НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Фізико-математичний факультет

Кафедра математичного аналізу та теорії ймовірностей

На правах рукопису
УДК 512

До захисту допущено:
Завідувач кафедри
Олег КЛЕСОВ
«___»_____2024 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Страхова та фінансова математика»

зі спеціальності 111 «Математика»

на тему: «Дослідження криптографічної захищеності криптовалюти»

Виконав:
студент ІІ курсу, групи ОМ-21мп
Лозовий Богдан Ігорович _____

Науковий керівник:
Доцент кафедри математичного аналізу та
теорії ймовірностей,
кандидат фізико-математичних наук, доцент
Кубайчук Оксана Олексіївна _____

Рецензент:
к.т.н., доцент Спеціальної кафедри № 1
ІСЗЗІ КПІ
Самойлов Ігор Володимирович _____

Засвідчую, що у цій магістерській
дисертації немає запозичень з праць
інших авторів без відповідних посилань.
Студент _____

Київ – 2024 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Фізико-математичний факультет
Кафедра математичного аналізу та теорії ймовірностей

Рівень вищої освіти – другий (магістерський)

Спеціальність – 111 «Математика»

Освітньо-професійна програма «Страхова та фінансова математика»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Олег КЛЕСОВ
«___»_____2024 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Лозовий Богдан Ігорович

1. Тема дисертації «**Дослідження криптографічної захищеності криптовалюти**», науковий керівник дисертації Кубайчук Оксана Олексіївна, кандидат фізико-математичних наук, доцент, затверджені наказом по університету від « 13 » листопада 2023 р. № 5250-с.

2. Термін подання студентом дисертації - 11.01.2023.

3. Об’єкт дослідження – Криптографічна захищеність криптовалюти.

4. Предмет дослідження – Блокчейн, Bitcoin, Ethereum, криптографічна хеш-функція, хешування.

5. Перелік завдань, які потрібно розробити:

- Розглянути основні аспекти крипографичної захищеності криптовалют, проаналізувати існуючі проблеми та запропонувати методи боротьби.
- Розробити математичну модель, яка описує криптовалютні системи, виходячи з їх криптографічних принципів та структурних характеристик. Створити уніфіковані формули, котрі здатні представити різноманітні аспекти криптовалют Bitcoin та Ethereum.
- Описати алгоритми криптографічних хеш-функцій SHA256 та Keccak256, які використовуються в криптовалютах Bitcoin та Ethereum.

- Формалізувати математичні моделі рандомізованих алгоритмів, що зменшують ймовірність виникнення колізій в хеш-функціях.

6. Орієнтовний перелік публікацій:

- Bogdan LOZOVYI, Oksana KUBAYCHUK, PhD (Phys. &Math.), Associate Professor «BLOCKCHAIN TECHNOLOGY AND ITS VULNERABILITIES». Матеріали VI науково-практичної конференції курсантів (студентів), аспірантів, докторантів та молодих учених "Актуальні питання застосування спеціальних інформаційно-комунікаційних систем". Київ : ІСЗЗІ КПІ ім. Ігоря Сікорського, 2023. 394 с.

7. Дата видачі завдання 07.09.2023

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Формування основних завдань дослідження.	07.09.2023 - 21.09.2023	виконано
2	Пошук необхідних джерел для дослідження, їх аналіз та початок роботи над основною частиною дисертації.	22.09.2023 - 13.10.2023	виконано
3	Розробка математичного представлення криптовалюти Bitcoin	14.10.2023 - 04.11.2023	виконано
4	Розробка математичного представлення криптовалюти Ethereum	05.11.2023 - 25.11.2023	виконано
5	Аналіз зібраних даних та впорядкування інформації	26.11.2023 - 16.12.2023	виконано
6	Написання основного тексту дисертації. Редагування основної частини дисертації.	17.12.2023 - 02.01.2024	виконано
7	Формування висновків. Робота над презентацією за результатами дослідження.	02.01.2024- 08.01.2024	виконано

Студент Богдан ЛОЗОВИЙ

Науковий керівник Оксана КУБАЙЧУК

Реферат

Магістерська дисертація виконана на 97 сторінках, містить 23 ілюстрацій та 4 таблиць. Робота присвячена аналізу криптографічної безпеки криптовалют, з особливим акцентом на технології блокчейну.

Мета дослідження полягає у глибокому вивченні блокчейн-технологій та їх застосування у сфері криптовалют.

Об'єкт дослідження – блокчейн як основа для більшості криптовалют.

Предмет дослідження – блокчейн та криптографія у контексті криптовалют.

В роботі використані методи аналізу технологій блокчейну, криптовалют, та математичного моделювання.

Новизна полягає у детальному аналізі криптографічної безпеки та математичних основ криптовалют Bitcoin та Ethereum.

Ключові слова: блокчейн, криптовалюта, криптографія, безпека інформації, Bitcoin, Ethereum, криптографічна хеш-функція, хешування, блок, транзакція, майнер, доказ роботи (Proof of Work), доказ частки (Proof of Stake).

Abstract

The master's thesis is completed on 97 pages, contains 23 illustrations, and 4 tables. The work is dedicated to the analysis of cryptographic security of cryptocurrencies, with a special emphasis on blockchain technology.

The aim of the study is to deeply explore blockchain technologies and their application in the field of cryptocurrencies.

The object of the research is blockchain as the basis for most cryptocurrencies.

The subject of the study is blockchain and cryptography in the context of cryptocurrencies.

The work employs methods of analysis of blockchain technologies, cryptocurrencies, and mathematical modeling.

The novelty lies in the detailed analysis of the cryptographic security and mathematical foundations of cryptocurrencies Bitcoin and Ethereum.

Keywords: blockchain, cryptocurrency, cryptography, information security, Bitcoin, Ethereum, cryptographic hash function, hashing, block, transaction, miner, Proof of Work, Proof of Stake.

ЗМІСТ

ВСТУП

У цій магістерській дисертації основна увага приділена розгляду технології блокчейну, яка є однією з найбільш прогресивних та впливових технологій сучасності. В останні роки блокчейн здобув значну популярність, особливо в контексті криптовалют, таких як Bitcoin і Ethereum, що змінили уявлення про цифрові фінанси та безпеку даних. Незважаючи на свою новизну, блокчейн вже знайшов застосування у багатьох сферах, включаючи фінансові послуги, управління ланцюгами поставок, та навіть у державному секторі.

Ця дисертація спрямована на аналіз криптографічної безпеки криптовалют, з акцентом на використанні блокчейн-технологій. Вона розглядає блокчейн не тільки як технологію, але й як інноваційний інструмент, який може вирішити багато сучасних проблем, зокрема у питаннях безпеки, прозорості та ефективності виконання транзакцій. У дисертації розглядаються різні аспекти блокчейну, включаючи його структуру, типи, потенційні атаки на мережі, та роль криптографії у забезпеченні безпеки криптовалют. Також вона включає аналіз математичних основ криптовалют Bitcoin та Ethereum, що є важливим для розуміння їхньої функціональності та безпеки.

Завдяки швидкому розвитку цифрових технологій, важливість розуміння блокчейну та його застосування стає все більш актуальною. Україна також не залишається осторонь від цих тенденцій, демонструючи зацікавленість та активне впровадження блокчейн-технологій у різних секторах. У цьому контексті дисертація намагається відповісти на ключові питання, пов'язані з безпекою та ефективністю використання блокчейну в сфері криптовалют, а також оцінити його потенціал та майбутні перспективи.

РОЗДІЛ 1. Blockchain (Блокчейн)

1.1 Поняття блокчейн та доктринальні підходи до його визначення

Перш за все, варто відзначити, що початок періоду вивчення та дослідження криптографічних систем захисту даних припав саме на кінець XX – початок XXI століть. Проте найбільше значення та закріплення методика отримала саме у 2008 році, після чого власне і було створено один з найбільш відомих наразі блокчейнів - блокчейн криптовалюти Bitcoin. Тут хотілося б більш детально звернути увагу на саме поняття «блокчейну».

Так, дослідженням та аналізом поняття блокчейн та сферами його теоретичного і практичного застосування займались багато вчених, таких як: Стефанчук Р., Спасітєлєва С.О., Бурячок В.Л., Базанов С. Саме на їх розуміння поняття блокчейн ми зараз звернемо увагу. Так, науковець Стефанчук Р. вважає, що технологія блокчейн є розподіленим електронним реєстром даних, які є незмінними і достовірними. Реєстр базується на криптографічних алгоритмах, про що ми детальніші розглянемо у розділі 3, і зберігає дані щодо усіх виконаних операцій завдяки створенню блоків, на думку Стефанчука Р [1]. Науковці Спасітєлєва С.О. та Бурячок В.Л. вважають, що блокчейн є саме способом для збереження різних даних або операцій, які в свою чергу впорядковані за принципом ланцюга по блоках[2]. Базанов С. у своїй статті «Криптовалюта: терміни та скорочення» визначає блокчейн як вид розділеного реєстру, що зберігає в собі дані, які є незмінними і укомплектованими в окремі блоки [3].

Якщо звернутись до зарубіжних дослідників терміну блокчейн, то тут можна згадати про роботи таких науковців як Лампорт Л. та Піз М, Хонг-Нінг Даі та Зібін Женг. Науковці Хонг-Нінг Даі та Зібін Женг у своїй праці «Виклики та можливості блокчейну: опитування» розглядають блокчейн як публічний реєстр у якому усі виконані операції зберігаються в ланцюжку блоків, цей ланцюжок постійно збільшується, коли до цього додаються нові

блоки. Так, Хонг-Нінг Даі та Зібін Женг визначили також деякі ключові характеристики, якими наділений блокчейн, сюди вони відносять – децентралізація, постійність, анонімність та перевіреність [4]. Лампорт Л. та Піз М. вважають, що блокчейном є певний порядок блоків з даними щодо виконаних операцій у певній базі, яка в свою чергу побудована за певними алгоритмами[5].

На основі усіх вищенаведених визначень поняття «блокчейн» ми можемо сформувати власне розуміння даного терміну. Так, блокчейн – розділений електроний реєстр, який містить у собі дані, що впорядковані в окремі блоки. Варто також зазначити, що цілісність даних таких реєстрів забезпечується саме криптографічними методами та механізмом консенсусу.

1.2 Концепція Blockchain

Технологія блокчейн - це метод зберігання записів у вигляді ланцюжків блоків в мережі. Зазвичай це зберігання називається "цифровим реєстром".

Тут варто звернути увагу, що кожен блок включає в себе хеш-код, що розраховується із даних попереднього блоку, та додаткової інформації. Цією інформацією можуть слугувати дані про операції, угоди, попередні контракти, реєстрацію особи, підприємство, майно та інше. Іншими словами, це може бути майже будь-яка інформація. Якщо блок має за собою значну кількість інших блоків, то його неможливо замінити без перерахунку хешів всіх наступних блоків. Кожна транзакція в цьому реєстрі авторизована цифровим підписом власника, який підтверджує транзакцію.

Також важливим є те, якщо транзакція, яка була здійснена на одному вузлі, стає доступною для всіх інших вузлів у мережі. Це означає, що учасники мережі можуть переглянути цю транзакцію та вирішити прийняти чи відхилити її за результатами перевірки.

Рисунок 1.1 ілюструє нам структуру та склад ланцюга блоків. У кожному блоку блокчейну міститься хеш самого блоку, хеш попереднього блоку та різноманітні дані. Лише генезисний блок не має хеша попереднього блоку,

оскільки є першим у ланцюгу. Дані, що знаходяться всередині блоку, залежать від типу блокчейну. Наприклад, у блокчейні Bitcoin зберігається масив транзакцій, де кожна містить інформацію про відправника, отримувача і суму передачі коштів.

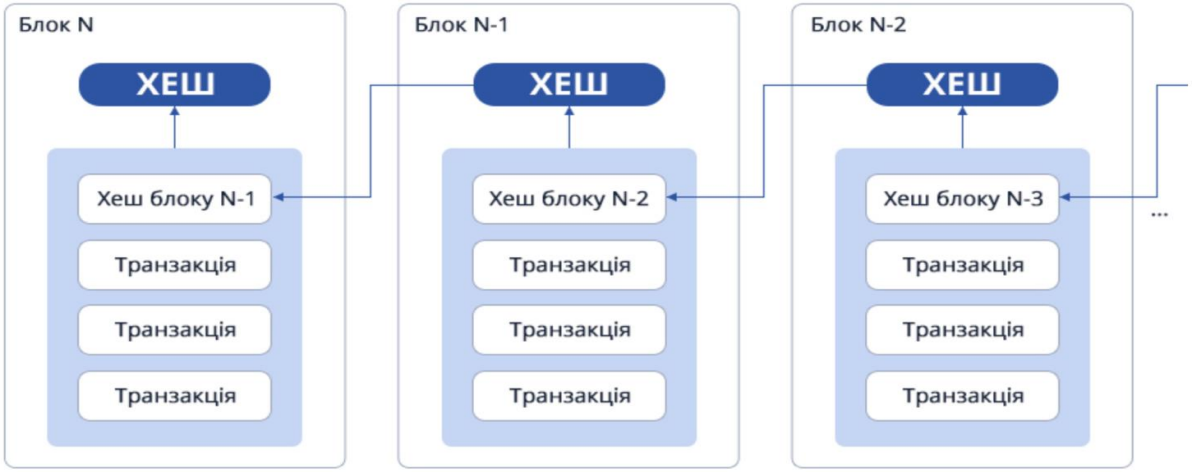


Рисунок 1.1 Структура та склад ланцюга блоків.

1.3 Класифікації типів блокчейнів

В даному аспекті перш за все варто сказати, що виділяють різні класифікації блочкчейні. В даному розділі ми розглянемо такі дві класифікації блокчейнів як - канадська та британська [6].

Канадська класифікація блокчейнів походить саме з поглядів на блокчейн засновника криптовалюти Ethereum Віталія Бутеріна. Так, канадський програміст вважає, що можна виділити такі три типи блокчейнів:

- *Публічні блокчейни:*

Цей тип блокчейну характеризується тим, що будь-хто і будь-коли може приєднуватися, читати, писати, переглядати та завантажувати протоколи. Валідаторам надаються стимули, а значне число учасників сприяє більшій безпеці блокчейну. Публічні блокчейни є повністю децентралізованими.

- *Приватні блокчейни:*

Приватні блокчейни відносять до частково децентралізованих. Вони управляються однією організацією, що часто призводить до нерівних прав учасників.

- *Консорціумні блокчейни:*

Ці блокчейни орієнтовані на проекти, не пов'язані з криптовалютою, і керуються групами організацій. Лише попередньо відібрані учасники з однаковими повноваженнями приймають участь [8].

Британська класифікація базується на поглядах наукового радника Великобританії Марка Валпорта. Так ним було виділено такі типи блокчейнів: відкриті публічні, закриті публіці, частково закриті. Проаналізувавши ці дві класифікації, ми можемо сказати, що вони є досить схожими та в деяких аспектах відображають одна одну.

1.4 Системи керування

Наразі науковці виділяють три види системи керування:

- Централізовані
- Децентралізовані
- Розподіленні

Системи з централізованим управлінням мають одну головну точку, де зосереджений весь контроль над процесами системи (див. рис. 1.2). Усі процеси та рішення здійснюються саме тут. Такі системи є вразливими, оскільки збій у роботі центру призводить до краху системи.

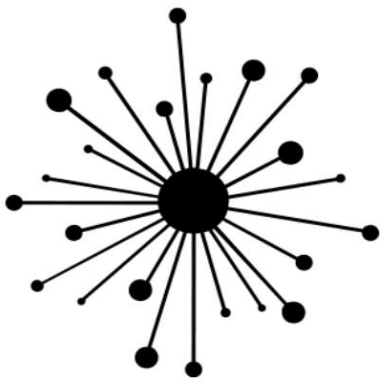


Рисунок 1.2 – Централізована система

Переваги централізованої системи:	Недоліки централізованої системи:
● Легка реалізація та швидкість прийняття рішень завдяки відсутності конфліктів між різними точками управління. ● Простота розширення масштабів системи. ● Централізована точка керівництва для прийняття рішень та управління даними.	● Ненадійність внаслідок використання лише однієї точки управління. ● Зайвий бюрократизм. ● Відсутність прозорості. ● Небезпека нападу на сервер, який може спричинити повну дестабілізацію системи. ● Збільшення загроз безпеки та порушення конфіденційності для користувачів.

Системи з децентралізованим управлінням мають кілька точок керівництва та різноманітність повноважень (див. рис. 1.3). Така система є більш стійкою, оскільки відключення однієї точки управління не призведе до збою всієї системи.

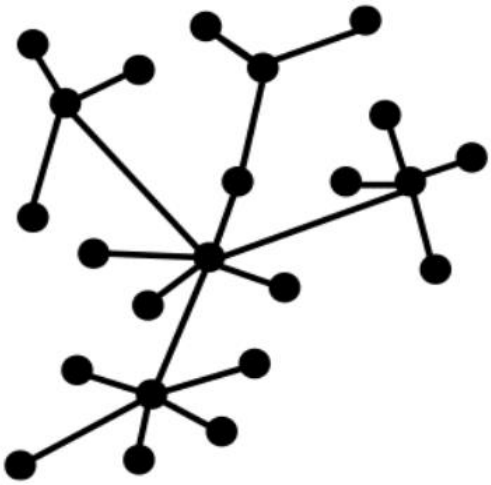


Рисунок 1.3 – Децентралізована система

Переваги децентралізованої системи:	Недоліки децентралізованої системи:
● Вища стійкість, зменшена ймовірність відмов у порівнянні з централізованою системою. ● Зближення прийняття рішень з користувачем. ● Можливість створення більш різноманітної та гнучкої системи.	● При розширенні масштабів може виникнути явище "дублювання". ● Збільшені витрати на обслуговування. ● Непостійна продуктивність при неправильній оптимізації.

У розподілених системах кожна точка системи є точкою управління (див. рис. 1.4), що робить систему стійкою до атак. Для зламу такої системи необхідно успішно атакувати більше половини точок управління, що є складною задачею.

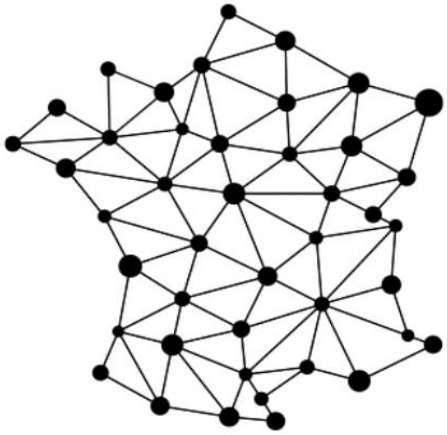


Рисунок 1.4 – Розподілена система

Переваги розділеної системи:	Недоліки розділеної системи:
● Немає посередників. ● Економічна невигідність взлому, що робить її найнадійнішою системою.	● Потреба у постійному вдосконаленні технологій і їхньому розвитку. ● Високі витрати на стабілізацію.

● Прозорість. ● Висока стійкість до відмов. ● Підтримує спільне використання ресурсів. ● Дуже гнучко масштабована.	

Централізовані системи відіграли ключову роль у встановленні перших мереж. Децентралізовані системи, які менше схильні до відмов і забезпечують швидкий доступ, значно поліпшили старі системи та активно використовуються сьогодні. Проте поміж інших, лише системи з розподіленою структурою здатні ефективно розподіляти ресурси та права по всій мережі, що забезпечує їх надзвичайну стійкість до відмов і робить більш прозорими, ніж централізовані та децентралізовані системи.

1.5 Структура блоків

Блок у блокчейні слугує як структурний контейнер, що групує транзакції для внесення в публічний реєстр, відомий як блокчейн. Блок має структуру, що включає заголовок, в якому містяться метадані, і за ним іде довгий перелік транзакцій, що складають основну частину його обсягу. Заголовок блоку становить 80 байт, тоді як середня транзакція - щонайменше 250 байт, а середній блок містить понад 500 транзакцій. Таким чином, повний блок з усіма транзакціями в 1000 разів більший за заголовок блоку.

Структура кожного блоку складається із заголовка (Head), в якому зберігається службова інформація, і корисної інформації (тіла блока Payload) - власне записи транзакцій (Рис. 1.5). Дані які містяться в блоках можуть бути різними в залежності від особливостей реалізації та потреб певного блокчейну.

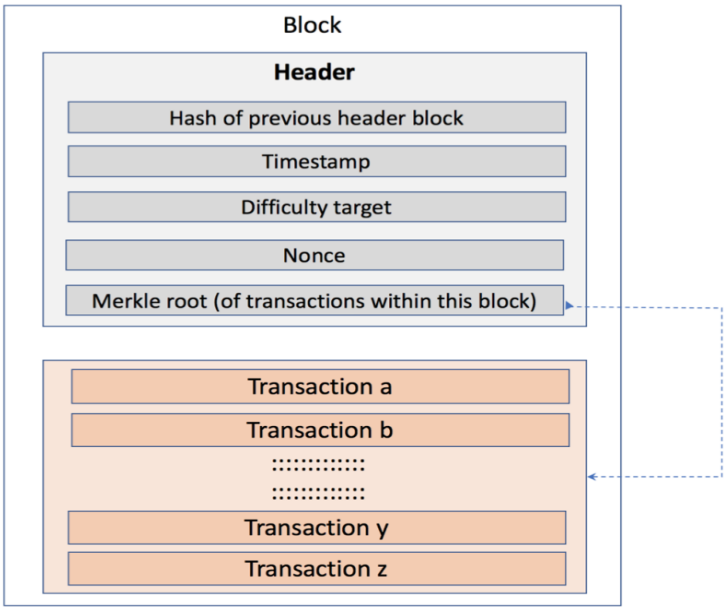


Рисунок 1.5 – Структура блока

Структура блоків в блокчейні є ключовим елементом, що забезпечує його функціональність та безпеку. Кожен блок в блокчейні може бути представлений як сукупність таких елементів: Хеш_{i-1}, Транзакції_i, Часова мітка_i та $Nonce_i$.

