адреси. Потім викликається функція network.StartServer(), яка ініціює запуск сервера для блокчейна вузла.

3) listAddresses(nodeID string): Функція відображає список гаманців, доступних у файлі гаманців. Вона приймає ідентифікатор вузла (nodeID) як аргумент. Спочатку функція створює новий екземпляр Wallets та завантажує гаманці з файлу за допомогою функції CreateWallets(). Потім вона отримує список всіх адрес гаманців за допомогою функції GetAllAddresses() і виводить кожну адресу на екран.

4) createWallet(nodeID string): Функція створює новий гаманець та зберігає його у файл. Вона приймає ідентифікатор вузла (nodeID) як аргумент. Спочатку функція створює новий екземпляр Wallets за допомогою функції CreateWallets(). Потім вона додає новий гаманець за допомогою функції AddWallet(), зберігає файл гаманців за допомогою функції SaveFile() і виводить створену адресу на екран.

5) printChain(nodeID string): Функція виводить інформацію про блоки в ланцюжку блоків. Вона приймає ідентифікатор вузла (nodeID) як аргумент. Спочатку функція продовжує блокчейн із заданим ідентифікатором, отримує ітератор для перебору блоків за допомогою

функції Iterator(). Потім вона послідовно виводить інформацію про кожен блок, включаючи хеш блоку, попередній хеш, результати роботи алгоритму Proof-of-Work (PoW) та інформацію про кожну транзакцію в блоці.

6) createBlockChain(address, nodeID string): Функція створює новий ланцюжок блоків і відправляє нагороду за генезис-блок на вказану адресу. Вона приймає адресу одержувача (address) та ідентифікатор вузла (nodeID) як аргументи. Якщо адреса невалідна, функція викликає помилку. Потім вона ініціює блокчейн із зазначеною адресою одержувача та ідентифікатором вузла за допомогою функції InitBlockChain(). Після створення блокчейна, функція виводить повідомлення про успішне завершення.

7) getBalance(address, nodeID string): Функція виводить баланс (кількість монет) для зазначеної адреси. Вона приймає адресу (address) та ідентифікатор вузла (nodeID) як аргументи. Якщо адреса невалідна, функція викликає помилку. Спочатку функція продовжує блокчейн із заданим ідентифікатором, створює екземпляр UTXOSet за допомогою блокчейна та отримує непотрачені виходи транзакцій (UTXO) для зазначеної адреси за допомогою функції FindUnspentTransactions(). Потім вона підсумовує значення всіх UTXO та виводить баланс на екран.

8) send(from, to string, amount int, nodeID string, mineNow bool): Функція надсилає вказану кількість монет з однієї адреси на іншу адресу. Вона приймає адресу відправника (from), адресу одержувача (to), кількість монет (amount), ідентифікатор вузла (nodeID) та прапор (mineNow), що вказує на майнінг нового блоку. Якщо адреса відправника або одержувача є невалідною, функція викликає помилку. Спочатку функція продовжує блокчейн із заданим ідентифікатором, створює екземпляр UTXOSet за допомогою блокчейна та отримує гаманець відправника із файлу гаманців за допомогою функції GetWallet(). Потім вона створює нову транзакцію за допомогою функції NewTransaction() та відправляє її на мережу за допомогою функції SendTx(). Якщо прапор mineNow встановлено в true, функція також створює coinbase-транзакцію (нагорода за майнінг) і майне

новий блок. Після завершення надсилання транзакції, функція виводить повідомлення про успішне виконання.

9) Run(): Функція виконує розбір аргументів командного рядка та запускає відповідні команди. Вона викликається для запуску командного рядка.. Залежно від команди викликається відповідна функція. Якщо не було передано жодну команду, виводиться довідка щодо використання командного рядка.

Далі у папці blockchain розписані структури block, blockchain і Transaction та методи пов’язанні з ними(код див. у Додаток Б). Розглянемо основні з них:

- **Block** (структура): Timestamp – тимчасова мітка створення блоку. Transactions – список транзакцій у блоці. PrevHash – хеш попереднього блоку в ланцюжку. Hash – хеш поточного блоку. Nonce – значення nonce, що використовується для доказу роботи (Proof-of-Work).
- **Blockchain** (структура): Blocks – список блоків у ланцюжку блоків. Database – покажчик на базу даних, що використовується для зберігання блоків. Tip - покажчик на останній доданий блок у ланцюжок.
- **ContinueBlockChain**: Приймає ідентифікатор вузла nodeID як аргумент. Створює або завантажує блокчейн із бази даних, використовуючи ідентифікатор вузла. Повертає покажчик на створений чи завантажений блокчейн.
- **InitBlockChain**: Приймає адресу address та ідентифікатор вузла nodeID як аргументи. Створює новий блокчейн із генезис-блоком, пов'язаним із заданою адресою. Повертає покажчик на створений блокчейн.
- **AddBlock**: Приймає список трансакцій транзакцій і ідентифікатор вузла nodeID як аргументи. Створює новий блок, використовуючи попередній хеш та список транзакцій. Додає блок у ланцюжок блоків. Зберігає блокчейн у базі даних.
- **MineBlock**: Приймає список трансакцій транзакцій і ідентифікатор

вузла nodeID як аргументи. Створює новий блок із попереднім хешом та список транзакцій. Виконує пошук доказу роботи (Proof-of-Work), змінюючи значення nonce блоку доти, доки знайдено відповідне значення хеша блоку. Додає блок у ланцюжок блоків. Зберігає блокчейн у базі даних.

- **FindTransaction**: Приймає хеш транзакції ID як аргумент.Шукає транзакцію у блоках ланцюжка блоків. Повертає знайдену транзакцію або nil, якщо транзакцію не знайдено.
- CreateBlock (Створення блоку): Приймає попередній блок, дані транзакцій та іншу необхідну інформацію. Створює екземпляр нового блоку з отриманим хешом та іншою інформацією. Додає транзакції до блоку. Повертає створений блок.
- CreateGenesisBlock (Створення генезис-блоку): Створює перший блок (генезис-блок) у блокчейні.
- InitBlockchain(address string) *Blockchain: Створює новий блокчейн із вказаною адресою одержувача. Це функція ініціалізації блокчейну та створення генезис-блоку.
- NewUTXOTransaction(wallet *Wallet, to string, amount int, UTXOSet *UTXOSet) *Transaction: Створює нову транзакцію. Приймає відправника, одержувача, суму переказу та набір UTXO для перевірки входів та створення виходів.
- CoinbaseTx(to, data string) * Transaction: Створює транзакцію Coinbase, яка є спеціальною транзакцією, що створюється майнером для отримання винагороди. Приймає адресу одержувача та дані транзакції.
- NewTransaction(inputs []TXInput, outputs []TXOutput) *Transaction: Створює спільну транзакцію із зазначеними входами та виходами.

Залишилось лише розглянути функції для створення і управління wallet, які знаходяться у папці wallet

Функції створення гаманця:

- Структура Wallet: Створює структуру гаманця з публічним та приватним ключами.
- GetAddress() []byte: Повертає адресу гаманця у вигляді байтів.

Функції керування набором гаманців:

- CreateWallets() (*Wallets, error): Створює новий набір гаманців. Якщо файл існує, завантажує дані із нього.
- AddWallet() []byte: Додає новий гаманець у набір і повертає його адресу.
- GetAllAddresses() []string: Повертає список усіх гаманців у наборі.
- GetWallet(address string) *Wallet: Повертає гаманець за вказаною адресою.

Функції серіалізації та збереження/завантаження даних:

- SaveFile(nodeID string): Зберігає набір гаманців у файл із вказаним ідентифікатором вузла.
- LoadFile(nodeID string) error: Завантажує дані набору гаманців із файлу із зазначеним ідентифікатором вузла.
-

3.4 Результат роботи програми

Програма запускається через консоль за допомогою команди **go run main.go** (рис. 3.3.1).

```
D:\projects\golang-blockchain>go run main.go
Usage:
 getbalance -address ADDRESS - отримати залишок за адресою
 createblockchain -address ADDRESS - створити блокчейн і надсилає винагороду Genesis на адресу
 printchain - вивести блоки в ланцюжку
 send -from FROM -to TO -amount AMOUNT -mine - Надіслати суму монет. Потім -mine прапор встановлено, майніть з цього вузла
 createwallet - Створити новий гаманець
 listaddresses - Перелік адреси в нашому файлі гаманця
 reindexutxo - Відновлити набір UTXO
 startnode -miner ADDRESS - Запустити вузол із ідентифікатором, указаним у NODE_ID env. var. -miner enables mining
```