Хеш попереднього блоку (Хеш_{i-1}):

- В кожному блоці блокчейну вбудовано криптографічний хеш, який є посиланням на попередній блок у ланцюжку.
- Це створює послідовний ланцюг блоків, де зміна інформації в одному блоку вимагає перерахунку хешів у всіх наступних блоках, що забезпечує безпеку даних.

Список транзакцій (Транзакції_i):

- Блок містить пакет транзакцій, які були підтверджені у певний період часу.
- Кожна транзакція включає інформацію про передачу цінностей між учасниками мережі.

- Транзакції є основною частиною даних блокчейна і можуть включати різні типи інформації, залежно від призначення блокчейну.

Часова мітка (Часова мітка$_i$):

- Це дата та час створення блоку.
- Часові мітки важливі для підтримки послідовності ланцюга блоків і допомагають уникнути певних видів атак.

Nonce ($Nonce_i$):

- Це довільне число, яке використовується в алгоритмі підтвердження роботи (Proof of Work).
- Майнери змінюють значення Nonce для того, щоб знайти хеш блоку, який відповідає встановленим критеріям складності.
- Знаходження правильного Nonce є основою майнінгу нових блоків у багатьох блокчейнах.

Також важливо ще раз відзначити, що блок може також містити інші дані, такі як версія протоколу, ліміт розміру блоку, інформацію про майнера, що створив блок, та інші метадані, необхідні для конкретної реалізації блокчейна.

Хоча базова структура блоку є відносно стандартною, конкретні характеристики можуть варіюватися в різних типах блокчейнів, таких як Bitcoin, Ethereum, або інші альтернативні реалізації.

1.6 Заголовок блоку (Head) і тіло блоку (Payload)

Заголовок блоку складається з трьох наборів метаданих блоку.

- По-перше, це посилання на хеш попереднього блоку, який пов'язує цей блок з попереднім блоком в блокчейні.
- Другий набір метаданих, а саме складність, мітка часу і nonce.
- Третя частина метаданих - це корінь дерева Меркла, структура даних, яка використовується для ефективного узагальнення всіх транзакцій в блоці.

Структура заголовка блоку		
Розмір	Назва	Опис
4 байти	Версія блоку	Номер версії для відстеження оновлень програмного забезпечення/протоколу
32 байти	Хеш попереднього блоку	Посилання на хеш попереднього (батьківського) блоку в ланцюжку
32 байти	Корінь Меркла	Хеш кореня дерева Меркла транзакцій цього блоку
4 байти	Мітка часу	Приблизний час створення цього блоку
4 байти	Цільова складність	Цільовий показник складності алгоритму доказу роботи для цього блоку
4 байти	Nonce	Лічильник, який використовується для алгоритму підтвердження роботи

Зв’язок між попереднім і наступним блоком створюється за допомогою хешу заголовка блоку. Хеш транзакцій обчислюється за допомогою алгоритму - дерево Меркла (Рис.1.6.).

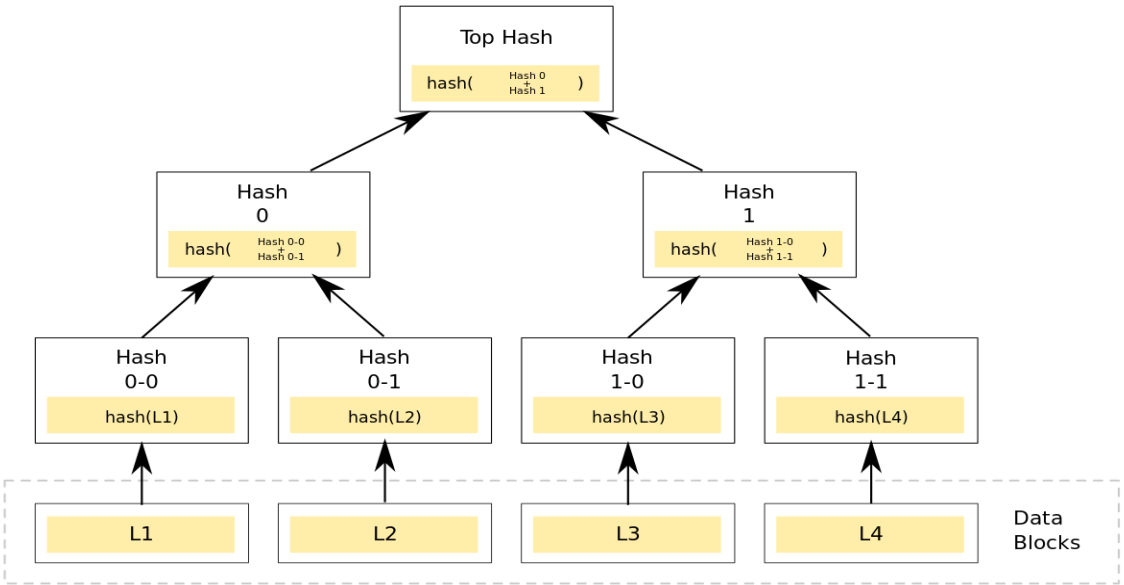


Рисунок 1.6 – дерево Меркла

Дерево Меркла — це спосіб зберігання потенційно великої кількості даних, який забезпечує користувачу простий спосіб перевірки, чи дані не були змінені. Для побудови дерева Меркла розглянемо хеш-функцію H та набір даних:

$$D = \{d_1, d_2, \dots, d_n)$$

Листя дерева представлені хеш-значеннями елементів у D: $H(d_1), H(d_2)$, …, тощо. Потім ми можемо побудувати дерево рекурсивно. Припустимо, весь шар L відомий, перший вузол наступного шару визначається хеш-значенням перших двох вузлів у L, другий вузол визначається хеш-значенням третього і четвертого вузлів у L і так далі. Якщо кількість вузлів у L непарна, то останній вузол у новому шарі просто дорівнює останньому вузлу у L. Для кращого розуміння розгляне графічний приклад (Рисунок 1.6). Розглянемо $D = \{L_1, L_2\ L_3, L_4)$, а Рисунок 1.6 показує відповідне дерево Меркла.

Корінь Меркла для $D = \{d_1, d_2, \dots, d_n)$ позначається як $R^H(D)$ і дорівнює верхньому хешу відповідного дерева Меркла.

РОЗДІЛ 2. Атаки на мережу блокчейн

2.1 Переваги та недоліки технології Blockchain

До переваг технології блокчейн ми можемо віднести:

- *Розподіленість:*

Блокчейни, як розподілена база даних, зберігають дані на тисячах пристроїв у розподіленій мережі нод, що забезпечує високу стійкість до технічних збоїв та атак зловмисника.

- *Стабільність:*

Підтверджені блоки майже не піддаються скасуванню. Це робить блокчейн ідеальним для зберігання фінансових записів та інших даних, де потрібна надзвичайна надійність та можливість проведення аудитів.

- *Система, яка не потребує посередників:*

Блокчейн-технологія дозволяє проведення транзакцій без посередників, таких як банки чи компанії, що випускають кредитні картки, завдяки використанню розподіленої мережі нод та процесу майнінгу. Це робить блокчейн системою "без довіри", мінімізуючи ризик довіри до конкретної організації та знижуючи витрати та комісії за транзакції [8].

Також науковець Григор'єв В. виділяє такі переваги технології блокчейн: прозорість операцій, анонімність(що полягає у використанні випадкового набору символів й знаків як адресу учасника транзакції) та криптозахист (тобто застосування електронного цифрового підпису або іншими словами проведення подвійної верифікації) [7].

До недоліків роботи технології блокчейн зазвичай відносять:

- *Приватні ключі:*

Блокчейн використовує криптографію з публічним ключем (асиметричну) для надання прав власності користувачам щодо їх монет чи інших блокчейн-даних.

- *Зберігання:*

Блокчейн-реєстри можуть зростати до величезних розмірів. Наприклад, розмір блокчейну Bitcoin складає близько 200 ГБ. Збільшення обсягу блокчейну може випереджати можливості жорстких дисків, що може вплинути на доступність та зберігання вузлів мережі.

2.2 Атаки на мережу блокчейн

Щодо мережі блокчейн, порушники можуть бути внутрішніми або зовнішніми. Внутрішні порушники, насамперед, представляють собою розробників конкретного проекту чи додаткового компонента (модуля) проекту, які бажають порушити безпеку системи на етапі розробки або експлуатації.

Зовнішні порушники атакують мережу як звичайні користувачі або учасники мережі (вузол або нода мережі), використовуючи віддалені атаки. Метою порушника може бути отримання можливості внесення змін в мережу

блокчейн згідно з власними намірами, заважання нормальній роботі мережі (перевантаження вузлів, блокування доступу, отримання контролю над мережею тощо) та отримання матеріальної або іншої вигоди через крадіжку даних або грошових ресурсів.

Для внутрішнього порушника ключовим є отримання доступу до редагування програмного забезпечення мережі або створення та впровадження власних компонентів (бекдорів), які надалі можна використовувати для зловживання мережею в майбутньому. Зовнішній порушник повинен мати значні обчислювальні та матеріальні ресурси, щоб виконати атаку, а також повинен бути знайомий з принципами функціонування мережі та її слабкими місцями. Успіх атаки залежить від кількості доступних обчислювальних ресурсів та знання помилок у мережевій безпеці.

2.3 Приклади типів атак на блокчейн

- *Атаки більшості (51%)*

У технології блокчейн для схвалення блоків потрібна підтримка 51% мережі (досягнення консенсусу). Іноді виникає ситуація, коли обробляються два блоки з конфліктуючими транзакціями. Ланцюжок зберігає тільки той блок, який отримав підтримку від консенсусу, а інший стає застарілим.

Припустімо, що 51% мережі потрапило під контроль зловмисників. Вони можуть використовувати свою більшість для скасування транзакцій або здійснення шахрайських операцій. Нові блокчейни особливо вразливі до атак більшості (51%). В 2019 році Ethereum Classic, гігантський учасник криптовалютного ринку, став жертвою атаки більшості (51%). Протягом трьох днів ціна криптовалюти знизилась на 16,4%, і шахраї здійснили атаку, яка призвела до втрати понад 1.000.000 USD.

- *Атака "Сівіли"*

Ця атака є методом, який використовується в однорангових мережах, де зловмисники створюють несправжні вузли, до яких приєднується жертва. Завдяки цим фальшивим вузлам, хакер може отримати контроль над більшістю

мережі (51%) і маніпулювати транзакціями у блокчейні, що нагадує 51% атаку, але базується на принципі контролю більшості.

- *Атаки з Подвійною Витратою*

Подвійна витрата відбувається, коли зловмисник витрачає одні й ті ж монети двічі: користувач 1 одночасно надсилає однакові 10 монет користувачеві 2 та користувачеві 3. У звичайних умовах блокчейн приймає лише одну транзакцію, яка отримала схвалення більшості, тому подвійна витрата завдає шкоди лише одній стороні. Багато блокчейнів вже мають засоби для уникнення подвійних витрат.

Існують різні види таких атак:

- *Атака "Race"* включає в себе одночасну відправку двох суперечливих транзакцій на однакову суму, де лише одна з них буде підтверджена. Ціль цієї атаки - переслати вкрадені кошти на конкретну адресу, але скасувати другу транзакцію. Така атака можлива в системах блокчейну, які дозволяють здійснювати транзакції без необхідності підтвердження.
- *Атака "Brute Force"* вимагає потужного обладнання і можлива, навіть при більше ніж одному підтвердженні транзакції. Шахраї в блокчейн-системах самостійно валідують блоки, що містять їх транзакції, для отримання необхідної кількості підтверджень, що спонукає продавця відправити товар. Однак, після цього зловмисник може змінити хід мережі, роблячи цю транзакцію недійсною.

- *Маршрутизаційні Атаки* Маршрутизаційні атаки не пов'язані безпосередньо з безпекою блокчейну, але вони використовують слабкості в протоколах з'єднання інтернет-провайдерів. Блокчейн залежить від інтернет-провайдерів, які взаємодіють за допомогою протоколу прикордонного шлюзу (BGP), що робить його вразливим для таких атак. Зловмисники, які мають доступ до цього протоколу, можуть розміщувати неправдиві маршрути. Це може призвести до блокування частини транзакцій і навіть до поділу мережі блокчейн на дві частини з різною кількістю учасників. Маршрутизаційні атаки можуть використовуватися

для затримання поширення нових блоків у мережі блокчейн, створюючи можливість для проведення атак з подвійною витратою, оскільки блоки залишаються невиявленими. Використання безпечних протоколів маршрутизації є ключовим для запобігання таким атакам, які можуть мати серйозні наслідки для стабільності блокчейн-мережі.

- *Атака на Приватний (закритий) Ключ*

Атака на приватний ключ у блокчейні пов'язана з асиметричним шифруванням, яке використовує публічні та приватні ключі для захисту конфіденційності даних. Ця система дозволяє зашифровувати повідомлення публічним ключем та розшифровувати їх приватним ключем. Однак, при необережному зберіганні приватного ключа, наприклад на комп'ютері, вразливому до хакерських атак, або через фішинг, ключ може потрапити до рук зловмисників, які зможуть використати його для доступу до криптовалюти користувача.

- *Атаки егоїстичного майнингу*

Егоїстичний майнінг - це тактика, де майнер приховує знайдені блоки, замість того, щоб додавати їх до публічного ланцюжка. Уявімо, що чотири майнери поділяють хешрейт порівну, але один з них зберігає знайдені блоки в секреті. Це дозволяє йому зберігати перевагу, завжди трохи випереджаючи публічний ланцюжок. Якщо його прихований ланцюжок виявиться довшим, він стане основним, і майнер отримає винагороду за всі блоки. Такі майнери можуть об'єднуватися в пули для збільшення потужності та потенційного проведення атак 51%.

Математичну модель цієї атаки ми представимо наступним чином:

$$p - \text{ймовірність з якою блок додається в чесний ланцюг;}$$

$$q - \text{ймовірність з якою зловмисник створює блок;}$$

$$q_z - \text{ймовірність з якою зловмисник змінює } z \text{ своїх блоків;}$$

$$q_z = \begin{cases} 1, & \text{якщо } p \leq q, \\ \left(\frac{q}{p}\right)^z, & \text{якщо } p > q. \end{cases}$$

За законом Пуасона ми можемо визначити ймовірність отримання зловмисником переваги над чесними учасниками мережі

$$P = 1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!}\left(1 - \left(\frac{q}{p}\right)^{(z-k)}\right), \text{де } \lambda = z\frac{q}{p}.$$

В дійсності ситуація має бути наступна: $p > q$, в той час як $p \leq q$ буде означати що зловмисник контролює більше 50% обчислювальних потужностей в мережі, а отже система більше не є децентралізованою.

- *Недоліки у смарт-контрактах*

Смарт-контракти застосовуються в блокчейні для автоматизації угод. Наприклад, при видачі криптовалюти під відсоткові ставки можна укласти смарт-контракт та зберегти відповідний запис у блокчейні. Найвідоміші та найбільші випадки викрадень включають в себе втрати в розмірі 30 мільйонів USD у Parity і 53 мільйони USD у DAO [9].

2.4 Запропоновані рішення проблем, які пов'язані з безпекою на надійністю блокчейну

- *Атака 51%:* Щоб протистояти атакам 51%, важливо залучати учасників мережі та підтримувати децентралізацію. Використання механізмів консенсусу, як-от доказ роботи (Proof of Work) або доказ частки (Proof of Stake), допомагає у розподілі обчислювальної потужності та мотивує учасників дотримуватися правил. Також, перехід від доказу роботи до доказу частки вважається ефективним у захисті від атак 51%.

- *Атаки Сівіли:* Існує кілька методів для запобігання атакам типу "Сівіла". Більшість криптовалют використовують алгоритм Proof of Work (доказ роботи) для цього. Майнінг, який ґрунтується на алгоритмі Proof of Work, полягає в приєднанні обчислювальних ресурсів до мережі криптовалюти, що забезпечує децентралізований підхід. Перевірка особистості всіх учасників мережі може виявитися заважливою для намагань підключити шахрайські вузли

до проекту, оскільки організаторові атаки доведеться підтверджувати реальність кожної неіснуючої особи. Система репутації передбачає нагородження учасників, які виявили свою добросовісність, специфічними визнаннями, такими як підвищений рейтинг.

- *Безпека смарт-контрактів:* Аудит і Навчання Коректне написання та аудит смарт-контрактів є ключовими для їх безпеки. Розробникам важливо пройти відповідне навчання для створення безпечних смарт-контрактів. Також корисним є застосування автоматизованих інструментів для аналізу коду, які можуть виявляти потенційні слабкості та допомагати уникати атак.

- *Егоїстичні майнінг-атаки:* Для запобігання егоїстичним майнінг-атакам ефективним є контроль за розподілом потужностей між майнінг-пулами. Коли майнінг-пул досягає більше ніж 40% загальної потужності, йому слід перерозподілити деяку частину майнерів до інших пулів, що сприятиме децентралізації мережі.

Протистояння методам соціальної інженерії: Освіта користувачів щодо виявлення фішингу та інших методів соціальної інженерії є важливою. Застосування додаткових заходів безпеки, як-от багатофакторна аутентифікація та використання апаратних гаманців, допомагає знижувати ризики небажаних витоків даних. Важливо також забезпечити безпечне зберігання приватних ключів і використовувати захищені поштові сервіси з двофакторною аутентифікацією.

- *Забезпечення безпеки криптовалютних бірж:* Криптовалютні біржі мають зосереджуватись на підвищенні безпеки своїх мереж, включаючи використання шифрування даних та імплементацію систем моніторингу для виявлення незвичайної активності. Ефективним заходом є також зберігання

значної частини активів у холодних гаманцях, які не мають постійного підключення до мережі, що допомагає уникнути потенційних загроз.

•*Протидія майнінг-ботнетам:* Оновлення антивірусних програм на регулярній основі є ключовим для запобігання використання комп'ютерів у майнінг-ботнетах. Виявлення нестандартної активності та непередбачуваного використання ресурсів також важливе для ідентифікації та запобігання несанкціонованим втручанням [10].

РОЗДІЛ 3. Криптографія в криптовалютах

3.1 Симетричне шифрування

Симетричне шифрування - це метод шифрування, де один і той самий секретний ключ використовується як для шифрування, так і для дешифрування інформації. Учасники, які використовують симетричне шифрування для спілкування, мають спершу поділитися ключем, щоб можна було використовувати його для розшифровки повідомлень.

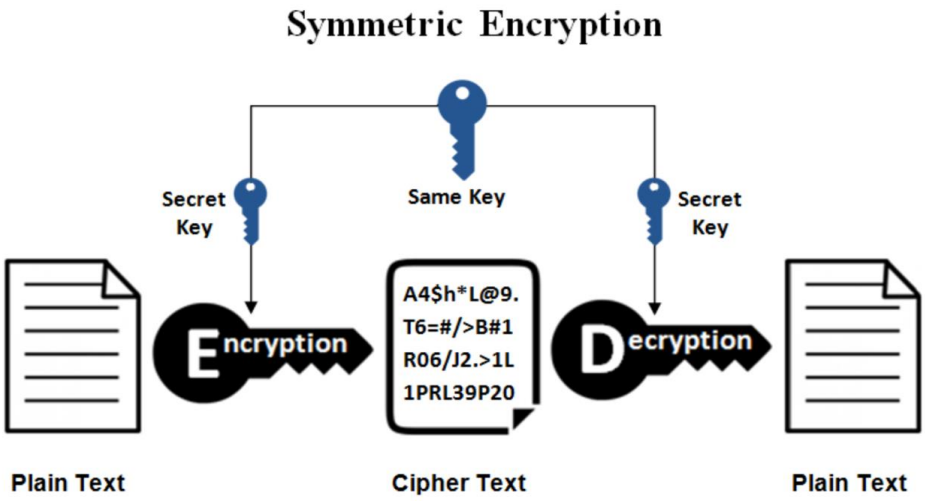


Рисунок 3.1 – Візуальне представлення симетричного шифрування

Схеми симетричної криптографії поділяються на блокові та потокові шифри.

3.1.1 Потоковий шифр

Потоковий шифр - вид шифру, у якому кожен біт даних зашифровується за допомогою гамування. Процес гамування передбачає накладення на інформацію гамми коду за строго визначеними правилами. Щоб розшифрувати дані, потрібне накладення тієї самої гами на зашифрований текст.