Рис. 3.3.1 – запуск програми

Як бачимо програма надає нам список команд. Для прикладу створимо гаманець та додамо туди блокчейн. За допомогою команди: **go run main.go createwallet**. Пілся виконання програми ми отримали гаманець з адресою **1LvHSxLxPCeMw74Da3MHHYmPAiTbUBPV9i**. Далі створюємо блокчейн за допомогою команди: **go run main.go createblockchain -address ADDRESS**, де **ADDRESS** – адреса гаманця який нам потрібен. У даному випадку **1LvHSxLxPCeMw74Da3MHHYmPAiTbUBPV9i**. Для перевірки водимо команду **go run main.go printchain**, яке виводіть усі ланцюги в гаманці. Результат можна подивитися на рис. 3.3.2

```
D:\projects\golang-blockchain>go run main.go createwallet
Нова адреса: 1LvHSxLxPCeMw74Da3MHHYmPAiTbUBPV9i

D:\projects\golang-blockchain>go run main.go createblockchain -address 1LvHSxLxPCeMw74Da3MHHYmPAiTbUBPV9i
2023/05/31 13:36:35 Replaying from value pointer: {Fid:0 Len:0 Offset:0}
2023/05/31 13:36:35 Iterating file id: 0
2023/05/31 13:36:35 Iteration took: 0s
000e435680172e8ad063ffce7c3b75d1e989f53fcadf97d0502778386b387c56
Genesis created
Finished!

D:\projects\golang-blockchain>go run main.go printchain
2023/05/31 13:36:47 Replaying from value pointer: {Fid:0 Len:42 Offset:915}
2023/05/31 13:36:47 Iterating file id: 0
2023/05/31 13:36:47 Iteration took: 944µs
Hash: 000e435680172e8ad063ffce7c3b75d1e989f53fcadf97d0502778386b387c56
Prev. hash:
PoW: true
--- Transaction 8b956d71b2fd536c9e1fbc9c7d6ad11537c15fa30e1412546169e84f7ee8fc5c:
     Input 0:
       TXID:
       Out:       -1
       Signature:
       PubKey:    466972737420547261616e73616374696f6e2066726f6d2047656e65736973
     Output 0:
       Value:  20
       Script: da7e2a609e58029f077e0abf30a00b60370c2003
```

Рис. 3.3.2 – створення гаманця і блокчейну в ньому

ВИСНОВОК

У цій дипломній роботі було проведено дослідження хеш-функції та розробка блокчейн-проекту, використовуючи мову програмування Golang. Робота зосереджувалась на вивченні основних концепцій хеш-функції та блокчейн технології, розгляді важливих принципів та застосуванні хеш-функцій у контексті блокчейну.

У першому розділі на дано теоричні відомості про хеш-функції, методи їх створення, види та атаки на них і методи захисту. У другому розділі можливі використання хеш-функції для цифровіх підписів, захисту паролю, аутентифікацію повідомлень і блокчейн технології.

Під час розробки проекту було використано переваги мови програмування Golang, такі як ефективність, простота використання та широкі можливості для розробки розподілених систем. Завдяки цьому було реалізовано функціонал блокчейн-мережі, включаючи додавання нових блоків до ланцюжка, підтвердження транзакцій та забезпечення цілісності даних за допомогою хеш-функцій.

Результати дослідження та розробки блокчейн-проекту показали його потенціал та можливості у різних сферах застосування. Блокчейн технологія може бути використана для забезпечення безпеки та автентичності електронних документів, захисту паролів, контролю цілісності даних, а також у сферах фінансів, логістики, ліцензування та управління правами, а також в Інтернеті речей та "розумних" контрактах.

Проект блокчейн на Golang виявився ефективним та функціональним, він відповідає поставленим цілям та вимогам. Однак, слід зазначити, що блокчейн технологія також стикається з викликами,

такими як масштабованість, безпека та приватність. Для подальшого розвитку та вдосконалення блокчейн технологій необхідно проводити додаткові дослідження та впроваджувати нові інноваційні рішення.

В цілому, розробка блокчейн-проекту на мові програмування Golang виявилася цікавим та перспективним завданням. Ця технологія має великий потенціал для трансформації різних галузей та вирішення ряду проблем, пов'язаних з цілісністю, безпекою та довірою до даних. Застосування блокчейн технологій може сприяти покращенню ефективності та надійності багатьох процесів і систем у сучасному світі.

Результати дипломного проекту можуть використовуватися для подальшого розвитку та вдосконалення блокчейн технологій, а також для конкретних застосувань у різних галузях. Основні результати дипломного проекту можуть бути використані для наступних цілей:

1) Подальше дослідження: Результати проекту можуть служити основою для подальших наукових досліджень у галузі блокчейн технологій. Це може включати розширення функціональності блокчейн-мережі, розробку нових протоколів консенсусу, вдосконалення алгоритмів безпеки та приватності, а також вивчення впливу блокчейну на суспільство та економіку.
2) Реалізація комерційних проектів: Отримані результати можуть бути використані для розробки комерційних блокчейн-проектів у різних галузях. Наприклад, проект може бути використаний для створення системи управління логістичними процесами, фінансових транзакцій або ліцензування та управління правами.
3) Підтримка академічних курсів: Результати проекту можуть стати основою для створення навчальних курсів та матеріалів з блокчейн технологій. Це може допомогти студентам та дослідникам у вивченні та

розумінні принципів та практик роботи з блокчейнами.

4) Консультаційні послуги: Отримані знання та навички в розробці блокчейн-проектів можуть бути використані для надання консультаційних послуг компаніям та організаціям, які планують впровадити блокчейн технології у своїх процесах.

Загальною метою використання результатів дипломного проекту є сприяння розвитку та популяризації блокчейн технологій, їх застосування в реальних сценаріях та сприяння інноваційному розвитку різних галузей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Черкасов, С. В. Криптографія: теорія та практика: навч. посібник для вузів/С. В. Черкасов, А. А. Ломовський. - Москва: Фізматліт, 2004.
2. Криптографія: підручник[С. І. Мацюк, О. І. Роднєва, Ю. П. Сивухін та ін.] ; за ред. С. І. Мацюка. – Київ: ВПЦ “Київський університет”, 2005.
3. Столар, А. Криптографія: теорія та практика / А. Столар; пров. з англ. Е. Гуніна; за ред. Ю. Шермана. - Москва: Інтернет-Університет Інформаційних Технологій, 2006.
4. Борисенко, І. І. Криптографічні методи захисту інформації/І. І. Борисенко, О. О. Духовний. – Київ: Видавництво “Навчальна книга – Богдан”, 2015. – 280 с.
5. Хоменко, Ю. В. Криптографічні методи захисту інформації: навч. посібник / Ю. В. Хоменко, В. М. Ковалев. – Київ: Національний технічний університет України "КПІ", 2017. – 252 с.
6. Бондаренко, О. В. Хеш-функції та їх застосування в інформаційних системах / О. В. Бондаренко, О. В. Семенов. – Київ: Видавництво “Навчальна книга – Богдан”, 2018. – 208 с.
7. Губарєв, В. В. Криптографія та захист інформації
8. Хеш-функція [Електронний ресурс]: https://en.wikipedia.org/wiki/Hash_function
9. Електронний цифровий підпис [Електронний ресурс]: https://en.wikipedia.org/wiki/Digital_signature
10. Функції хешування. Поняття. Вимоги. Класифікація, властивості і застосування [Електронний ресурс]: https://openarchive.nure.ua/server/api/core/bitstreams/1003a2e8-f582-45bb-a487-a8f17de52f3f/content
11. Криптографічні хеш-функції [Електронний ресурс]: https://learn.ztu.edu.ua/mod/resource/view.php?id=120952
12. Design of Hashing Algorithms [Pieprzyk & Sadeghiyan 1993-11-23]: http://crypto.stanford.edu/~mironov/papers/hash_survey.pdf
13. International Journal of Electrical and Computer Engineering (IJECE) Vol. 9, No. 1, February 2019, pp. 352~358
14. International Journal of Security and Its Applications Vol. 7, No. 3, May, 2013
15. G. Brassard, editor. Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology. Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings, volume 435 of Lecture Notes in Computer Science. Springer, 1990
16. B. Denton, R. Adhami. Modern Hash Function Construction. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.218.1717.
17. A. Rimoldi. An introduction to Hash functions. http://www.science.unitn.it/~sala/BunnyTN/rimoldi.pdf.