Розглянемо математичну модель алгоритму потокового шифрування RC4:
Застосовується S-блок : $S_0, S_1, S_2, \dots, S_{255}$
Ініціалізація S-блоку
Початкове заповнення: $S_0 = 0, S_1 = 1, S_2 = 2, \dots, S_{255} = 255.$
Заповнюємо 256-байтовий масив секретним ключем K_C:

$$K_0 = K_C, K_1 = K_C, K_2 = K_C, \dots, K_{255} = K_C.$$

$$j = 0$$

$for\ i = 0\ to\ 255$

$j = (j + S_i + K_i) mod 256$

Поміняти місцями S_i і S_j

Генерація псевдовипадкового байту:

$$i = (i + 1) mod 256; j = (j + S_i) mod\ 256.$$

Поміняти місцями S_i і S_j:

$$t = \big(S_i + S_j\big) mod\ 256; K = S_t$$

Зашифрування:

$$C_i = M_i + K$$

Приклад реалізації потокового шифру RC4:

```python
1 usage
def KSA(key):
    key_length = len(key)
    S = list(range(256))
    j = 0
    for i in range(256):
        j = (j + S[i] + key[i % key_length]) % 256
        S[i], S[j] = S[j], S[i]
    return S

1 usage
def PRGA(S):
    i = 0
    j = 0
    while True:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        K = S[(S[i] + S[j]) % 256]
        yield K
```

```python

2 usages
def RC4(key, text):
    S = KSA(key)
    keystream = PRGA(S)
    cipher = bytearray()
    for byte in text:
        cipher.append(byte ^ next(keystream))
    return cipher

# Приклад використання:

key = [0x4A, 0x65, 0x79, 0x2B, 0x31, 0x32, 0x38, 0x42, 0x69, 0x74, 0x4D, 0x6F, 0x64, 0x65]
plaintext = "Glory to Ukraine"  # вхідний текст

# Перетворюємо вхідний текст у байтовий формат
plaintext_bytes = bytearray(plaintext, 'utf-8')

# Зашифруємо текст за допомогою RC4
encrypted = RC4(key, plaintext_bytes)

```

```
# Виведемо зашифрований текст у вигляді шістнадцяткових чисел
encrypted_hex = ''.join(format(x, '02x') for x in encrypted)
print("Зашифрований текст (в шістнадцятковому форматі):", encrypted_hex)

# Розшифруємо зашифрований текст
decrypted = RC4(key, encrypted)

# Відновимо розшифрований текст
decrypted_text = decrypted.decode('utf-8')
print("Розшифрований текст:", decrypted_text)
```

RC4() › for byte in text

Run main

```
/Users/bogdanlozovoj/PycharmProjects/pythonProject2/venv/bin/python /Users/bogdanloz
Зашифрований текст (в шістнадцятковому форматі): f346d8b52c7f8bf96e664a28e3221933
Розшифрований текст: Glory to Ukraine

Process finished with exit code 0
```

3.1.2 Блоковий шифр

Сучасний блоковий шифр - це шифр із симетричним ключем, що розбиває перед шифруванням відкритий текст на n-бітні блоки і далі шифрує повідомлення блоками, тобто:

$$y = E_k(x) \text{ і } x = D_k(y),$$

де x, y - блоки відкритого та шифрованого текстів; k – секретний ключ шифру; E, D- функції шифрування і дешифрування (іншими словами, блоковий алгоритм - це алгоритм простої заміни блоків тексту фіксованої довжини). Алгоритми дешифрування і шифрування - інверсні, обидва працюють на одному і тому ж секретному ключі.

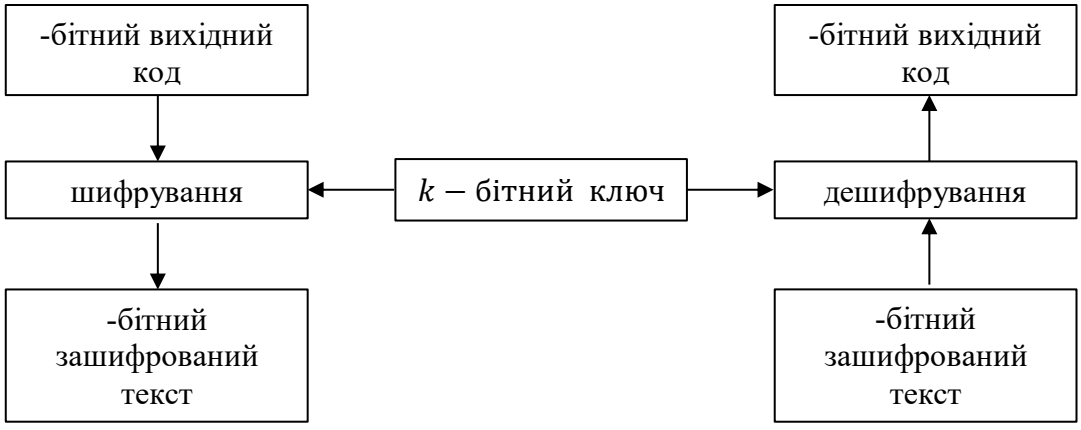


Рисунок 3.2 – Загальна ідея шифрування та дешифрування за допомогою блокового шифру

Якщо повідомлення містить менше n бітів, то його доповнюють додатковими даними (частіше нулями) до $n-$ бітового розміру; якщо текст має більше, ніж n біт, то його ділять на $n-$ бітові блоки.

Найчастіше блокові шифри обробляють блоки довжиною $n = 64$, 128, 256, 512 біт.

Блоковий шифр можна спроектувати так, щоб він діяв і як шифр підстановки, і як шифр перестановки:

1) якщо шифр спроектований як шифр підстановки, кожен біт відкритого тексту може бути замінений на 0 або 1 $=>$ вихідний текст і шифротекст можуть мати різне число одиниць.

2) якщо шифр спроектований як шифр перестановки, то біти відкритого тексту тільки міняються місцями.

У будь-якому разі, число можливих $n-$ бітових відкритих текстів дорівнює числу шифротекстів і дорівнює 2^n (оскільки кожен з n бітів блоку може дорівнювати 0 або 1).

3.2 Асиметричне шифрування

Асиметричне шифрування, відоме також як криптографія з відкритим ключем, відрізняється від симетричного шифрування використанням двох ключів. Один ключ є публічним і доступним для всіх, хто бажає надіслати зашифроване повідомлення, а інший - приватним, зберігається у секреті. Асиметричне шифрування здебільшого використовується в повсякденних

каналах зв'язку, особливо в Інтернеті. Популярні алгоритми шифрування з асиметричним ключем включають EIGamal, RSA, DSA, методи еліптичних кривих, PKCS.

Розглянемо принципи Асиметричного шифрування. Починаємо з шифрування числового повідомлення M. Наприклад, Макс хоче відправити повідомлення M Марії, щоб ніхто інший не міг його прочитати. Публічний та приватний ключі Макса позначені як ($K^{\text{Макс}}_{\text{публічний}}$, $K^{\text{Макс}}_{\text{приватний}}$), а ключі Марії - ($K^{\text{Марія}}_{\text{публічний}}$, $K^{\text{Марія}}_{\text{приватний}}$). Макс може зашифрувати M, використовуючи функцію шифрування C та публічний ключ Марії:

$$V = C(K^{\text{Марія}}_{\text{публічний}}, M)$$

Коли Марія отримує значення V, вона може розшифрувати повідомлення, використовуючи свій приватний ключ та функцію розшифрування D, асоційовану з C:

$$M = D(K^{\text{Марія}}_{\text{приватний}}, V)$$

Підпис повідомлення $M -$ це друге питання: як Марія може бути впевнена, що це справді Макс надіслав повідомлення? Рішення таке: Макс надсилає Марії як M, так і підпис S, де:

$$S = C(K^{\text{Макс}}_{\text{приватний}}, M)$$

Коли Марія отримує (S, M), вона може розшифрувати S за допомогою відкритого ключа Макса:

$$m = D(K^{\text{Макс}}_{\text{публічний}}, S)$$

Якщо $m = M$, це підтверджує, що Макс дійсно відправник повідомлення M.

Ключовим моментом є те, що приватний ключ завжди має залишатися приватним: лише його власник має знати його, тоді як публічний ключ можна вільно відправляти будь-кому.

Для використання асиметричного шифрування необхідно мати можливість визначити відкриті ключі. Один зі звичайних методів для цього полягає використанні цифрових сертифікатів у взаємодії між клієнтом і сервером. Сертифікат - це набір інформації, який служить для ідентифікації користувача та сервера, і містить різні дані, такі як назва організації, видавець сертифіката, адреса електронної пошти та країна користувача, а також відкритий ключ користувача. У випадку коли сервер та клієнт потребують безпечного зашифрованого зв'язку, вони взаємодіють через мережу з іншою стороною, яка надсилає копію свого сертифіката. З використанням цього сертифіката можна отримати відкритий ключ іншої сторони, і цей сертифікат також може бути використаний для однозначної ідентифікації власника.

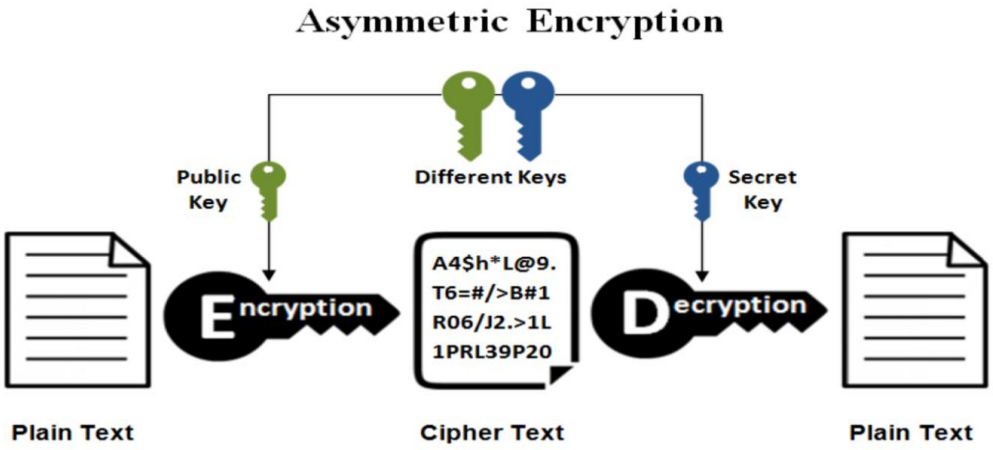


Рисунок 3.3 – Візуальне представлення асиметричного шифрування

3.3 Приватні та публічні ключі

Гаманець для криптовалюти Bitcoin містить набір пар ключів, кожна з яких включає в себе приватний та публічний ключ. Зазвичай приватний ключ k є випадковим числом. З використанням приватного ключа і односторонньої

функції множення на еліптичних кривих ми отримуємо публічний ключ K. За допомогою односторонньої функції криптографічного хеша з публічного ключа K ми отримуємо адресу біткоїна A (див. Рисунок 2.2).

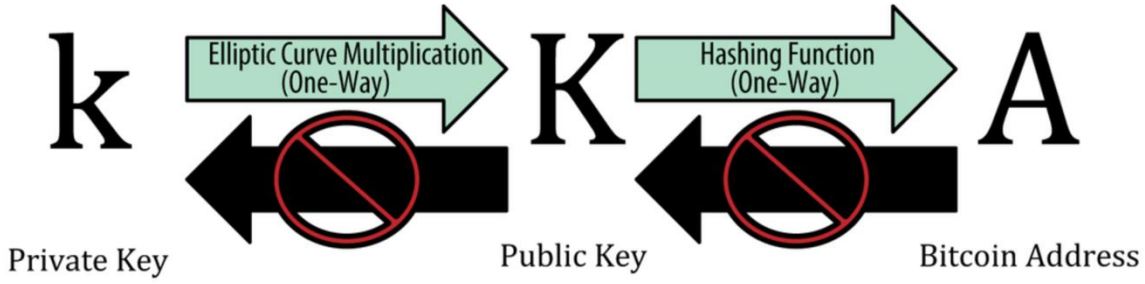


Рисунок 3.4 Ілюстрація взаємозв'язку між приватним ключем та публічним ключем і адресою гаманця для криптовалюти Bitcoin

3.3.1 Приватні ключі

Приватні ключі представляють собою прості числа, що вибираються випадковим чином. Володіння приватним ключем є важливим для керування активами, пов'язаними з відповідною адресою біткоїна. Використання приватного ключа включається в процес створення підпису, який підтверджує володіння операціями в транзакції. Можливо створити випадковий приватний ключ за допомогою лише монети, олівця та аркуша паперу. Для цього вам потрібно підкинути монету 256 разів і записати отриману послідовність гербів і решок у вигляді двійкового числа, яке потім можна використовувати як приватний ключ для вашого біткоїн-гаманця. Публічний ключ можна створити з цього приватного ключа.

Першим і найважливішим етапом при створенні ключів є знаходження надійного джерела випадковості або ентропії. Створення ключа для біткоїна суттєво зводиться до обрання числа від 1 до 2^{256}. Як саме вибирається це число не має значення, якщо процес не є передбачуваним або повторюваним. Програмне забезпечення біткоїна використовує генератори випадкових чисел операційної системи для створення 256-бітної ентропії (випадковості). Зазвичай цей генератор ініціалізується взаємодією з випадковими діями людини,

наприклад, обертанням мишки на кілька секунд. Для осіб із великою обережністю немає нічого кращого, ніж використання кубиків, олівця та паперу.

Щоб бути точним, приватний ключ може бути будь-яким числом від 1 до $n-1$, де n- стала ($n = 1{,}158 * 10^{77}$, трохи менше 2^256), яка визначена як порядок еліптичної кривої, що використовується у біткоїні. Для створення такого ключа ми випадковим чином обираємо 256-бітне число і перевіряємо, чи воно менше за $n-1$. З точки зору програмування, це зазвичай досягається шляхом подачі більшого рядка випадкових бітів, зібраних з криптографічно безпечного джерела випадковості, у хеш-алгоритм SHA256, який зручно перетворюється у 256-бітне число. Якщо результат менший за $n-1$, то ми отримуємо відповідний приватний ключ. Якщо ні, ми просто повторюємо процес з іншим випадковим числом [11].

На рисунку 3.5 наведено секретний ключ k, який був згенерований випадковим чином у форматі шістнадцяткового числа. Цей ключ складається з 256 двійкових цифр, які відображені як 64 шістнадцяткові цифри, по чотири біти кожна.

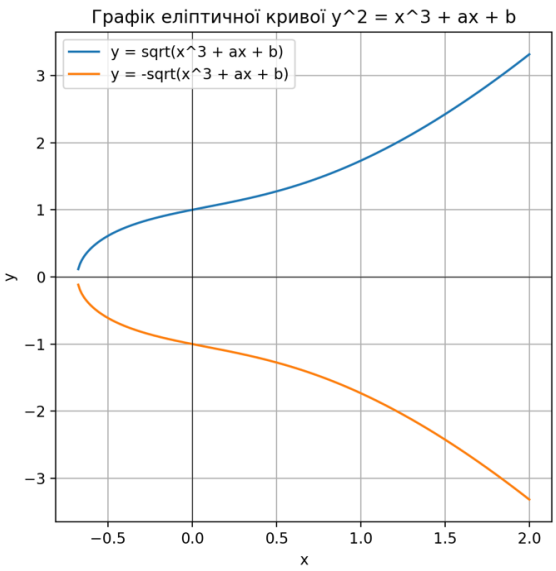
Рисунок 3.5

3.3.2 Публічний ключ

Публічний ключ обчислюється на основі приватного ключа за допомогою еліптичного множення ($K = k * G$), яке є незворотнім: де k - закритий ключ, G - постійна точка (точка генератора,) а K - отриманий публічний ключ. Обернена операція, яка іноді називається знаходженням дискретного логарифма - обчислення k, якщо відомо K - настільки ж складна, як і перебір всіх можливих значень k, тобто пошук грубою силою.

Розглянемо криптографію еліптичних кривих більш детально. Криптографія еліптичних кривих - це тип асиметричної криптографії або криптографії з відкритим ключем, заснований на задачі дискретного логарифмування, що виражається додаванням і множенням в точках еліптичної кривої. На даний момент в криптографії велика увага приділяється використанню еліптичних кривих. Еліптична криптографія (ЕК) - це галузь криптографії, яка вивчає асиметричні криптосистеми, які використовують еліптичні криві над скінченними полями як основу.

Еліптична крива - це кубічна крива, що представлена на площині XY декартової системи координат, із заданим рівнянням $y^2 = x^3 + ax + b$, яке не має особливих точок.




```python
import matplotlib.pyplot as plt
import numpy as np
a = 1
b = 1

x = np.linspace(-2, 2, 400)

y_positive = np.sqrt(x**3 + a*x + b)
y_negative = -np.sqrt(x**3 + a*x + b)

plt.figure(figsize=(6, 6))
plt.plot(x, y_positive, label='y = sqrt(x^3 + ax + b)')
plt.plot(x, y_negative, label='y = -sqrt(x^3 + ax + b)')
plt.title('Графік еліптичної кривої y^2 = x^3 + ax + b')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid(True)
plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)
plt.show()

```

Рисунок 3.6 Графіку еліптичної кривої

У сучасних криптосистемах застосовується рівняння $y^2 = x^3 + ax + b \ (mod\ p)$ (1), де a і b належать до поля $GF(p)$, при цьому $4a^3 + 27b^2 (mod\ p) \neq 0$, де p є простим числом. Множина $E_p(a, b)$ включає всі точки $(x, y), x \geq 0, p > y$, та вони задовольняють рівнянню (1), а також точку у нескінченності O, яка служить нульовою точкою для площини еліптичних кривих. Кількість точок на еліптичній кривій позначається як $E_p(a, b)$.

Множина точок $E_p(a, b)$ буде мати такі властивості:

- якщо $P = (x, y)$ є точкою еліптичної кривої, то $(x, y) + (x, -y) = O$; точка $(x, -y)$ позначається як $-P$;
- якщо $P = (x_1, y_1)$ і $Q = (x_2, y_2)$, де $P \neq Q$, то $P + Q = (x_3, y_3)$ обчислюється так:
  $x_3 = \frac{y_2 - y_1}{x_2 - x_1} - x_1 - x_2;\ y_3 = -y_1 - \frac{(y_2 - y_1)(x_2 - x_1)}{x_2 - x_1}$ (2)
- якщо $P = Q$, то подвоєння точки $P = (x_3, y_3)$ обчислюється за формулою:

$$x_3 = \left(\frac{\frac{3x_1^2 + a}{2}}{y_1} \right)^2 - 2x_1;\ \ y_3 = -y_1 - \frac{\frac{3x_1^2 + a}{2}}{y_1} \quad (3)$$

Одним з найпоширеніших застосувань криптографічних еліптичних кривих є використання їх для розподілу ключів. У цьому криптографічному протоколі обміну ключами, що використовує еліптичну криву, маються наступні кроки:

- Користувач повинен вибрати та сповістити всім параметри еліптичної кривої та точку G, яка є генеруючою точкою на цій кривій.
- Перший користувач обирає ціле число k і знаходить точку $P_A = k * G$ (іншими словами, додає точку G до себе k разів).
- Другий користувач вибирає число m і обчислює точку $P_B = m * G$. Після цих операцій користувачі обмінюються своїми результатами, а спільний секретний ключ для них стає точка $k * m * G$.

Для того, щоб розкрити цей протокол, потрібно з відомими значеннями $k * G$ та G обчислити $k < p$. Ця задача є аналогічно складною, як і задача дискретного логарифмування алгоритму RSA де ми можемо з меншими зусиллями обчислити P з відомими k та G, але складно визначити k з відомими P та G.

Алгоритм створення надійної випадкової кривої:

1. Вибираємо випадкове число p яке є простим.

2. Обираємо випадкові числа a і b, де $a, b \neq 0$ та $(4a^3 + 27b^2)(mod\ p) \neq 0$. Для оптимізації обчислень можна обирати випадково лише b, а a встановлювати як невелике ціле число.

3. Визначаємо кількість точок на кривій $n = E_p(a, b)$. Важливо, щоб n мало досить великий простий дільник q (розмір підмножини точок на кривій).

4. Перевіряємо, чи виконується умова $(p^k - 1)mod1 \neq 0$ для всіх $k, 0 < k < 32$. Якщо умова не виконується, повертаємось до кроку 2.

5. Перевіряємо, чи $q = n$. Якщо умова не виконується, повертаємось до кроку 2.

6. Знаходимо точку G, яка є генератором підмножини точок q. Якщо $q = n$, то будь-які точки (крім O) будуть генераторами. Якщо $q < n$, обираємо випадкові точки G', поки умова $G = \left[\frac{n}{q}\right] G' \neq 0$ не виконається. [19]

3.4 Електронний цифровий підпис

У сучасному світі електронний цифровий підпис (ЕЦП) широко використовується для захисту електронних документів від фальсифікації та несанкціонованих змін. Як у сфері підприємництва, так і в особистих цілях, електронний підпис виконує ті ж самі функції, що й звичайний підпис людини. Наявність електронного підпису гарантує, що документ створений конкретно

цією особою, тому документ із цифровим підписом має таку ж юридичну силу, як і його паперовий аналог. [16]

Вибір ключів для ЕЦП:

1. Нам потрібно вибрати 2 великих простих числа p і q.
2. Далі ми обчислюємо $N = p * q \;\; i \; \varphi(N) = (p-1)(q-1)$.
3. Вибираємо E, що задовольняє умовам $E < \varphi(N)$, НОД $\big(E, \varphi(N)\big) = 1$.
4. Обчислюємо D, що задовольняє умові $E * D \equiv 1\big(mod\varphi(N)\big)$.

Пара чисел (D, N) зберігається у відправника як секретний ключ який буде використовуватись для підписання. Пара чисел (E, N) є відкритим ключем, яку одержувачам передає відправник для того аби вони могли перевірити його цифровий підпис.

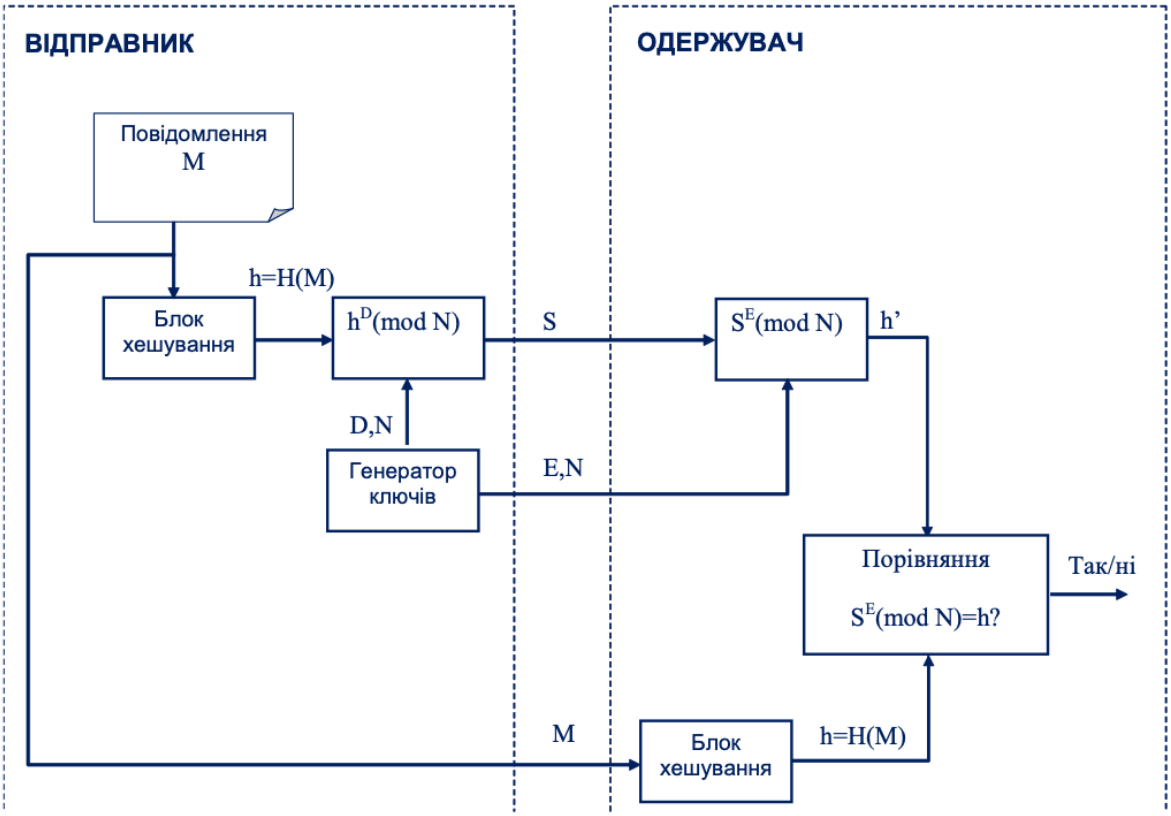


Рисунок 3.7 - Схема підписання та перевірки ЕЦП

Еліптична крива алгоритму цифрового підпису, відома як ECDSA (Elliptic Curve Digital Signature Algorithm). ECDSA приймає повідомлення msg та приватний ключ (Private Key) як вхідні дані і за допомогою цього алгоритму створює підпис $\{r, s\}$:

1. $h = hash(msg)$
2. Генерує випадкове число k (приватний ключ) безпечним способом у діапазоні $[1, \dots n]$
3. $R = k * G$ і береться його x-координата r, де G є фіксованою точкою на еліптичній кривій, відому і використовувану всіма учасниками системи.
4. Підпис $s = k^{-1} * (h + r * k) mod\ (n)$
5. Повертає підпис $\{r, s\}$.

Для перевірки підпису ECDSA береться підписане повідомлення msg + підпис $\{r, s\}$, створений за допомогою алгоритму підпису + публічний ключ (PubKey), що відповідає приватному ключу підписувача.

Вихід це булеве значення: дійсний чи недійсний підпис.

1. $h = hash(msg)$
2. $R' = (h * s_1) * G + (r * s_1) * PubKey$, де $s_1 = s^{-1}(mod\ (n))$
3. Береться його x-координата r' з R'
4. Обчислюється результат перевірки підпису, порівнюючи, чи $r' = r$

Основний принцип перевірки підпису полягає у використанні публічного ключа для відновлення точки R' і підтвердження того, що це саме та точка R, яка була випадково вироблена під час процедури підпису.

3.5 Смарт контракти

Смарт-контракт - це автоматизований протокол транзакцій, який виконує визначені умови контракту (Szabo, 1997). Він був запропонований вже давно, і тепер ця концепція може бути реалізована за допомогою блокчейну. У блокчейні смарт-контракт - це фрагмент коду, який може виконуватися майнерами автоматично. Сьогодні з'являється все більше платформ для розробки смарт-контрактів, і смарт-контракти можуть набувати все більшої

функціональності. Блокчейн можна використовувати в багатьох сферах, таких як IoT (Christidis and Devetsikiotis, 2016) та банківські послуги (Peters and Panayi, 2015).

Ми класифікуємо дослідження смарт-контрактів на два типи: розробка та оцінка. Розробка може бути розробкою смарт-контрактів або розробкою платформи для смарт-контрактів. Зараз багато смарт-контрактів розгорнуто на блокчейні Ethereum (Wood, 2014). Що стосується розробки платформ, то з'являється багато платформ для розробки смарт-контрактів, таких як Ethereum (Wood, 2014) і Hawk (Kosba et al., 2016). Оцінювання означає аналіз коду та оцінку продуктивності. Помилки в смарт-контрактах можуть призвести до катастрофічних збитків. [17]

3.6 Proof of Work і Proof of Stake

У традиційних мережах часто потрібно залучення третьої сторони для затвердження угод, але блокчейн дозволяє користувачам взаємодіяти безпосередньо з розподіленою базою даних, доступною для всіх. Для цього необхідно виконати умову консенсусу, зазвичай через спеціальні алгоритми перевірки транзакцій, серед яких найпоширеніші - PoS і PoW.

У системі Proof of Work від користувача вимагається розв'язання математичної задачі, яка має бути складною для розв'язання, але легкою для перевірки. У таких криптовалютах, як біткойн та лайткойн, алгоритм PoW існує у певній формі вузлів, що конкурують між собою, а блок має бути прийнятий в блокчейн тільки одним з них. За допомогою криптографії досягається такий ефект. Для розв'язання задачі потрібно знайти хеш, тобто одне вірне значення серед великої кількості. Для цього потрібно використовувати потужне комп'ютерне обладнання. Багато майнерів намагаються знайти це значення, але тільки перший, хто знаходить його, отримує винагороду. Чим потужніше обладнання, тим більша ймовірність знайти хеш першим.

Алгоритм Proof of Stake представляє собою принципово інший метод досягнення консенсусу та перевірки транзакцій. Мета схожа на PoW, але умова консенсусу тут кардинально відрізняється. У алгоритмі Proof of Stake творець нового блоку вибирається не шляхом розв'язання математичних задач, а детерміністично, залежно від кількості цифрових активів (токенів), які в нього є. Плата за видобутий блок відсутня, а за рахунок плати за транзакції підтримується хешрейт мережі. Алгоритм Proof of Stake працює наступним чином:

1. Користувачу потрібно купити валюту, яка працює за алгоритмом Proof of Stake;
2. Потім завантажити програму-майнер та запустити її в режимі форжингу та синхронізувати ланцюг;
3. Очікувати синхронізації та запустити майнинг. [18]

3.7 Сід-фраза

Сід-фраза виконує функцію альтернативного представлення приватних ключів, які традиційно являють собою довгий рядок з 64 символів, включаючи цифри та літери. Це робить ключ більш легким для запам'ятовування, оскільки люди краще запам'ятовують слова, ніж послідовність випадкових символів. Зазвичай сід-фраза гаманця містить 12 або 24 англійські слова, і важливо пам'ятати їх порядок для відновлення доступу до гаманця. Безпека сід-фрази є ключовою, адже володіння нею еквівалентно володінню приватним ключем та, отже, контролю над активами в гаманці.

3.8 Хешування

Хеш-функції - це математичні алгоритми, які приймають вхідні та видають на їхній основі вихідні дані фіксованого розміру, відомі як хеш-значення або хеш-код. Хеш-функції відіграють важливу роль у сучасній криптографії.

- *Визначеність:* для одних і тих самих вхідних даних хеш-функція завжди видаватиме одне й те саме хеш-значення - заради забезпечення узгодженості та передбачуваності.
- Фіксований розмір виведення: хеш-функції створюють хеш-значення фіксованої довжини, незалежно від розміру вхідних даних. Наприклад, звичайна хеш-функція SHA-256 завжди генерує 256-бітове хеш-значення.
- *Незворотність:* хеш-функції спроектовані як односторонні, що надзвичайно ускладнює відновлення вихідного повідомлення з хеш-коду.
- *"Ефект лавини":* невелика зміна вступних даних призведе до значної зміни хеш-значення. Це гарантує, що навіть невелика зміна ввідних даних призведе до зовсім іншого хеш-коду. [14]
- *Стійкість до колізій: хеш*-функція має мінімізує вірогідність того, що два різні аргументи дадуть одне й те саме значення хеш-функції. Сучасні хеш-функції роблять цю подію вкрай малоймовірною. [13]

Таким чином, хеш-функції є незамінним інструментом, коли йдеться про цілісність даних, зберігання паролів, цифрові підписи та безпеку блокчейнів.

Хеш-функція - це математичний алгоритм, який приймає дані довільної довжини на вхід і перетворює їх у зашифрований текст фіксованої довжини на виході. Цей результат називається дайджестом повідомлення, хеш-значенням, хеш-кодом або просто хешем. Більш формально, хеш-функція - це математична функція:

$$H: D \to R, \text{де область } D = \{0,1\}^* \text{ і } R = \{0,1\}^n \text{ для деякого } n \geq 1,$$

яка відображає числове значення на вході m довільної довжини у стисле числове значення на виході h фіксованої довжини. Тобто: $h = H(m)$. Хеш-функція, яка задовольняє деяким додатковим вимогам, щоб її можна було використовувати для криптографічних застосувань, називається криптографічною хеш-функцією. Ці функції є важливими конструкціями, які

мають безліч варіантів використання. Основними сферами їх застосування є захист збережених паролів, автентифікація повідомлень, цифрові підписи, а отже і сертифікати.

Хеш-функція, яка задовольняє деяким додатковим вимогам, щоб її можна було використовувати для криптографічних додатків, відома як криптографічна хеш-функція. Ці функції є важливими конструкціями, які мають безліч варіантів використання. Основними сферами їх застосування є захист збережених паролів, автентифікація повідомлень, цифрові підписи і, відповідно, сертифікати. Криптографічні хеш-функції поділяються на два класи: хеш-функції без ключа, також відомі як код виявлення маніпуляцій (MDC) або код автентифікації повідомлень (MAC) з одним параметром - вхідним повідомленням - і хеш-функції з ключем з двома різними входами - вхідним повідомленням і секретним ключем. Як правило, термін "хеш-функції" відноситься до хеш-функцій без ключа.

Деякі приклади криптографічних хеш-алгоритмів:

- Сімейство SHA (Secure Hash Algorithm) - опубліковане Національним інститутом стандартів і технологій (NIST) як Федеральний стандарт обробки інформації США (FIPS). Це сімейство визначає шість різних хеш-функцій: SHA-0, SHA-1, SHA-224, SHA-256, SHA-384 і SHA-512. Перші чотири працюють з 512-бітними блоками повідомлень, розділеними на 32-розрядні слова, а останні дві - з 1024-бітними блоками, розділеними на 64-розрядні слова.
- Сімейство MD (Message Digest) - складається з MD2, MD4, MD5 і MD6, розроблене Рональдом Рівестом для забезпечення безпеки RSA і прийняте як Інтернет-стандарт RFC 1321.
- RIPEMD - сімейство криптографічних хеш-функцій, засноване на принципах побудови MD4, розроблене Гансом Доббертіном, Антуном Босселаерсом і Бартом Пренілом в дослідницькій групі COSIC в

Левенському університеті Католицької церкви. RIPEMD-160 створює хеш-дайджест довжиною 160 біт (20 байт).

- Whirlpool - розроблена Вінсентом Райменом і Пауло С. Л. М. Баррето, ця хеш-функція заснована на істотно модифікованій версії Advanced Encryption Standard (AES). Whirlpool створює хеш-дайджест розміром 512 біт (64 байти).
- BLAKE - хеш-функція, подана на конкурс хеш-функцій NIST Jean-Philippe Aumasson, Luca Henzen, Willi Meier і Raphael C.-W. Phan. Вона заснована на потоковому шифрі ChaCha Дена Бернстайна, але перед кожним раундом ChaCha додається переставлена копія вхідного блоку, XOR з константами раунду.
- Curl-P - хеш-функція, яка раніше використовувалася в схемі підпису IOTA (ISS). IOTA - криптовалюта, розроблена для використання з Інтернетом речей (IoT) та автомобільними екосистемами. Інтернет речей (або IoT - Internet of Things) - це ідея мережі для передачі даних між фізичними об'єктами, так званими "речами", які обладнані вбудованими технологіями і засобами для взаємодії один з одним або з навколишнім середовищем. Ця концепція передбачає можливість переформатування економічних та соціальних процесів, що може зменшити потребу в участі людини у деяких діях і операціях. ISS базується на одноразових підписах Winternitz, але на відміну від традиційного Winternitz, в IOTA користувачі підписують хеш повідомлення. Winternitz One Time Signature (WOTS) - це квантово-стійка схема цифрового підпису, яка використовує відносно невеликі розміри ключа та підпису. Таким чином, безпека ISS покладається на її криптографічну хеш-функцію, якою стала Curl-P-27.

SHA-256 є, мабуть, найвідомішою з усіх криптографічних хеш-функцій, оскільки вона широко використовується в технології блокчейн. Вона використовується в оригінальному протоколі Bitcoin Сатоши Накамото.

MD5 є одним з найпоширеніших алгоритмів хешування. Проте, через свою вразливість до колізій (коли дві різні вхідні послідовності можуть мати однаковий хеш), його використання в критичних застосунках не рекомендується.

3.8.1 Алгоритм хеш функції MD5

Крок 1: На цьому кроці ми повинні вирівняти поток

Процес вирівнювання потоку для MD5 полягає у доповненні вихідного повідомлення так, щоб його довжина стала близькою до кратної 512 бітам. В нас є повідомлення з довжиною λ біт яке ми доповнюємо до повідомлення з довжиною λ' біт:

$$\lambda' = \lambda \cdot 512 + 448.$$

Ми отримали, що довжина повідомлення яке ми розширили меньша на 64 розряди ніж кратне 512 число розрядів.

Для вирівнювання потоку, ми дописуємо біт 1 до кінця початкового потоку, потім біти 0 і це відбувається до моменту, коли потік не стане довжиною рівною 448 за $mod(512)$.

Крок 2: Додавання бітів довжини

В другому кроці обробки MD5, до розширеного на попередньому етапі повідомлення додається 64-бітна інформація про його первісну довжину. Спершу вносять молодші, а потім старші 4 байти цього 64-бітного числа. Якщо довжина вихідного повідомлення перевищує $2^{64} - 1$, то додають лише молодші 4 байти. Після цього повідомлення стає кратним 512 бітам і складається з послідовності 512-бітних блоків, $Y_0, Y_1, \dots, Y_{L-1}$ кожен з яких містить шістнадцять 32-бітних "слів". $L \times 512 -$ це загальна довжина повідомлення яке було розширене.

Крок 3: Ініціалізуємо MD буфер

У третьому кроці ініціалізації MD буфера алгоритму MD5 використовується 128-бітний буфер, поділений на чотири 32-бітні регістри (A, B, C, D), які представляють слова. Кожне слово складається з 32 послідовних бітів або 4 послідовних байтів, де молодший байт є першим. Ці регістри ініціалізуються специфічними шістнадцятковими числами для підготовки до подальшої обробки повідомлення.

$$A = 01\ 23\ 45\ 67; 67452301h$$

$$B = 89\ AB\ CD\ EF;\ EFCDAB89h$$

$$C = FE\ DC\ BA\ 98;\ 98BADCFEh$$

$$D = 76\ 54\ 32\ 10; 10325476h$$

Крок 4: У четвертому кроці алгоритму MD5 відбувається обробка блоків, кожен з яких містить 512 біт або 16 слів. Алгоритм включає чотири цикли обробки для кожного блоку, блок позначається як H_{MD5} причому кожен цикл використовує свою унікальну елементарну логічну функцію, хоча загальна структура циклів схожа. Ці цикли є ключовою частиною MD5, що дозволяє ефективно обробляти введені дані.

$$f_F = (B\&C) \vee (not\ B\&D)$$

$$f_G = (B\&D) \vee (C\ \&\ not\ D)$$

$$f_H = B \oplus C \oplus D$$

$$f_I = C \oplus (B\ \&\ not\ D)$$

Операції є побітовими.

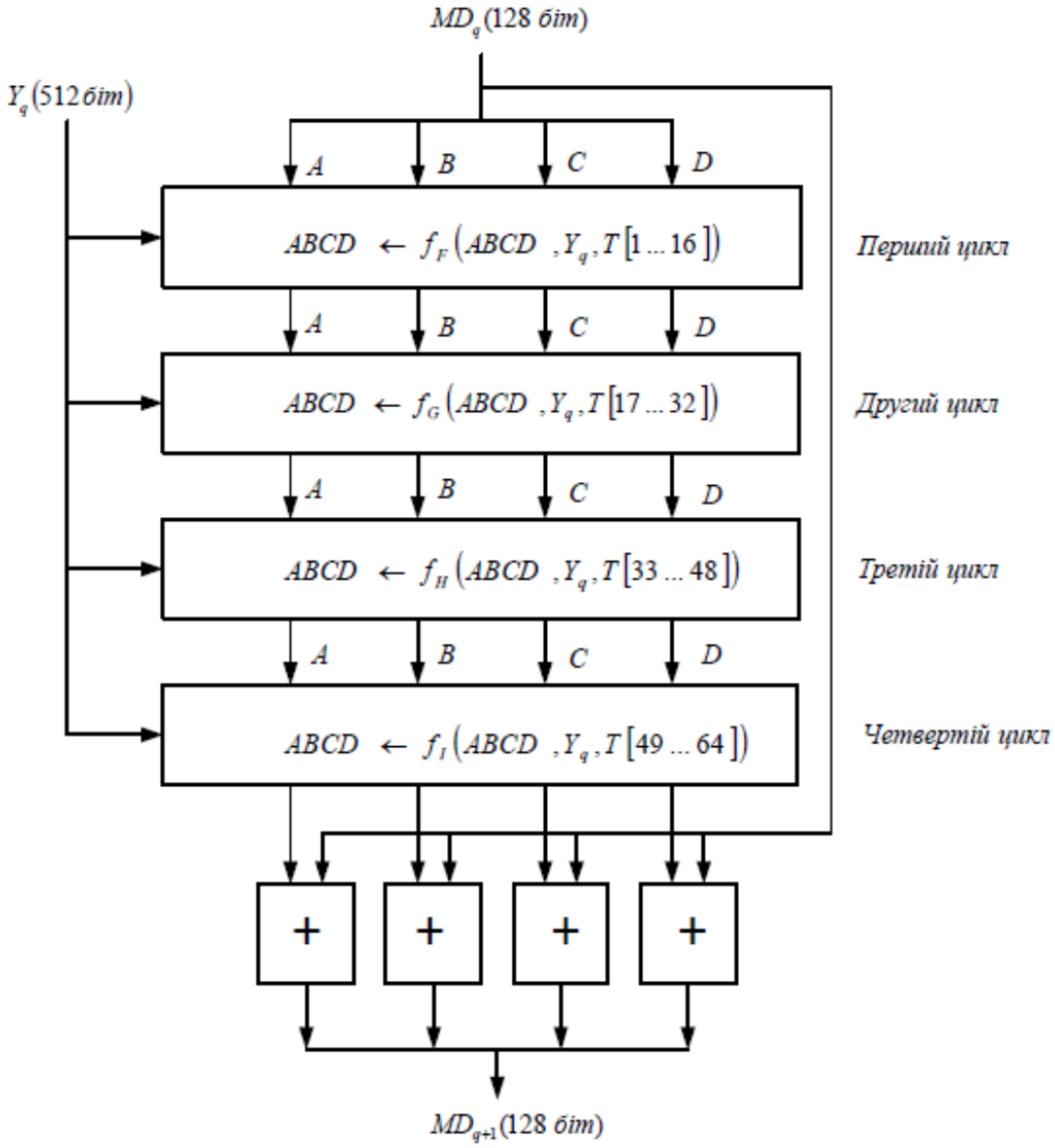


Рисунок 3.8

Кожен з циклів застосовує 4-ту частину з 64-елементного масиву

$$\mathrm{T}[1, \dots, 64], \text{де } T[i] = int(2^{32} \times |sini|)$$

Варто звернути увагу на те, що кожен з елементів масиву є цілочисельним значенням, яке може бути зображене 32 бітами. Для того аби отримати MD_{q+1}, за модулем 2^{32} з MD_q додається вихід з чотирьох циклів.