18. Информационная технология. Криптографическая защита информации. Функция хеширования. ГОСТ Р 34.11-2012. – Москва, Стандартинформ, 2012.
19. S. Al-Kuwari, J. H. Davenport, R. J. Bradford. Cryptographic Hash Functions: Recent Design Trends and Security Notions. Short Paper Proceedings of 6th China International Conference on Information Security and Cryptology (Inscrypt '10). 2010, Science Press of China, pp. 133-150.
20. J.-S. Coron, Y. Dodis, C. Malinaud, P. Puniya. Merkle-Damgård Revisited: How to Construct a Hash Function. In Advances in Cryptology — CRYPTO 2005, volume 3621 of Lecture Notes in Computer Science, pages 430–448. Springer, 2005.
21. T. Peyrin. State-of-the-art of Hash Functions. http://www1.spms.ntu.edu.sg/~ccrg/WAC2010/slides/session_4/4_3_Peyrin_hash.pdf .
22. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. On the indifferentiability of the sponge construction. In Advances in Cryptology — EUROCRYPT 2008, volume 4965 of Lecture Notes in Computer Science, pages 181–197. Springer, 2008
23. G. Bertoni, J. Daemen, M. Peeters, G. Van Assche. Sponge functions. Ecrypt Hash Workshop, Barcelona, Spain, May 2007. http://sponge.noekeon.org/.
24. И.Д. Горбенко, С.И. Збитнев, А.А. Поляков Криптографические преобразования в группах точек эллиптических кривых методом Полларда // Радиотехника: Всеукр. межвед. науч-тех. сб 2001. Вып. 119. С. 43-50.

ДОДАТОК А

Код у файлі main.go

```
package main

import (
	"os"

	"github.com/tensor-programming/golang-blockchain/cli"
)

func main() {
	defer os.Exit(0)
	cmd := cli.CommandLine{}
	cmd.Run()
}
```

ДОДАТОК Б

Функції в файлі cli.go
Посилання на код: https://github.com/Koshman-Nikita/golang-blockchain/blob/main/cli/cli.go
Функції в файлі blockchain
Посилання на код: https://github.com/Koshman-Nikita/golang-blockchain/tree/main/blockchain

ДОДАТОК В

```
type Wallet struct {
	PrivateKey ecdsa.PrivateKey
	PublicKey  []byte
}

func (w Wallet) Address() []byte {
	pubHash := PublicKeyHash(w.PublicKey)

	versionedHash := append([]byte{version}, pubHash...)
	checksum := Checksum(versionedHash)

	fullHash := append(versionedHash, checksum...)
	address := Base58Encode(fullHash)

	return address
}

func CreateWallets(nodeId string) (*Wallets, error) {
	wallets := Wallets{}
	wallets.Wallets = make(map[string]*Wallet)

	err := wallets.LoadFile(nodeId)

	return &wallets, err
}

func (ws *Wallets) AddWallet() string {
	wallet := MakeWallet()
	address := fmt.Sprintf("%s", wallet.Address())

	ws.Wallets[address] = wallet

	return address
}

func (ws *Wallets) GetAllAddresses() []string {
	var addresses []string

	for address := range ws.Wallets {
		addresses = append(addresses, address)
	}

	return addresses
}

func (ws Wallets) GetWallet(address string) Wallet {
	return *ws.Wallets[address]
}
```

```
func (ws *Wallets) LoadFile(nodeId string) error {
	walletFile := fmt.Sprintf(walletFile, nodeId)
	if _, err := os.Stat(walletFile); os.IsNotExist(err) {
		return err
	}

	var wallets Wallets

	fileContent, err := ioutil.ReadFile(walletFile)
	if err != nil {
		return err
	}

	gob.Register(elliptic.P256())
	decoder := gob.NewDecoder(bytes.NewReader(fileContent))
	err = decoder.Decode(&wallets)
	if err != nil {
		return err
	}

	ws.Wallets = wallets.Wallets

	return nil
}

func (ws *Wallets) SaveFile(nodeId string) {
	var content bytes.Buffer
	walletFile := fmt.Sprintf(walletFile, nodeId)

	gob.Register(elliptic.P256())

	encoder := gob.NewEncoder(&content)
	err := encoder.Encode(ws)
	if err != nil {
		log.Panic(err)
	}

	err = ioutil.WriteFile(walletFile, content.Bytes(), 0644)
	if err != nil {
		log.Panic(err)
	}
}
```