Крок 5: Завершення циклу – вихід. Як результат ми отримуємо 128 – розрядний, унікальний хеш-код який називається дайджестом повідомлення. Данний результат ми отримали після того як обробили всі L, 512 – розрядних блоків. Цикл для опрацювання 512 – розрядного блока:

$$A \leftarrow B + CLS_s(A + f(B, C, D) + X[k] + T[i]$$

Даний цикл складається з 16-ти кроків. Схема 2 (Рисунок 3.9):

$f -$ одна з f_F, f_G, f_H, f_I функцій;

$CLS_s -$ функція яка означає, зміщення на s розрядів 32-розрядного аргумента вліво циклічно;

У q-тому блоці, що містить 512 біт, k-те слово позначається як $X[k]$;

i- те слово масиву T позначається як $T[i]$;

додавання по модулю 2^{32} позначено "+".

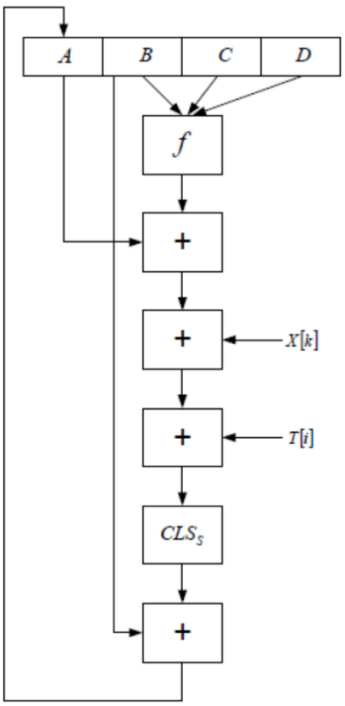
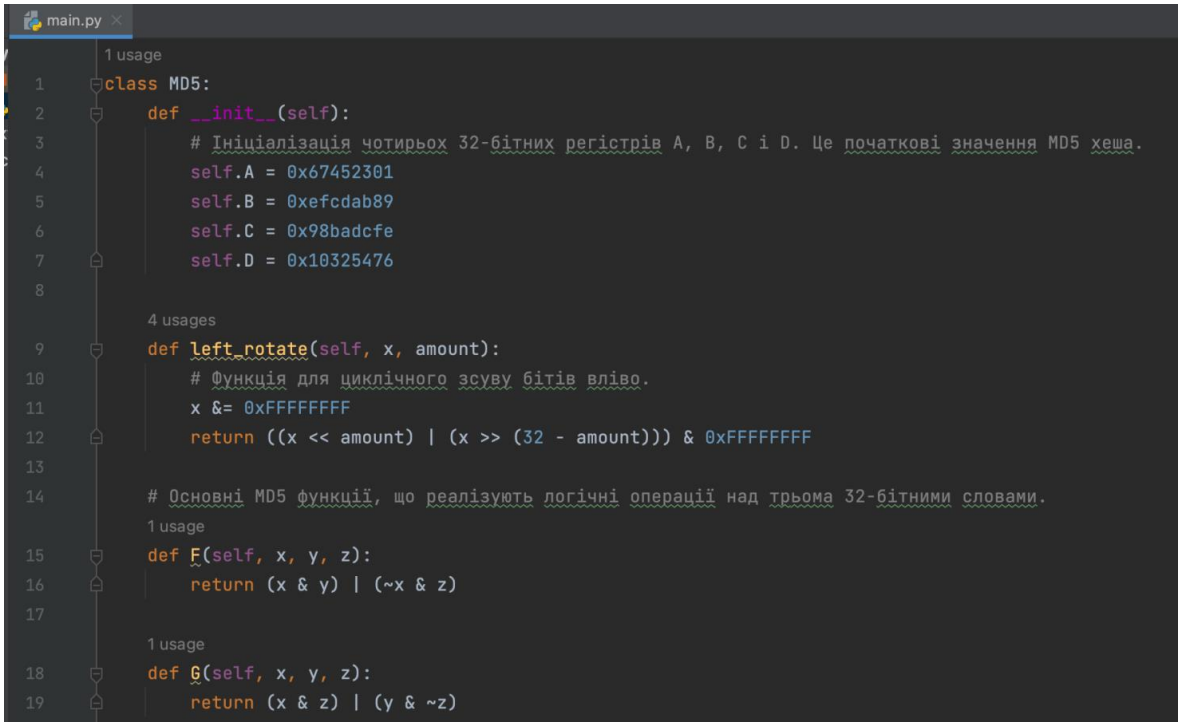


Рисунок 3.9

Кожен з циклів виконується по 16 разів. В чотирьох циклах опрацьовується кожен з блоків вхідного повідомлення. Це означає, що кожен з блоків вхідного повідомлення проходить опрацювання 64 рази за схемою 2. З кожним кроком змінюється одне з чотирьох слів буфера $\Longrightarrow$ кожне зі слів буфера отримує зміни 16 разів. Розглянемо приклад реалізації алгоритму хеш-функції MD5 на мові програмування Python:

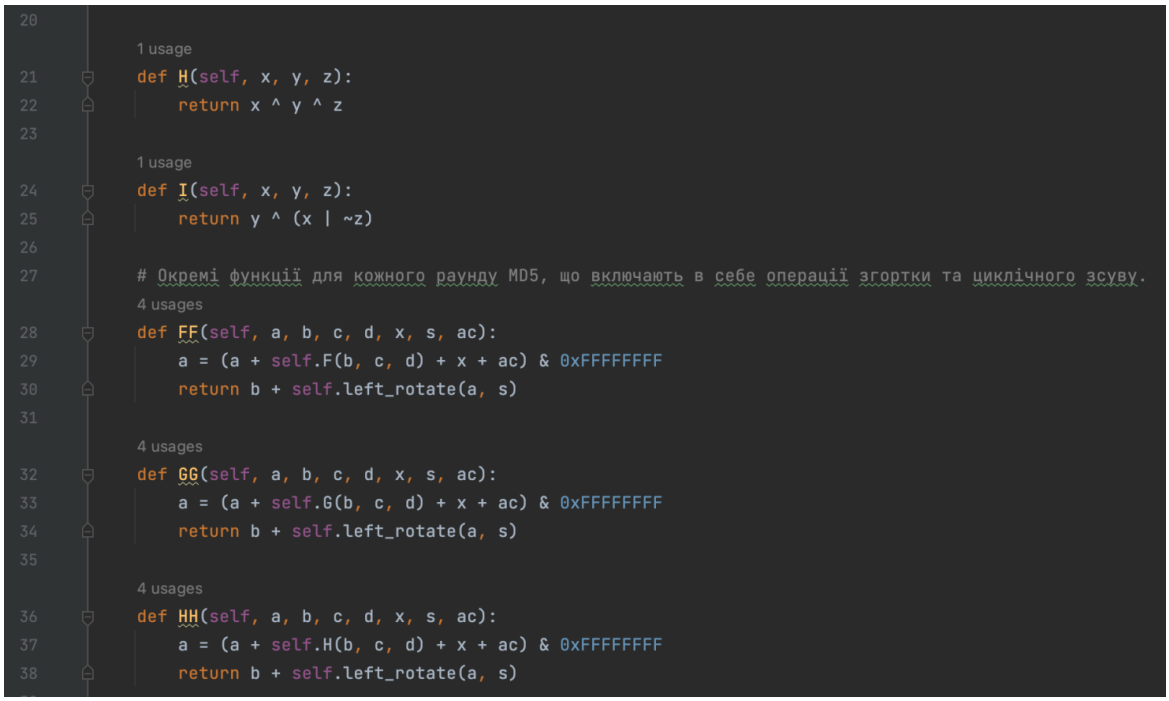


```
main.py
1 usage
class MD5:
    def __init__(self):
        # Ініціалізація чотирьох 32-бітних регістрів A, B, C і D. Це початкові значення MD5 хеша.
        self.A = 0x67452301
        self.B = 0xefcdab89
        self.C = 0x98badcfe
        self.D = 0x10325476

    4 usages
    def left_rotate(self, x, amount):
        # Функція для циклічного зсуву бітів вліво.
        x &= 0xFFFFFFFF
        return ((x << amount) | (x >> (32 - amount))) & 0xFFFFFFFF

    # Основні MD5 функції, що реалізують логічні операції над трьома 32-бітними словами.
    1 usage
    def F(self, x, y, z):
        return (x & y) | (~x & z)

    1 usage
    def G(self, x, y, z):
        return (x & z) | (y & ~z)
```



```
    1 usage
    def H(self, x, y, z):
        return x ^ y ^ z

    1 usage
    def I(self, x, y, z):
        return y ^ (x | ~z)

    # Окремі функції для кожного раунду MD5, що включають в себе операції згортки та циклічного зсуву.
    4 usages
    def FF(self, a, b, c, d, x, s, ac):
        a = (a + self.F(b, c, d) + x + ac) & 0xFFFFFFFF
        return b + self.left_rotate(a, s)

    4 usages
    def GG(self, a, b, c, d, x, s, ac):
        a = (a + self.G(b, c, d) + x + ac) & 0xFFFFFFFF
        return b + self.left_rotate(a, s)

    4 usages
    def HH(self, a, b, c, d, x, s, ac):
        a = (a + self.H(b, c, d) + x + ac) & 0xFFFFFFFF
        return b + self.left_rotate(a, s)
```

```
    4 usages
    def II(self, a, b, c, d, x, s, ac):
        a = (a + self.I(b, c, d) + x + ac) & 0xFFFFFFFF
        return b + self.left_rotate(a, s)

    1 usage
    def calculate_md5(self, input_str):
        # Перетворення вхідного рядка у байт-масив для обробки.
        message = bytearray(input_str, 'utf-8')

        # Зберігання довжини повідомлення у бітах.
        orig_len_in_bits = (8 * len(message)) & 0xffffffffffffffff

        # Додавання біта '1' до кінця повідомлення.
        message.append(0x80)

        # Доповнення повідомлення нулями до досягнення 448 біт (56 байт) в останньому 512-бітному блоку.
        while len(message) % 64 != 56:
            message.append(0)

        # Додавання довжини оригінального повідомлення в бітах до кінця повідомлення.
        message += orig_len_in_bits.to_bytes(8, byteorder='little')

        # Обробка повідомлення у блоках по 512 біт.
        for chunk in range(0, len(message), 64):
            a, b, c, d = self.A, self.B, self.C, self.D
```

```
            chunk_data = message[chunk:chunk + 64]
            X = [int.from_bytes(chunk_data[i:i + 4], byteorder='little') for i in range(0, 64, 4)]

            # Основний цикл алгоритму, що включає 64 ітерації.
            for i in range(64):
                if 0 <= i <= 15:
                    # 1-й раунд
                    a = self.FF(a, b, c, d, X[i], 7, 0xd76aa478)
                    d = self.FF(d, a, b, c, X[(i + 1) % 16], 12, 0xe8c7b756)
                    c = self.FF(c, d, a, b, X[(i + 2) % 16], 17, 0x242070db)
                    b = self.FF(b, c, d, a, X[(i + 3) % 16], 22, 0xc1bdceee)
                elif 16 <= i <= 31:
                    # 2-й раунд
                    a = self.GG(a, b, c, d, X[(5 * i + 1) % 16], 5, 0xf57c0faf)
                    d = self.GG(d, a, b, c, X[(5 * i + 6) % 16], 9, 0x4787c62a)
                    c = self.GG(c, d, a, b, X[(5 * i + 11) % 16], 14, 0xa8304613)
                    b = self.GG(b, c, d, a, X[(5 * i + 0) % 16], 20, 0xfd469501)
                elif 32 <= i <= 47:
                    # 3-й раунд
                    a = self.HH(a, b, c, d, X[(3 * i + 5) % 16], 4, 0x698098d8)
                    d = self.HH(d, a, b, c, X[(3 * i + 8) % 16], 11, 0x8b44f7af)
                    c = self.HH(c, d, a, b, X[(3 * i + 11) % 16], 16, 0xffff5bb1)
                    b = self.HH(b, c, d, a, X[(3 * i + 14) % 16], 23, 0x895cd7be)
                elif 48 <= i <= 63:
```

```
            elif 48 <= i <= 63:
                # 4-й раунд
                a = self.II(a, b, c, d, X[(7 * i) % 16], 6, 0xfd987193)
                d = self.II(d, a, b, c, X[(7 * i + 7) % 16], 10, 0xa679438e)
                c = self.II(c, d, a, b, X[(7 * i + 14) % 16], 15, 0x49b40821)
                b = self.II(b, c, d, a, X[(7 * i + 21) % 16], 21, 0xf61e2562)

        # Додаємо отримані значення до поточного стану хешу.
        self.A = (self.A + a) & 0xFFFFFFFF
        self.B = (self.B + b) & 0xFFFFFFFF
        self.C = (self.C + c) & 0xFFFFFFFF
        self.D = (self.D + d) & 0xFFFFFFFF

    # Повертаємо кінцевий хеш у вигляді шістнадцяткового рядка.
    return ''.join([format(i, '02x') for i in [self.A, self.B, self.C, self.D]])


# Виклик функції хешування
md5 = MD5()
print(md5.calculate_md5("MD5 hash-function example"))
```

Результат:

```
main
/Users/bogdanlozovoj/PycharmProjects/pythonProject5/venv/bin/python /Users/bogdanlozovoj/Pyc
f30fab904a831417368a2ab045de7d53

Process finished with exit code 0
```

3.8.2 Властивості криптографічних хеш-функцій

Очікується, що криптографічна хеш-функція повинна мати наступні властивості, які гарантують її ефективність та безпеку:

- Одностороння функція (стійкість до попередніх зображень) - ця властивість вимагає, щоб для хеш-функції H, якщо задано будь-яке значення хешу h, було обчислювально неможливо знайти вхідні дані m такі, що $H(m) = h$. Іншими словами, хеш-функцію має бути легко обчислити для кожного входу, але надзвичайно важко інвертувати за зображенням випадкового входу.

- Стійкість до колізій - ця властивість вимагає, щоб для заданої хеш-функції H було обчислювально неможливо знайти два входи m і m', такі, що $m \neq m'$ та $H(m) = H(m')$.
- Через фіксований розмір хеш-значень у порівнянні з набагато більшим і довільним розміром вхідних даних, очікується, що в хеш-функціях існують колізії. Однак, вони повинні бути обчислювально-нерозв'язними для пошуку.
- Детермінованість - ця властивість вимагає, щоб хеш-функція H послідовно відображала задані вхідні дані m у хеш-значення h. Вона також повинна бути загальнодоступною і обчислюваною.
- Лавинний ефект - ця властивість вимагає, щоб зміна лише одного біта вхідних даних призводила до значних змін у вихідних даних. Така "дифузія" гарантує, що будь-який висновок про вхідні дані на основі вихідних неможливий, тому цю властивість також іноді визначають як некерованість.

- Швидкість хешування - це ідеальна властивість криптографічної хеш-функції, яка відзначається її можливістю працювати з відповідною швидкістю. У багатьох випадках, хеш-алгоритм повинен обчислювати хеш-значення досить оперативно. Але важливо відзначити, що надто велика швидкість не завжди означає кращу або безпечнішу роботу.

3.8.3 Атаки на криптографічні хеш-функції. Колізії

Атака на криптографічну хеш-функцію передбачає порушення однієї з її властивостей безпеки. Наприклад, порушення стійкості до попереднього зображення означає, що зловмисник може створити повідомлення, яке хешується до певного хешу. Атаки на хеш-функції можуть бути зосереджені або на структурі хеш-функції, або на алгоритмі функції стиснення, яка використовується для стиснення вхідних даних довільного розміру в хеш-значення фіксованого розміру. Існують пари "ключ - значення", що дають

однакові хеш-коди. У цьому випадку виникає колізія. Імовірність виникнення колізій важлива під час оцінювання якості хеш-функцій. Колізії ускладнюють використання хеш-таблиць, оскільки порушують однозначну відповідність між хеш-кодами і даними. Проте існують способи подолання складнощів, що виникають:

- метод ланцюжків - зовнішнє або відкрите хешування;
- метод відкритої адресації - закрите хешування.

Відкрите (зовнішнє) хешування:

1. потенційно нескінчена множина розділяється на обмежену кількість класів;

2. для m класів, позначених від 0 до $m-1$, будується хеш-функція $h(x): x \to \{0, \dots, m-1\}$, де x - довільний елемент вихідної множини.

Класи можуть називати сегментами. Кажуть, що x належить сегменту $h(x)$. Масив, який називають таблицею сегментів, включають заголовки для m списків. Якщо сегменти однакові за розміром, то середня довжина списків буде $\frac{n}{m}$.

3.8.4 Метод ланцюгів

Метод об'єднання елементів передбачає, що елементи множини, які мають однакове хеш-значення, створюють ланцюжок або список:

1. в позиції номер i знаходиться вказівник на початок списку елементів, де хеш-значення ключа відповідає значенню i;
2. якщо у множині відсутні елементи, які мають хеш-значення ключа, рівне i, то в позиції і буде збережено значення NULL.

Приклад реалізації методу ланцюжків для вирішення колізій включає в себе ситуацію, коли для ключа 002 (зображено на Рисунку 3.10) існують два значення, які організовані у вигляді лінійного списку. Кожна комірка в масиві є посиланням на зв'язаний список (ланцюжок) пар ключ-значення, який відповідає одному і тому ж хеш-значенню ключа. Колізії призводять до створення ланцюжків, довжина яких може бути більше одного елемента.

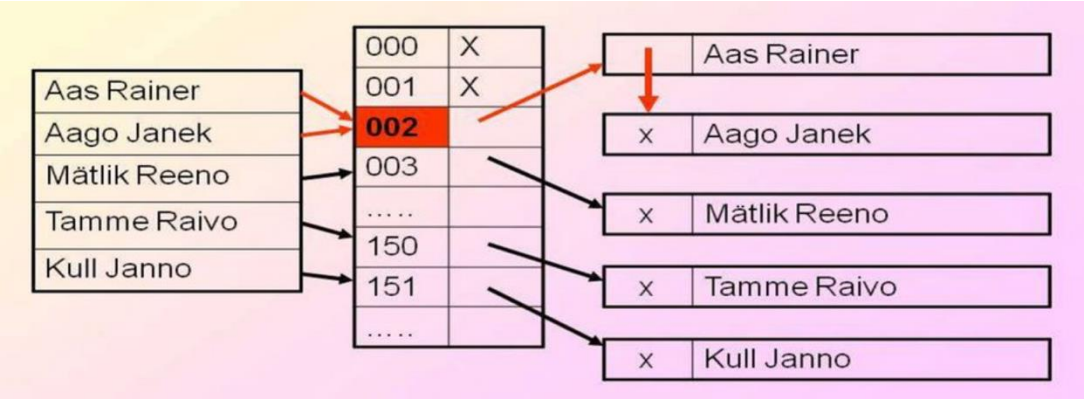


Рисунок 3.10

Для здійснення операцій пошуку або видалення даних потрібно переглянути всі елементи у відповідному ланцюжку, щоб знайти той, який має вказаний ключ (див. Рисунок 3.11).

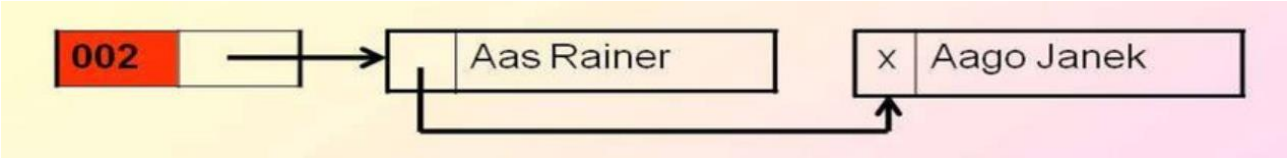


Рисунок 3.11

Для додавання даних потрібно вставити елемент у кінець або на початок відповідного списку. Якщо коефіцієнт заповнення стає надто великим, то розмір масиву збільшується і таблиця перебудовується (див. Рисунок 3.12).

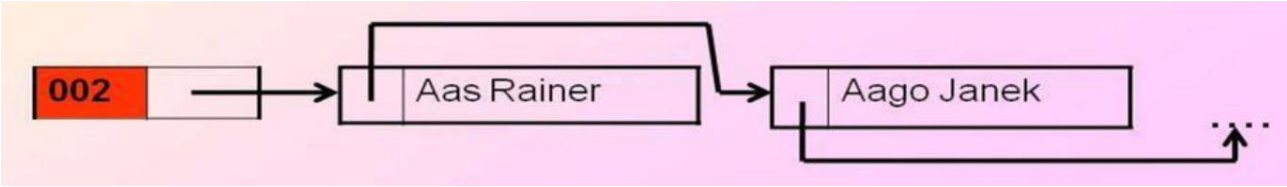


Рисунок 3.12

За умови, що кожен елемент має однакову ймовірність потрапити в будь-яку позицію таблиці і ця подія незалежна від того, куди розташовано інші елементи, середній час виконання операції пошуку елемента дорівнює $O(1+k)$. Тут k – це коефіцієнт заповнення таблиці, який визначається як $k = \frac{n}{m}$, де $n -$ це кількість елементів у таблиці, а $m -$ її розмір

3.8.5 Атака Birthday

Атака Birthday - це криптографічна атака, яка використовує математику, що лежить в основі задачі про день народження, для злому зашифрованого тексту. Кіберзлочинці використовують атаки дня народження, щоб обманювати системи, зламуючи методи цифрової автентифікації. Проблема дня народження - це задача з теорії ймовірностей. Вона запитує про ймовірність того, що принаймні двоє людей матимуть спільний день народження у випадковій групі людей. Вона також широко відома як парадокс дня народження. Цей вид атаки використовує метод грубої сили (Brute force attack) та експоненціально зростаючу ймовірність зіткнення. Теоретична можливість уникнути колізій хеш-значень вимагала б нескінченної кількості можливих результатів. Однак є два важливих обмеження на те, що можуть видавати хеш-функції. [15]

- Фіксована довжина. Шифрований текст не може тривати вічно. Він повинен мати кінцеву довжину. Більшість хеш-функцій мають одну фіксовану довжину для всіх результатів. Навіть змінна довжина лише збільшить набір можливих результатів, але не зробить його нескінченним.

- Обмежений набір символів. Деякі шифрування використовують цифри від 0 до 9, інші - літери алфавіту. Деякі можуть комбінувати обидва варіанти і додавати більше символів. Таким чином, будь-яка хеш-функція може створити лише обмежену кількість різних зашифрованих текстів. Так само, як кожна людина матиме день народження з множини можливих днів народження.

На відміну від виходів, входи можуть бути будь-якої довжини. Іншими словами, можна зашифрувати дуже короткі або дуже довгі повідомлення, або щось середнє між ними. Це робить множину можливих входів більшою за множину можливих виходів. Таким чином, деякі вхідні дані неминуче призведуть до однакових хеш-значень. Для того, щоб захистити себе і свої дані від Birthday атак, використовуються наступні практики:

- Методи шифрування з більшою кількістю потенційних хеш-значень. Пошук колізії в такому наборі потребує або більшого часу, або більшої обчислювальної потужності, ніж є у хакерів.
- Використання двофакторну автентифікацію для всіх цифрових підписів.
- Шифрування повідомлення за допомогою декількох хеш-функцій. Це значно зменшує ймовірність злому повідомлення, оскільки хакеру потрібно буде знайти безліч непов'язаних між собою колізій.

Атаки на хеш-функції можна розділити на дві великі категорії: атаки грубої сили та криптоаналітичні атаки.

3.8.6 Атаки грубої сили

Атака грубої сили, також відома як вичерпний пошук - це підхід, заснований на методі проб і помилок, в якому зловмисник використовує набір заздалегідь визначених вхідних даних (здогадок) проти алгоритму, одночасно аналізуючи вихідні дані на предмет можливого збігу. Це еквівалентно перебору всіх ключів на кільці для ключів і, врешті-решт, знаходженню правильного ключа. Атаки

грубої сили працюють на всіх хеш-функціях, незалежно від їх структури або будь-яких інших робочих деталей. Міцність і безпека хеш-функції і складність атаки грубої сили залежить виключно від розміру її вихідного хеш-значення. Для вихідного хешу довжини n, зусилля, необхідні для протистояння різним класичним атакам грубої сили, можна виразити наступним чином:

Атака односторонньої інверсії функції (стійкість до попереднього зображення) - зусилля, необхідні для знаходження вхідного значення m, яке відображається в h за допомогою H, за умови, що виклик h дорівнює $2n$, оскільки для заданого n -бітового хешу h хеш-функції H зловмисник буде обчислювати H з кожним правдоподібним входом m, поки не буде отримано бажане значення хешу виходу h.

2-га атака з попереднім відображенням - зусилля, необхідні для пошуку двох входів m і m', які відображаються в один і той самий вихід за допомогою H, дорівнює $2n$. У цій варіації атаки грубої сили зловмисник обчислює хеш-функцію H з кожним можливим входом $m' \neq m$, для заданого входу m, доки не буде отримано значення хешу $h = H(m)$

Атака на зіткнення - зусилля, необхідні для заданої хеш-функції H, щоб знайти два входи m і m', такі, що $m \neq m'$ і $H(m) = H(m')$ дорівнює $2^{\frac{n}{2}}$, оскільки в середньому зловмисник повинен був би спробувати 2^{n-1}(тобто $\frac{2^n}{2}$) входів, щоб знайти один, хеш-значення якого збігається. Однак, в атаці на день народження, яку ми розглянули раніше і яка базується на парадоксі дня народження, можлива атака на обраний відкритий текст. В цьому випадку зусилля, необхідні для зіткнення в хеш-функції, дорівнюють $2^{\frac{n}{2}}$ на противагу 2^{n-1}.

Подальші розширення цих класичних атак перебором вивчалися різними авторами. До них відносяться: K-стороння атака зіткнення для $K \geq 2$, метою

якої є знайти K різних входів m і , таких що $H(m^i) = \cdots = H(m^k)$ та K - стороння атака 2-го попереднього зображення для $K \geq 1$, де для входу m , хеш-значення h та хеш-функції H, такої що $h = H(m)$, метою є знайти K різних входів m і з $H(m^i) = h$ та $m^i \neq m$.

3.8.7 Криптоаналітичні атаки

Криптоаналіз - це дослідження зашифрованого тексту, шифрів і криптосистем з метою виявлення слабких місць або витоків прихованих аспектів криптосистем, які можуть бути корисними для отримання значення зашифрованої інформації без доступу до секретного ключа, який зазвичай необхідний для цього. Криптоаналіз хеш-функцій фокусується на їхній базовій структурі та/або алгоритмі функції стиснення. Ця ітеративна структура була незалежно запропонована Іваном Дамгордом та Ральфом Меркле на конференції Crypto '89. Вона називається конструкцією Меркла-Дамгорда і застосовується в конструкції більшості хеш-функцій, що використовуються сьогодні.

Хеш-функція приймає вхідне повідомлення і розбиває його на L блоків фіксованого розміру по b біт кожен. Останній блок може бути збільшений до b біт, якщо необхідно, і може також включати значення загальної довжини вхідного повідомлення, що ускладнює роботу зловмисника. Зловмисник повинен або знайти два повідомлення однакової довжини, які хешуються до однакового значення, або два повідомлення різної довжини, які з додаванням довжини кожного з них хешуються до однакового значення. Якщо існує стійка до вхідних колізій функція стиснення $f: \{0,1\}^b$ x $\{0,1\}^t$ фіксованої довжини, то можна створити стійку до вхідних колізій хеш-функцію $H: \{0,1\}^* \rightarrow \{0,1\}^t$ шляхом ітерацій над цією функцією стиснення. Іншими словами, якщо функція стиснення є стійкою до колізій, то такою є і результуюча ітераційна хеш-функція. Таким чином, якщо функція стиснення вразлива до будь-якої атаки, то вразливою є і ітераційна хеш-функція, але зворотне твердження не обов'язково

вірне в загальному випадку. Криптоаналітичні алгоритми пошуку колізій та атаки можуть бути класифіковані як одно- або багатоблокові в залежності від того, чи використовує атака одну функцію стиснення (один блок) або більше однієї ітерації функції стиснення (більше одного блоку) для пошуку колізій або попередніх зображень.

Криптоаналітичні атаки на хеш-функції, так само як і на алгоритми шифрування, намагаються використати певну властивість алгоритму для здійснення атаки, відмінної від вичерпного перебору. Однак хеш-функції практично легше атакувати, ніж алгоритми шифрування, оскільки зловмисникові не потрібно припускати наявність будь-яких секретів, а максимальні обчислювальні зусилля, необхідні для атаки на хеш-функцію, обмежуються лише ресурсами зловмисника, а не довірливістю користувача. Це не так у випадку з блоковими шифрами, де максимальна практична кількість виконань блокового алгоритму обмежена тим, скільки обчислювальних зусиль зловмисник може змусити користувача виконати. Міра стійкості хеш-алгоритму до криптоаналізу ґрунтується на порівнянні його стійкості з рівнем зусиль, необхідних для атаки грубої сили. Тобто, ідеальний хеш-алгоритм потребує криптоаналітичних зусиль, що перевищують або дорівнюють зусиллям, необхідним для перебору грубої сили.

Метод "Зустрічі посередині" використовує відомий відкритий текст x та відповідний криптотекст y за такими кроками:

1. Для тексту x формується впорядкована база даних за криптограмами, що містить випадкову множину ключів k' і відповідні криптограми $w = F(k', x)$. Обсяг цієї бази даних $O(\sqrt{|\{k'\}|})$, де $\left|\left\{k^{'}\right\}\right| -$ є потужністю множини ключів k'.

2. Випадковим чином обираються ключі k'', що застосовуються для розшифрування текстів y, і отримані результати розшифрування $v = F(k'', y)$

порівнюються з базою даних. Якщо текст v співпадає з однією з криптограм w, то ключ $k'k''$ вважається еквівалентним шуканому ключу k.

3.9 Рандомізація в хеш функціях

Хешування ефективно працює, коли для всіх різних елементів u та v з множини S виконується умова $h(u) \neq h(v)$. У такому випадку перевірка наявності елемента u здійснюється за рахунок перевірки позиції масиві $H[h(u)]$, яка або порожня, або містить лише u.

Однак, як ми вже знаємо для елементів u та v які належать S можуть виникати колізії ($h(u) = h(v)$).

Існують різні методи вирішення проблеми колізій у хеш-таблицях. Розглянемо випадок, коли в кожній позиції $H[i]$ хеш-таблиці зберігається зв'язний список всіх елементів $u \in S$ з $h(u) = i$. Операція пошуку елемента $Lookup(u)$ виконується за наступним алгоритмом:

- обчислити хеш-функцію $h(u)$;
- перевірити зв'язний список на позиції $H[h(u)]$ і встановити, чи присутній u в списку.

Звідси випливає, що час, необхідний для $Lookup(u)$, пропорційний сумі часу обчислення $h(u)$ та довжини зв'язного списку в позиції $H[h(u)]$. Остання величина відображає кількість елементів в S, які знаходяться в колізії з u. Операції вставки $Insert$ та видалення $Delete$ працюють аналогічно: $Insert$ додає u до зв'язного списку на позиції $H[h(u)$, а $Delete$ переглядає цей список і видаляє елемент u, якщо він присутній.

Мета полягає в тому, щоб знайти хеш-функцію, яка рівномірно розподіляє додані елементи, щоб жодна позиція в хеш-таблиці H не містила занадто багато елементів. Припустимо, що $|U| \geq n^2$, тоді для будь-якої обраної хеш-функції h існує певна множина S з n елементів, які відображаються на одну позицію. У найгіршому випадку всі елементи множини будуть вставлені, і тоді у операції $Lookup$ буде задіяно перебір зв'язного списку довжиною n.

Рандомізація може надати суттєву допомогу у таких задачах. Ми не робими припущень щодо випадковості множини елементів S, але застосовуємо рандомізацію при плануванні хеш-функції. Колізії не усуваються повністю, але стають відносно рідкісними, так що списки виходять достатньо короткими.

3.9.1 Вибір ефективної хеш-функції

Розглядаємо U як велику множину чисел, до яких застосовуємо легко обчислювану функцію h, яка відображає кожне число $u \in U$ у значення з меншого цілочисельного діапазоні $\{0,1,\dots,n-1\}$. Прості методи, такі як використання кількох перших або останніх цифр u або обчислення залишку від ділення u на n, можуть бути ефективними, але також можуть призвести до численних колізій.

На практиці краще використовувати $u\ (mod\ p)$ для простого числа p, близького до n. Хоча цей метод може породжувати ефективні хеш-функції, він не є універсальним, а деякі прості числа працюють краще за інших.

Розглянемо емпірично ефективні хеш-функції. Основна ідея полягає у використанні рандомізації при побудові h. Наприклад, для кожного елемента $u \in U$ при вставці у S значення $h(u)$ обирається випадково із рівномірним розподілом із множини $\{0,1,\dots,n-1\}$.

У схемі випадкового рівномірного хешування ймовірність того, що два випадково обрані значення $h(u)$ та $h(v)$ викликають колізію дорівнює $\frac{1}{n}$. Доведення:

Уявімо, що ми маємо n можливих значень для $h(u)$ та $h(v)$. Це означає, що існує n^2 можливих пар значень $h(u), h(v)$.

Оскільки в рандомізованому хешуванні кожне значення вибирається незалежно та рівномірно, кожна з цих n^2 пар має однакову ймовірність виникнення. Тепер, щоб визначити ймовірність колізії, потрібно врахувати лише ті пари, де $h(u) = h(v)$. Таких пар рівно n, оскільки для кожного можливого значення хеш-функції (скажімо, i), існує рівно одна пара (i,i), де $h(u) = h(v) = i$. Отже, з усіх n^2 пар, лише n з них викликають колізію.

■

Ймовірність колізії між двома випадково вибраними значеннями $h(u)$ та $h(v)$ досить мала, проте використання хеш-функції з незалежними випадковими рівномірними значеннями є непрактичним, оскільки потрібно зберігати значення $h(u)$ для операцій видалення або пошуку.

3.9.2 Універсальні класи хеш-функцій

Універсальні класи хеш-функцій передбачають вибір хеш-функції не з усіх можливих, а зі спеціального сімейства. Функції у цьому класі відображають універсальну множину U на множину $\{0,1,\dots,n-1\}$ і мають дві характеристики:

- Ймовірність того, що для будь-якої пари елементів $u, v \in U$ і випадково обраної функції $h \in H$ виконується рівність $h(u) = h(v)$, не перевищує $\frac{1}{n}$.
- Кожна функція $h \in H$ має компактне представлення, і для будь-яких h та $u \in U$ значення $h(u)$ може бути ефективно обчислено.

Такі класи H можуть володіти обома цими властивостями. Основна властивість універсального класу хеш-функцій полягає у тому, що якщо функція h вибрана випадково з універсального класу, то для будь-якого множини $S \subseteq U$ розмір якої не перевищує n, та будь-якого елемента $u \in U$, очікувана кількість елементів S, що створюють колізію з u, є сталою.

Нехай $H-$ це універсальний клас хеш-функцій, які відображають універсальну множину U на множину $\{0,1,\dots,n-1\}$, $S-$ довільна підмножина U розміром не більше n, а $u-$ будь-який елемент U. Визначимо X як випадкову змінну, рівну кількості елементів $s \in S$, для яких $h(s) = h(u)$ для випадково обраної хеш-функції $h \in H$. (S та u фіксовані, а випадковість проявляється у виборі $h \in H$). Тоді $E[X] \leq 1$.

Доведення: Для елемента $s \in S$ визначається випадкова змінна X_s, яка дорівнює 1, якщо $h(s) = h(u)$, або 0 в іншому випадку, оскільки клас функцій універсальний.

$X = \sum_{s \in S} X$, внаслідок лінійного очікування маємо $E[X] = \sum_{s \in S} E[X_s] \leq |S| \times \frac{1}{n} \leq 1$ ■

3.9.3 Використання кількох незалежних хеш-функцій

Цей підхід полягає у використанні декількох незалежних хеш-функцій $h_1, h_2, \ldots, h_k$, кожна з яких відображає вхідні дані в хеш-таблицю. Це знижує ймовірність колізії, оскільки елемент потрібно вставити або знайти в кожній з цих таблиць.

Якщо ймовірність колізії для однієї хеш-функції становить p, то при використанні k незалежних функцій ймовірність того, що всі функції призведуть до колізії, значно знижується і становить приблизно p^k.

Опишемо алгоритм, який обирає одну з хеш-функцій для кожного елемента або різні функції для різних операцій:

1. Вибір функції на основі ключа:

- Нехай $P = \{h_1, h_2, \ldots, h_k\} -$ множина хеш-функцій.
- Функція вибору $F: Key \longrightarrow \{1,2, \ldots, k\}$, де $Key -$ множина можливих ключів.
- Для ключа key, хеш-функція, що використовується, буде $h_{F(key)}$.

2. Вибір функції на основі операції:

- Визначимо множину операцій $O = \{o_1, o_2, \ldots, o_m\}$.
- Асоціюємо кожну операцію з певною хеш-функцією через функцію $G: O \to \{1,2, \ldots, k\}$.
- Для операції o, використаємо хеш-функцію $h_{G(o)}$.

Використання різних хеш-функцій для різних елементів або операцій у контексті рандомізованих алгоритмів для боротьби з колізіями в хеш-функціях зменшують ймовірність колізій, розподіляють навантаження та дають можливість адаптувати вибір хеш-функції до конкретних умов даних. [23]

3.9.4 Метод "солі" (Salt)

Цей метод застосовується у боротьбі з колізіями в хеш-функціях. У математичному контексті рандомізації, цей метод можна виразити функцією

$h(x, s)$, де $h-$ хеш-функція, $x-$ вхідні дані, а $s-$випадково генероване значення (сіль). Використання різних значень s для кожного набору вхідних даних x забезпечує унікальність хеш-значень, навіть для ідентичних x:
$h(x_1, s_1) \neq h(x_2, s_2)$, для $x_1 = x_2$ та $s_1 \neq s_2$.

Метод зменшує ймовірність колізій, оскільки кожен набір вхідних даних отримує своє унікальне випадкове значення.

3.9.5 Рандомізовані алгоритми у захисті криптовалют

Рандомізація підвищує безпеку шляхом ускладнення передбачення або відтворення криптографічних ключів. Використання сильного генератора випадкових чисел (PRNG) можна математично описати через його ентропію H, де $H = -\sum p(x)\ log\ log\ p(x)$, $i\ p(x)$ ймовірність появи випадкового числа x.

Приклади використання в криптовалютах

- Bitcoin:

В Bitcoin, рандомізація використовується при створенні приватних ключів. Нехай Pє приватним ключем, тоді $P = f(R)$, де $R-$ випадкове число.

- Ethereum:

Ethereum впроваджує рандомізацію у своїх Proof of Stake протоколах, де вибір валідаторів залежить від випадкової величини.

- Ripple:

Ripple використовує рандомізацію в процесі валідації транзакцій, де кожен валідатор випадково вибирає підмножину інших валідаторів для перевірки транзакцій.

- Monero:

У Monero, рандомізація використовується для забезпечення анонімності транзакцій через кільцеві підписи, де вибір "співпідписників" є випадковим.

Аналіз та порівняння захищеності криптовалют вимагає розуміння криптографічних принципів та здатності кількісно оцінити різні аспекти безпеки. Використання математичних формул дозволяє об'єктивно порівнювати захищеність різних криптовалют.

Основні критерії безпеки:

- Стійкість до криптоаналітичних атак:

Математично, це можна визначити через криптографічну складність C , де

$C = 2^n$ і n - кількість бітів у хеші. Чим більше n, тим складніше взломати систему.

- Масштабування хеш-функцій:

Ефективність хеш-функції може бути оцінена через час T потрібний для обробки даних обсягом n, де $T(n)$ повинна рости лінійно або сублінійно відносно n.

- Здатність системи протистояти спробам зламу:

Можна оцінити через ймовірність успішної атаки P, де $P = 1 - (1 - p)^k$,

$p -$ ймовірність успішної атаки на один вузол, а k - кількість вузлів, які зловмисник може атакувати.

Методологія порівняння:

- Аналіз стійкості хеш-функцій: Для оцінки стійкості до колізій можна використовувати ймовірність знаходження двох різних вхідних даних, що

дають однаковий хеш. Для ідеальної хеш-функції з n-бітовим виходом, ця ймовірність дорівнює $1/2^n$.

- Оцінка протоколів консенсусу: Можна розглянути ймовірність успішної атаки на протокол. Наприклад, для доказу роботи (Proof of Work), $P = \left(\frac{q}{p}\right)^z$, де q - частка хешрейту зловмисника, p - загальний хешрейт мережі, і z - кількість підтверджень. Хешрейт — це міра обчислювальної потужності, яка використовується при майнінгу та обробці транзакцій у блокчейн мережах, які використовують Proof of Work. Це вимірюється у кількості хеш-операцій, які майнінгова система може виконати за секунду. Наприклад, хешрейт 5 TH/s (терахешів за секунду) означає, що система може проводити 5 трильйонів хеш-обчислень кожну секунду.
- Вивчення історії безпеки: Аналізування історичних даних про вразливості та атаки, використовуючи статистичні методи для оцінки частоти та серйозності інцидентів.

Цей аналіз дозволяє не тільки оцінити поточний стан безпеки різних криптовалют, але й розробити рекомендації щодо покращення захищеності та вибору найбільш безпечних систем.

РОЗДІЛ 4. Математичне представлення криптовалюти Bitcoin та криптовалюти Ethereum

4.1 Доказ роботи (Proof-of-Work, PoW)

Доказ роботи - це математична проблема, метою якої є створення зв'язку між двома блоками. Цей зв'язок буде матеріалізований у заголовку другого блоку. Того, хто намагається вирішити завдання доказу роботи, називають майнером.

Розгляньмо два блоки, позначені як $B^{\text{попередній}}$ і B, та число, що називається bits і позначається як b. b визначає, наскільки складним є доказ

роботи: з b можна безпосередньо обчислити цільове число. Ця ціль - це 64-значне шістнадцяткове число з декількома нулями на своїх лівих цифрах, наприклад:
0000000000000000af101257cb4250ec39d3e6a584c3dc408c3fbb59139381.

Ми припускаємо, що хеш попереднього блоку відомий і це $H(B^{\text{попередній}})$. $H -$ це хеш функція $SHA256$.

Вирішення доказу роботи для блоку B, також відоме як майнінг блоку B, полягає в знаходженні числа, яке називається "$nonce$", таким чином:

$$H\left(H\left(B^{\text{попередній}}\right) \oplus R^H(B) \oplus t \oplus b \oplus nonce\right) \leq \text{ціль},$$

де $\oplus$ - це операція конкатенації; $t -$ позначає поточний час до секунд.

Оскільки H є хеш-функцією, єдина можливість знайти підходящий $nonce$ полягає у спробі шляхом жорсткого підбору, поступово збільшуючи значення $nonce$, поки t змінюється. Ми припускаємо, що в момент t^0 ми знаходимо $nonce^0$, який вирішує доказ роботи. Хеш $H(B)$ блоку B визначається так:

$$\underbrace{H\left(H\left(B^{\text{попередній}}\right) \oplus R^H(B) \oplus t^0 \oplus b \oplus nonce^0\right)}_{H(B)} \leq \text{ціль},$$

Оскільки хеш блоку визначається рекурсивно, слідуючи вищезазначеній процедурі (ми припустили, що $H(B^{\text{попередній}})$ вже відомий), нам потрібно ініціалізація: якщо B є першим блоком, то попереднього блоку немає, отже, $H(B^{\text{попередній}})$ є просто конвенцією.

4.1.1 Заголовок блоку

Коли доказ роботи стає розв'язаний для пари блоків (B^{prev}, B), ми можемо визначити заголовок блоку B, використовуючи вищезазначені позначення:

$$\text{Заголовок блоку}(B) = \left(id_m, H\left(B^{\text{попередній}}\right), R^M(B), t^0, b, \text{с}, H(B)\right),$$

де id_m позначає ідентичність суб'єкта майнингу.

Доказ роботи є ключовою поняттям у світі блокчейну. Визначення “доказ роботи” означає, що вирішення вищезгаданої математичної проблеми є способом довести, що ви приділили певний час для знаходження відповіді.

Дійсно, оскільки H є односторонньою функцією, найкращий метод знайти дійсний $nonce$ полягає у тому аби спробувати кожне можливе значення: якщо $nonce$ не підходить, то: $nonce \leftarrow nonce + 1$. "Майнер", тобто людина з одним (або багатьма) комп'ютером(ами), яка намагається вирішити доказ роботи, потребує все більш значної обчислювальної потужності, оскільки складність доказу роботи зростає з кількістю "видобутих" блоків: b регулюється алгоритмом, він залежить від кількості блоків, які вже присутні в Bitcoin.

Як тільки майнер розв'язує “доказ роботи”, будь-кому легко перевірити, що рішення правильне: все, що потрібно, - це обчислити хеш

$$H\left(H\left(B^{\text{попередній}}\right) \oplus R^H(B) \oplus t^0 \oplus b \oplus nonce^0\right)$$

з t^0 та $nonce^0$, наданими майнером.

Майнінг вимагає певної обчислювальної потужності, але за це майнер отимує винагороду. З Bitcoin майнер, який правильно розв'язав проблему доказу роботи, винагороджується біткоїнами. Винагорода зменшується з кількістю блоків у Bitcoin. Ось як в цілому система може створювати нові біткоїни.

4.1.2 Побудова блокчейну Bitcoin

Вузол є основною одиницею в розподіленій мережі. На Рисунок 4.1 кожна чорна точка представляє вузол. В реальному світі вузлом може бути людина із одним або кількома комп'ютерами.

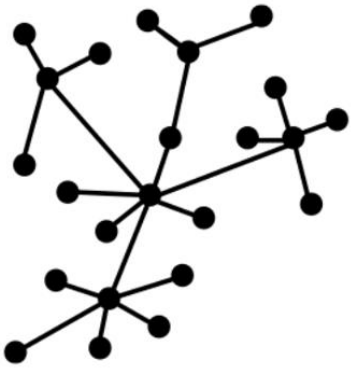


Рисунок 4.1

Існує декілька видів вузлів:

- прості вузли, чия роль полягає в передачі інформації, без зберігання локальної копії блокчейну;
- повні вузли, чия роль полягає в майнінгу нових блоків. Вони зберігають локальну копію всього блокчейну.

Кожен повний вузол зберігає свою власну версію блокчейну, але завдяки механізмам мережі, ці вузли досягають консенсусу.

Крок 1 в побудові блокчейну: новий запис у системі

Позначимо Tr_x як новий запис, наприклад, Макс відправляє 1,1 біткоїнів Марії. Він потрапляє в мережу через вузол, позначений $n_{\text{вхід}}$. Кожен вузол має свій протокол валідації, тобто програму для перевірки, чи є запис дійсним чи ні. Вузол може обрати використання стандартної програми для етапу валідації, але

це не обов'язково: з багатьох причин (експериментальних, шахрайства, місцевої необхідності тощо) вузол може вибрати індивідуальний протокол валідації.

Вузол не довіряє інформації, яку він отримує (у блокчейні немає місця для довіри), тому він проводить кілька перевірок, використовуючи свій власний протокол валідації. Протокол валідації можна розглянути як функцію:

$$PrV : \mathbb{N} \times \mathbb{E} \to \{True, False\}$$

$$(n, T_r) \mapsto P_rV(n, T_r),$$

де $\mathbb{N}$ - це множина, що містить усі вузли, а $\mathbb{E}$ - множина всіх транзакцій.

Отже, у нашому випадку, $n_{\text{вхід}}$ обчислює $P_rV(n_{\text{вхід}}, T_{r_x})$. Якщо відповідь є істинною (True), $n_{\text{вхід}}$ розглядає транзакцію як дійсну: T_{r_x} передається сусіднім вузлам. У противному випадку, $n_{\text{вхід}}$ відхиляє запис, і він не передається далі.

У наших позначеннях, якщо $n_{\text{вхід}}$ є простим вузлом. Для нашої моделі ми припускаємо, що простий вузол перевіряє лише те, що транзакція підписана її ініціатором.

Розглянемо, що буде якщо $n_{\text{вхід}}$ є повним вузлом.

Крок 2 в побудові блокчейну: запис досягає повного вузла.

Як тільки транзакція T_{r_x} входить в мережу, вона може врешті-решт досягти повного вузла, позначимо його як n_c. Подібно до простого, повний вузол спочатку обчислює $P_rV(n, T_{r_x})$. Перевірки, які проводить повний вузол, є більш широкими, ніж у простого вузла. Припустимо, що в нашій моделі повний вузол перевіряє як ідентичність ініціатора, так і відсутність подвійного витрачання.

Дійсно, повний вузол може переконатися, що немає подвійного витрачання, оскільки він зберігає свою локальну версію блокчейну: маючи

доступ до історії усіх транзакцій, вузол може дати відповідь на таке питання: чи має Макс 1.1 біткоїнів, які він може відправити?

Якщо транзакція T_{r_x} вважається дійсною, n_c додає її до свого локального списку дійсних транзакцій $L^{n_c}_{\text{локальний}}$:

$$L^{n_c}_{\text{локальний}} append\left(T_{r_x}\right),$$

а потім T_{r_x} передається сусіднім вузлам.

Коли повний вузол "майнить", він створює новий блок, додаючи до нього деякі дійсні транзакції, що присутні в його локальному списку $L^{n_c}_{\text{локальний}}$:

$$B_{n_c} = \left(T_{r^1}, \dots, T_{r_N}\right),$$

де $T_{r_i} \in L^{n_c}_{\text{локальний}}$ для $1 \leq i \leq N$. Потім n_c намагається вирішити задачу доказу роботи для $\left(B^{n_c}_{\text{останній}}, B_{n_c}\right)$, де $B^{n_c}_{\text{останній}}$ це останній блок в локальній версії блокчейну.

Повні вузли завжди змагаються один з одним: коли в їхньому локальному списку достатньо дійсних транзакцій, або після певного періоду часу, повний вузол генерує новий блок, після чого намагається розв'язати проблему доказу роботи. Як тільки це виконано, переможцем створюється новий блок разом з його заголовком.

Крок 3 в побудові блокчейну: випуск нового блоку.

Коли новий блок $B_{\text{новий}}$ та його заголовок були випущені вузлом n, це означає, що n стверджує, що він вирішив проблему доказу роботи. Новий блок передається сусіднім вузлам: оскільки у блокчейні немає місця для довіри, знову вузли проводять багато перевірок. Позначимо n_r вузлом, який щойно отримав $B_{\text{новий}}$: якщо n (повний) намагається вирішити свою власну проблему доказу роботи, він зупиняється. Якщо n (повний або простий) вже отримав $B_{\text{новий}}$, останній автоматично відхиляється. В іншому випадку він

починає з перевірки того, чи всі транзакції у $B_{\text{новий}}$ є дійсними. Якщо одна транзакція не дійсна, $B_{\text{новий}}$ відхиляється.

Потім n перевіряє, чи доказ роботи був правильно виконаний: достатньо одного обчислення хеш-функції, щоб побачити, чи вихідне значення менше цілі. Також швидко перевірити, що корінь Меркла нового блоку та його хеш-значення коректні.

Якщо n є повним вузлом, ситуація трохи складніша. Дійсно, у цьому конкретному випадку, n повинен вирішити, чи буде $B_{\text{новий}}$ додано до його локальної версії блокчейну. Блок додається, якщо хеш попереднього блоку, який з'являється у переданому заголовку $Head(B_{\text{новий}})$, дорівнює хешу останнього блоку у локальній версії блокчейну. У цьому випадку n видаляє транзакції всередині $B_{\text{новий}}$ зі свого локального списку, і майнер, який випустив новий блок, отримує винагороду : n починає створювати новий блок з дійсними транзакціями, і цей новий блок починається з конкретного рядка, який говорить, що переможець попереднього змагання (чия ідентичність зберігається у $Head(B_{\text{новий}})$) отримує винагороду деякою кількістю біткоїнів.

В іншому випадку $B_{\text{новий}}$ зберігається в n.

Ми прийшли до важкого питання, а саме конфліктів між вузлами щодо "справжньої" версії блокчейну. Що відбувається, коли деякі вузли зберігають версію B_{k_1} блокчейну, а інші - версію B_{k_2}?

Перш за все важливо зрозуміти, що така ситуація можлива. Припустимо, що в певний момент часу t всі вузли мають однакову версію B_{k_0} блокчейну. У $t_1 > t$ два різних вузли випускають два різних дійсних блоки, B_a та B_b. Така ситуація може виникнути, оскільки повний вузол вільний обирати будь-які дійсні транзакції у своєму локальному списку для генерації нового блоку.

Оскільки новому блоку потрібен час, щоб досягти кожного вузла мережі, деякі вузли отримають B_a до того як отримали B_b, а інші B_b до того як отрим B_a. Обидва ці блоки вважаються дійсними, тому в обох випадках вони будуть

додані: у $t_2 > t$ можна знайти вузли з локальною версією $B_{k_0} + B_a$ та інші з $B_{k_0} + B_b$. Це називається "розгалуженням" або "форком".

Для Bitcoin та інших криптовалют, форки можуть виникати природним чином, це може відбуватись через затримки в мережі або у випадку коли різні групи майнерів використовують різні правила для валідації блоків.

Щоб вирішити таке розгалуження, необхідні правила консенсусу: повний вузол завжди зберігатиме "найдовшу" версію серед усіх локальних, які він має. Під "найдовшою" мається на увазі версія, яка потребувала найбільшої кількості роботи, це спосіб посилити безпеку блокчейну.

Припустимо, що у $t_3 > t_2$ вузол з локальною версією $B_{k_0} + B_a$ випускає новий дійсний блок B_c, і цей блок досягає всіх вузлів раніше, ніж буде випущений інший. Вузли, які зберегли $B_{k_0} + B_a$ як локальну версію, тепер зберігатимуть $B_{k_0} + B_a + B_c$. Але вузли, які зберегли $B_{k_0} + B_b$, стикаються з дилемою.

Коли такий вузол n отримує B_c, важливо звернути увагу на те, що локальною версією є $B_{k_0} + B_b$, але n також раніше отримав B_a. Оскільки не було можливості з'єднати B_a з $B_{k_0} + B_b$, B_a зберігається. B_c не можна додати до $B_{k_0} + B_b$, але можна додати до B_a: $B_a + B_c$. Є сумнів щодо блоків після B_{k_0}: між $B_{k_0} + B_b$ та $B_{k_0} + B_a + B_c$, n зберігатиме найдовший ланцюжок, тобто $B_{k_0} + B_a + B_c$. Отже, ми досягаємо консенсусу між усіма вузлами.

4.1.3 Безпека блокчейну Bitcoin

Щоб мати загальне уявлення про Bitcoin, важливо зрозуміти, чому вищезазначені механізми створюють безпечний блокчейн. Доказ роботи відіграє вирішальну роль. Він гарантує, що неможливо змінити історію блокчейну: зміна одного блоку означає зміну його хешу та відповідно всіх хеш-значень наступних блоків. Для зміни минулого блоку потрібно було б перебудувати весь послідовний ланцюжок блоків, починаючи з блоку, який був змінений. Якщо хтось захоче обдурити загальну систему, зламавши один вузол, навіть повний, наприклад, щоб змусити його прийняти незаконну транзакцію,

це не матиме жодного ефекту. Оскільки у блокчейні немає місця для довіри, інші вузли відразу виявлять незаконну транзакцію та відкинуть її. Таким чином, єдиний спосіб змусити систему прийняти незаконну транзакцію - це взяти під контроль більшість повних вузлів. І навіть у цьому випадку незаконна транзакція була б визнана дійсною лише підконтольними порушнику вузлами. Це створило б розходження між версіями блокчейну, збереженими вузлами а отже консенсус не був би досягнутий.

Таким чином, навіть якщо комусь вдасться взяти під контроль половину повних вузлів (атака 51%), і видати “незаконну” транзакція за “законну”, враховуючи більшість вузлів, ми б спостерігали відсутність консенсусу. Звісно, ця ситуація є лише теоретичною, оскільки для того, щоб взяти під контроль половину повних вузлів, потрібні велічезні обчислювальні можлимості.

4.1.4 Хеш-функція SHA-256

Bitcoin використовує хеш-алгоритм SHA-256. SHA-256 (Secure Hash Algorithm 256) – це широко використовуваний криптографічний алгоритм, який створює хеш-значення фіксованої довжини, 256 біт (32 байти). Метою алгоритму SHA-256 є створення унікального цифрового відбитка даних, таких як повідомлення або файл.

Генерація хешу SHA-256 включає обробку вхідних даних через складну математичну функцію, яка виробляє унікальне вихідне значення. Це вихідне значення є хешем, який служить як цифровий відбиток вхідних даних. Алгоритм SHA-256 використовується у багатьох застосуваннях, таких як цифрові підписи, аутентифікація паролів та технологія блокчейну. Оскільки хеш-значення, створене за допомогою SHA-256, є унікальним, практично неможливо здійснити зворотну інженерію вхідних даних.

Хоча жоден криптографічний алгоритм не є невразливим до атак, алгоритм SHA-256 пройшов ретельний аналіз та випробування часом, залишаючись суттєвим елементом у забезпеченні безпеки цифрових даних.

Безпека SHA-256 використовує комбінацію складних математичних та бітових операцій для генерації хеш-значення. Ця конструкція робить надзвичайно важким знаходження двох вхідних даних, які виробляють однаковий хеш. Проте безпека SHA-256 залежить від коректного впровадження та міцності захисних протоколів, які використовуються в застосуваннях, що його використовують.

Одним із недоліків алгоритму SHA-256 є можливість атаки на основі колізії. Вона відбувається, коли два різних вхідних значення виробляють однаковий хеш. Хоча це вкрай малоймовірно з SHA-256, оскільки він генерує 256-бітне хеш-значення (тобто багато можливих вихідних даних), теоретично це все ще можливо. Якщо атака на основі колізії була б успішною, вона могла б скомпрометувати застосування, яке покладається на хеш-значення SHA-256 для перевірки цілісності даних.

Ключові характеристики алгоритму SHA-256 включають довжину повідомлення, довжину дайджесту та незворотність:

- Довжина повідомлення: Довжина відкритого тексту (тексту до його шифрування) повинна бути меншою за 264 бітів.
- Довжина дайджесту: Довжина хеш-дайджесту (результат застосування криптографічної хеш-функції до даних) повинна складати 256 бітів.
- Незворотність: Усі хеш-функції, такі як SHA-256, є незворотними за конструкцією. На кожний вхід є точно один вихід, але не навпаки. Різні вхідні дані можуть виробляти однаковий вихід. Вихід має фіксований розмір, але вхід не має обмежень щодо розміру.

Алгоритм криптографічної хеш-функції SHA-256

SHA-256 - це криптографічна хеш-функція з довжиною дайджесту 256 бітів. Це хеш-функція без ключа. Повідомлення обробляється блоками по $512 = 16 \times 32$ бітів, кожен блок потребує 64 раунди.

Основні операції:

- Булеві операції, позначені як $\wedge$, $\oplus$ і $\vee$ відповідно.
- Бітове доповнення, позначене як $^{-}$.

- Цілочисельне додавання за модулем 2^{32}, позначене як $A + B$.

Кожна з них діє на 32-бітні слова. Для останньої операції бінарні слова інтерпретуються як цілі числа, записані в системі з основою 2.

- $RotR(A, n)$ позначає циклічний правий зсув на n бітів бінарного слова A.
- $ShR(A, n)$ позначає правий зсув на n бітів бінарного слова A.
- $A \parallel B$ позначає конкатенацію (додавання) бінарних слів A та B.

Алгоритм використовує такі функції:

$$Ch(X, Y, Z) = (X \wedge Y) \oplus (\bar{X} \wedge Z),$$

$$Maj(X, Y, Z) = (X \wedge Y) \oplus (X \wedge Z) \oplus (\mathrm{Y} \wedge Z),$$

$$\sum_{0}(X) = RotR(X, 2) \oplus \mathrm{RotR(X, 13)} \oplus \mathrm{RotR(X, 22)},$$

$$\sum_{1}(X) = RotR(X, 6) \oplus \mathrm{RotR(X, 11)} \oplus \mathrm{RotR(X, 25)},$$

$$\sigma_0(X) = RotR(X, 7) \oplus \mathrm{RotR(X, 18)} \oplus \mathrm{ShR(X, 3)},$$

$$\sigma_1(X) = RotR(X, 17) \oplus \mathrm{RotR(X, 19)} \oplus \mathrm{ShR(X, 10)},$$

та 64 бінарні слова K_i, які визначаються як 32 перші біти дробових частин кубічних коренів перших 64 простих чисел:

```
0x428a2f98  0x71374491  0xb5c0fbcf  0xe9b5dba5  0x3956c25b  0x59f111f1  0x923f82a4  0xab1c5ed5
0xd807aa98  0x12835b01  0x243185be  0x550c7dc3  0x72be5d74  0x80deb1fe  0x9bdc06a7  0xc19bf174
0xe49b69c1  0xefbe4786  0x0fc19dc6  0x240ca1cc  0x2de92c6f  0x4a7484aa  0x5cb0a9dc  0x76f988da
0x983e5152  0xa831c66d  0xb00327c8  0xbf597fc7  0xc6e00bf3  0xd5a79147  0x06ca6351  0x14292967
0x27b70a85  0x2e1b2138  0x4d2c6dfc  0x53380d13  0x650a7354  0x766a0abb  0x81c2c92e  0x92722c85
0xa2bfe8a1  0xa81a664b  0xc24b8b70  0xc76c51a3  0xd192e819  0xd6990624  0xf40e3585  0x106aa070
0x19a4c116  0x1e376c08  0x2748774c  0x34b0bcb5  0x391c0cb3  0x4ed8aa4a  0x5b9cca4f  0x682e6ff3
0x748f82ee  0x78a5636f  0x84c87814  0x8cc70208  0x90befffa  0xa4506ceb  0xbef9a3f7  0xc67178f2
```

Ми припускаємо, що довжина повідомлення може бути представлена 64-бітним цілим числом. Щоб забезпечити, що довжина повідомлення має кратність 512 бітів:

- Спочатку додається біт 1,
- Потім додається k бітів 0, де k це найменше натуральне число таке, що
- $l + 1 + k \equiv 448 \, mod \, 512$, l є довжиною в бітах початкового повідомлення,
- Довжина $l < 2^{64}$ початкового повідомлення стає представленою точно 64 бітами і ці біти додаються в кінці повідомлення

Початкове повідомлення завжди доповнюється, навіть у викадку коли початкова довжина вже є кратною 512.

Для кожного блока $M \in \{0,1\}^{512}, 64$ слова по 32 біти кожне створюється наступним чином:

- перші 16 отримуються шляхом розділення M на 32-бітні блоки

$$M = W_1 \| W_2 \| \dots \| W_{15} \| W_{16}$$

- наступні 48 отримуються за формулою:

$$W_i = \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16}, \qquad 17 \le i \le 64.$$

Обчислення хешу

Спочатку восьми змінним присвоюються їхні початкові значення, які визначаються першими 32 бітами дробової частини квадратних коренів перших 8 простих чисел:

$H_1^0 = 0\text{x}6\text{a}09\text{e}667$ $H_2^0 = 0\text{xbb}67\text{ae}85$ $H_3^0 = 0\text{x}3\text{c}6\text{ef}372$ $H_4^0 = 0\text{xa}54\text{ff}53\text{a}$
$H_5^0 = 0\text{x}510\text{e}527\text{f}$ $H_6^0 = 0\text{x}9\text{b}05688\text{c}$ $H_7^0 = 0\text{x}1\text{f}83\text{d}9\text{ab}$ $H_8^0 = 0\text{x}5\text{be}0\text{cd}19$

- Далі блоки $M^{(1)}, M^{(2)}, \dots, M^{(N)}$ обробляються по порядку:

Для $t = 1$ до N

- сконструюємо 64 блоки W_i з $M^{(t)}$, як пояснено вище

- встановимо

$$(a, b, c, d, e, f, g, h) = (H_1^{t-1}, H_2^{t-1}, H_3^{t-1}, H_4^{t-1}, H_5^{t-1}, H_6^{t-1}, H_7^{t-1}, H_8^{t-1})$$

- проведемо 64 раунди, які складаються з:

$$T_1 = h + \sum_1(e) + Ch(e, f, g) + K_i + W_i$$

$$T_2 = \sum_0(a) + Maj(a, b, c)$$

$$h = g; g = f; f = e;\ e = d + T_1;\ d = c;\ c = b;\ b = a;\ a = T_1 + T_2$$

- обчислимо нове значення $H_j^{(t)}$:

$$H_1^{(t)} = H_1^{(t-1)} + a; H_2^{(t)} = H_2^{(t-1)} + b; H_3^{(t)} = H_3^{(t-1)} + c; H_4^{(t)} = H_4^{(t-1)} + d;$$
$$H_5^{(t)} = H_5^{(t-1)} + e; H_6^{(t)} = H_6^{(t-1)} + f; H_7^{(t)} = H_7^{(t-1)} + g; H_8^{(t)} = H_8^{(t-1)} + h$$

Завершення циклу.

Хеш повідомлення є конкатенацією змінних H_i^N після обробки останнього блоку

$$H = H_1^{(N)} \left\| H_2^{(N)} \right\| H_3^{(N)} \left\| H_4^{(N)} \right\| H_5^{(N)} \left\| H_6^{(N)} \right\| \left\| H_7^{(N)} \right\| H_8^{(N)}. [20]$$

Розглянемо приклад реалізації алгоритму криптографічної хеш-функції SHA-256 на мові програмування Python:

main.py

```python
import struct


10 usages
def right_rotate(value, amount):
    return ((value >> amount) | (value << (32 - amount))) & 0xFFFFFFFF


1 usage
def sha256(message):
    h0, h1, h2, h3, h4, h5, h6, h7 = (
        0x6A09E667, 0xBB67AE85, 0x3C6EF372, 0xA54FF53A,
        0x510E527F, 0x9B05688C, 0x1F83D9AB, 0x5BE0CD19
    )

    k = [
        0x428A2F98, 0x71374491, 0xB5C0FBCF, 0xE9B5DBA5,
        0x3956C25B, 0x59F111F1, 0x923F82A4, 0xAB1C5ED5,
        0xD807AA98, 0x12835B01, 0x243185BE, 0x550C7DC3,
        0x72BE5D74, 0x80DEB1FE, 0x9BDC06A7, 0xC19BF174,
        0xE49B69C1, 0xEFBE4786, 0x0FC19DC6, 0x240CA1CC,
        0x2DE92C6F, 0x4A7484AA, 0x5CB0A9DC, 0x76F988DA,
        0x983E5152, 0xA831C66D, 0xB00327C8, 0xBF597FC7,
        0xC6E00BF3, 0xD5A79147, 0x06CA6351, 0x14292967,
        0x27B70A85, 0x2E1B2138, 0x4D2C6DFC, 0x53380D13,
        0x650A7354, 0x766A0ABB, 0x81C2C92E, 0x92722C85,
        0xA2BFE8A1, 0xA81A664B, 0xC24B8B70, 0xC76C51A3,
        0xD192E819, 0xD6990624, 0xF40E3585, 0x106AA070,
        0x19A4C116, 0x1E376C08, 0x2748774C, 0x34B0BCB5,
        0x391C0CB3, 0x4ED8AA4A, 0x5B9CCA4F, 0x682E6FF3
```

sha256()

main.py

```python
        0x19A4C116, 0x1E376C08, 0x2748774C, 0x34B0BCB5,
        0x391C0CB3, 0x4ED8AA4A, 0x5B9CCA4F, 0x682E6FF3,
        0x748F82EE, 0x78A5636F, 0x84C87814, 0x8CC70208,
        0x90BEFFFA, 0xA4506CEB, 0xBEF9A3F7, 0xC67178F2
    ]
# Підготовка повідомлення
    message += b'\x80'
    while len(message) % 64 != 56:
        message += b'\x00'
    message += struct.pack('>Q', 8 * len(message))

# Обробка блоків
    for i in range(0, len(message), 64):
        w = [0] * 64
        for j in range(16):
            w[j] = struct.unpack('>I', message[i + j*4:i + j*4 + 4])[0]
        for j in range(16, 64):
            s0 = right_rotate(w[j - 15], 7) ^ right_rotate(w[j - 15], 18) ^ (w[j - 15] >> 3)
            s1 = right_rotate(w[j - 2], 17) ^ right_rotate(w[j - 2], 19) ^ (w[j - 2] >> 10)
            w[j] = (w[j - 16] + s0 + w[j - 7] + s1) & 0xFFFFFFFF

        a, b, c, d, e, f, g, h = h0, h1, h2, h3, h4, h5, h6, h7

        # Основний цикл
        for j in range(64):
            s0 = right_rotate(a, 2) ^ right_rotate(a, 13) ^ right_rotate(a, 22)
            maj = (a & b) ^ (a & c) ^ (b & c)
            t2 = s0 + maj
```

sha256()

```python
            t2 = s0 + maj
            s1 = right_rotate(e, 6) ^ right_rotate(e, 11) ^ right_rotate(e, 25)
            ch = (e & f) ^ ((~e) & g)
            t1 = h + s1 + ch + k[j] + w[j]

            h = g
            g = f
            f = e
            e = (d + t1) & 0xFFFFFFFF
            d = c
            c = b
            b = a
            a = (t1 + t2) & 0xFFFFFFFF

        h0 = (h0 + a) & 0xFFFFFFFF
        h1 = (h1 + b) & 0xFFFFFFFF
        h2 = (h2 + c) & 0xFFFFFFFF
        h3 = (h3 + d) & 0xFFFFFFFF
        h4 = (h4 + e) & 0xFFFFFFFF
        h5 = (h5 + f) & 0xFFFFFFFF
        h6 = (h6 + g) & 0xFFFFFFFF
        h7 = (h7 + h) & 0xFFFFFFFF

    return '%08x%08x%08x%08x%08x%08x%08x%08x' % (h0, h1, h2, h3, h4, h5, h6, h7)

# Приклад використання
print(sha256(b"An example of using a SHA-256 hash function algorithm"))
```

Результат :

```
/Users/bogdanlozovoj/PycharmProjects/pythonProject4/venv/bin/python /Users/bogdanlozovoj/
452902a14d70954b22231134560e23db944649f313e11205f151bf5fdb66e27d

Process finished with exit code 0
```

Узагальнення

Описати концепцію криптовалюти Bitcoin однією формулою є складною задачею, оскільки Bitcoin є комлексною системою, що включає в себе: криптографію, мережеві транзакції, алгоритми консенсусу, та економічні моделі.

Однак, узагальнену ключову ідею можна представити як:

$$H(B_n) = H(T_{r_x} \cdot H(B_{n-1}) \cdot Nonce)$$

де:

$H(B_n)$ – хеш n-го блоку в блокчейні;

H – криптографічна хеш-функція SHA-256, яка використовується в Bitcoin;

T_{r_x} − дані транзакції, що включені в блок;

$H(B_{n-1})$ − хеш попереднього $(n-1)$-го блоку в блокчейні;

$Nonce$ − число, яке майнери підбирають в процесі майнінгу для виконання умови доказу роботи (Proof-of-Work).

· −операція конкатенації

Ця формула відображає процес створення нового блоку в блокчейні Bitcoin, де кожен новий блок включає хеш попереднього блоку, дані транзакцій та спеціальне значення (nonce), яке підбирається для забезпечення дійсності хешу за умовами доказу роботи.

4.2 Математична модель криптовалюти Ethereum

Ethereum - це відкрита платформа та середовище, базоване на блокчейні, призначене для запуску децентралізованих додатків та смарт-контрактів. Вона також включає в себе внутрішню криптовалюту, яка називається Ethereum (ефір), ця криптовалюта використовується для фінансових обмінів та оплати комісій за транзакції або використання інших послуг на ній.

Ethereum використовує стандартні криптографічні хеш-функції Keccak-256 та Keccak-512 для генерації адрес облікових записів, підписів транзакцій та валідації блоків.

Введемо деякі позначення: нехай $\mathbb{A} = \{0,1\}^{160}$ буде множиною всіх можливих адрес, $\mathbb{B} = \{0, \dots, 255\}$ множиною байтів та $\mathbb{N}_{256}$ множиною натуральних чисел нижче 2^{256}. S^* позначає замикання Кліні множини S, а ε позначає символ, такий що $\{\varepsilon\}$ є порожнім елементом S^*.

Зірка Кліні (або оператор Кліні, або замикання Кліні) — це унарна операція на множині рядків скінченної довжини із символів. Застосування зірки Кліні до множини S позначається S^*.

Означення. Нехай $S_0 = \{\varepsilon\}$- порожній рядок. $S_{i+1} = \{wu : w \in S_i, u \in S\}, i \geq 0$. Тоді, $S^* = \bigcup_i S_i$.

4.2.1 Приватні та публічні ключі в середовищі Ethereum

У мережі Ethereum усі вузли використовують одну і ту ж еліптичну криву криптографії (ECC) для генерації пари публічного та приватного ключів.

Математичний процес генерації цих ключів включає кілька кроків:
Користувач генерує випадковий 256-бітний ключ. Це приватний ключ.

1	0	1	1	1	……	0	1	1	1	1

Рисунок 4.2 - 256-бітний приватний ключ

В еліптичній криптографії (ECC), використовуваній у мережі Ethereum, публічний ключ генерується з приватного ключа за допомогою операції еліптичного множення. Формула для цього виглядає як публічний ключ $= n \times G$, де $G -$ це генераторна точка (з заданими координатами x та y) на еліптичній кривій, що є стійкою до дискретного логарифмування, а $n -$ це приватний ключ.

1	1	1	0	0	0	0	0	……	1	1	1	1	0	0	0	0

Рисунок 4.3 - 512-бітний публічний ключ

У еліптичній криптографії Ethereum порядок точки G на еліптичній кривій дорівнює N, так що виконується умова:

$$\infty = N \times G = G + G + \cdots + G \ (N \text{ разів})$$

Приватний ключ $n < N$.

Публічний ключ формально виражається як $(n \times x, n \times y)$, де x та $y -$ координати точки G, і виконується операція конкатенації: публічний ключ = $(n \times x)||(n \times y)$.

Далі обчислюється Keccak-256 хеш цього публічного ключа:

Keccak-256 $((n \times x)||(n \times y))$

1. Ethereum адреса - це останні 160 біт цього хеша.

Обліковий запис.

Обліковий запис - це кортеж $A = (b, code, stor, n)$,

де:

- $b \in \mathbb{N}_{256}$ – це баланс
- $code \in \mathbb{B}^*$ є елементом замикання Кліні над множиною $\mathbb{B}$ (кодом). Таким чином, $code$ визначає програмний код смарт-контракту в Ethereum, який є послідовністю байтів, де кожен байт може мати будь-яке значення від 0 до 255. Обліковий запис є контрактом, якщо $code \neq \varepsilon$, та зовнішньо керованим в іншому випадку;
- $stor$ – це функція з $\{0,1\}^{256}$ у $\{0,1\}^{256}$, тобто може приймати 256-бітні числа як ключі та повертати 256-бітні числа як значення. У контексті смарт-контрактів Ethereum, функція $stor$ використовується для зберігання стану смарт-контракту. У цьому компоненті зберігається вся змінна інформація, яка потрібна для виконання контракту та його взаємодії з блокчейном.
- $n \in \mathbb{N}_{256}$ – це nonce. Якщо обліковий запис зовнішньо керований, то n буде кількістю транзакцій, відправлених з нього. Якщо обліковий запис є контрактом, в такому випадку n буде кількістю створень облікових записів.

Nonce облікового запису – це лічильник транзакцій або створень контрактів. Зовнішньо керовані облікові записи доступні за допомогою приватних ключів, а контракти можна взаємодіяти лише через виконання коду. Облікові записи ідентифікуються за їхніми адресами, а стан кожного облікового запису в мережі Ethereum становить глобальний стан системи.

4.2.2 Глобальний стан мережі

Глобальний стан системи - це функція між множиною адрес та множиною облікових записів:

$$\sigma: \mathbb{A} \to \mathbb{N}_{256} \times (\{0,1\}^{256} \to \{0,1\}^{256}) \times \mathbb{N}_{256}$$

Якщо $\sigma(a) = (b, code, stor, n)$, то ми кажемо, що a є “адресою” облікового запису $(b, code, stor, n)$.

Ми будемо використовувати слово "обліковий запис" для позначення сутності, що складається з адреси a та облікового запису $\sigma(a)$.

σ повинна відображати елементи $a \in \mathbb{A}$ на баланс, код, зберігання та nonce облікового запису з адресою a в блокчейні Ethereum.

Значення $\sigma(a)$ залежить від часу. $code$ є незмінним полем облікового запису, і воно визначається під час його створення. Ми припускаємо, що обліковий запис з адресою a існує, якщо $\sigma(a) \neq (0, \varepsilon, 0_F, 0)$, де 0_F - це постійна функція, значення якої завжди нуль.

4.2.3 Транзакції

Транзакції є засобом комунікації між обліковими записами. Транзакція містить переказ Ethereum, а також інструкції для активації виконання коду одержувача.

Транзакція - це кортеж $T = (from, to, n, v, d, g, p)$, де:

- $from, to \in \mathbb{A} -$ це адреси відправника та одержувача транзакції відповідно;
- $n \in \mathbb{N}_{256} -$ це nonce облікового запису з адресою $from$;
- $v \in \mathbb{N}_{256} -$ це значення;
- $d \in \mathbb{B}^* -$ це дані;
- $g \in \mathbb{N}_{256} -$ ліміт газу, тобто максимальна кількість газу, яка повинна бути використана в цій транзакції;
- $p \in \mathbb{N}_{256} -$ це ціна газу.

Якщо $t_0 = 0$, то T є транзакцією створення облікового запису, і d є кодом, що використовується для визначення коду нового облікового запису. В іншому випадку, $d -$ це дані, що подаються на вхід для виконання коду облікового запису з адресою to.

Позначимо множину вмісту транзакцій як:

$$\mathbb{T} = \mathbb{A} \times \mathbb{A} \times \mathbb{N}_{256} \times \mathbb{N}_{256} \times \mathbb{B}^* \times \mathbb{N}_{256} \times \mathbb{N}_{256}.$$

Транзакція створення облікового запису призводить до розміщення нового облікового запису A на блокчейні, що оновлює глобальну функцію стану σ. Адреса A – як ми вже знаємо, це число, сформоване з останніх 160 бітів результату хеш-функції KECCAK-256, застосованої до кодування $from$ та n. Таким чином, ймовірність перезапису вже існуючої адреси є надзвичайно малою. Баланс A становить v, значення, передане у транзакції. Код A є результатом виконання d. Зберігання A - це нульова функція, а nonce A становить 0.

Транзакція, де $t_0 \neq 0$, призводить до збільшення балансу облікового запису to на v одиниць, а також до зменшення балансу облікового запису $from$ на $v + gp$ одиниць, і до кожної зміни в глобальному стані, зробленої виконанням коду облікового запису to з входом d.

Коли транзакція передається в блокчейн Ethereum, який працює на основі механізму доказу частки (Proof of Stake, PoS), вона виконується валідаторами. Валідатори - це учасники мережі, що внесли депозит ефіру як заставу для участі в процесі валідації транзакцій і створення блоків. Виконання транзакції включає виконання коду, тому кожна транзакція вимагає оплати комісії в ефірі, рівної вартості використаного газу, помноженої на ціну газу p. Ціну газу p встановлює відправник, і вона обмежена максимальним лімітом газу, який визначає верхню межу можливої комісії. Якщо баланс відправника менший за цю максимальну комісію, транзакція вважається недійсною. У разі закінчення газу до завершення виконання, відправник платить максимальну комісію, але транзакція не завершується, і всі зміни, зроблені під час часткового виконання, скасовуються. Якщо газу достатньо, транзакція завершується, і валідатори оновлюють блокчейн, відображаючи зроблені зміни.

Газ у Ethereum - це механізм, що існує лише всередині транзакцій і використовується для обмеження ресурсів, які використовуються при виконанні транзакцій. Відокремлення понять газу і ефіру забезпечує, що вартість транзакцій не залежить безпосередньо від поточної ціни ефіру. Газ у

мережі Ethereum - це мірка, що використовується для вимірювання кількості обчислювальних зусиль, необхідних для виконання певних операцій, а саме виконання смарт-контрактів та обробці транзакцій. Вартість газу для різних операцій встановлюється заздалегідь і відображає обчислювальні зусилля, які вони вимагають.

Ціни на газ встановлюються користувачами, які ініціюють транзакції, і вони можуть вибирати їх, залежно від бажаного пріоритету виконання транзакції: валідатори можуть вибирати, які транзакції вони хочуть валідувати, зазвичай віддаючи перевагу тим, які пропонують вищу ціну газу. Вартість газу для транзакції $=$ кількість використаного газу $\times p$, де $p -$ це ціна газу. Кількість використаного газу залежить від типу та складності операцій, які виконуються у транзакції.

Ethereum вирішує проблему подвійного витрачання (цю проблему ми вже розглядали в Розділі 4), використовуючи nonce n облікового запису в транзакції. Нечесний обліковий запис A з балансом x може спробувати перевести y на рахунок B і z на рахунок C таким чином, що $y \leq x, z \leq x$ та $y + z > x$. Якщо транзакція до C транслюється раніше, ніж транзакція до B була оброблена, транзакція до C не буде відхилена через недостатність коштів, оскільки баланс A все ще становитиме x. Однак у системі Ethereum транзакції валідуються валідаторами, а не майнерами. Транзакція з вищою ціною газу може бути обрана для обробки першою, але через систему nonce, яка визначає послідовність транзакцій, друга транзакція буде визнана недійсною, якщо її nonce не дорівнює одиниці плюс останній nonce, використаний обліковим записом A.

Транзакції в системі Ethereum записуються всередині блоків, які є основними структурними одиницями блокчейну. Блоки містять різні компоненти, зокрема посилання на попередні блоки та структури даних, які зберігають інформацію про транзакції у блоку. Система доказу частки забезпечує валідацію та інтеграцію цих блоків до блокчейну, використовуючи процес, який базується не на обчислювальному зусиллі, а на заставі валідаторів.

4.2.4 Блок

Блок - це кортеж $B = (nonce, \mathcal{T}, i, l, ben, d)$, де:

- $nonce \in \mathbb{B}^* -$ nonce;
- $\mathcal{T} = (T_1, \dots, T_m) -$ список транзакцій;
- $i \in \mathbb{N} -$ номер;
- $l \in \mathbb{N} -$ ліміт газу;
- $ben \in \mathbb{A} -$ адреса облікового запису отримувача;
- $d \in \mathbb{N} -$ складність.

Множину вмісту блоків позначимо як $\mathcal{B}$.

$\mathcal{B} = \mathbb{B}^* \times \mathbb{T} \times \mathbb{N} \times \mathbb{N} \times \mathbb{A} \times \mathbb{N}$, де $\mathbb{T} -$ множина вмісту транзакцій.

Перший блок, коли-небудь розгорнутий у блокчейні, названий "генезис", має номер $i = 0$. Номер наступника блоку номер i, B_i, є блок B_{i+1}. Nonce блоку - це число, яке можна записати за допомогою 8 байтів або 64 бітів, і використовується у перевірці його дійсності.

4.2.5 Блокчейн Ethereum

Блокчейн Ethereum - це упорядкована послідовність блоків, $(B_0, B_1, \dots, B_k)$, таких, що $B_0 = (42, \emptyset, 0, 3141592, 0, 2^{17})$.

Кожен блок B_i може бути представлений як:

$$B_i = \left(Header, T_{x_i}\, state_{i-1}, state_i\right),$$

де:

- $Header$ включає блок-інформацію, таку як ідентифікатор попереднього блоку, ліміт газу та інше.
- $T_{x_i} -$ це множина транзакцій в блоці B_i;
- $state_{i-1} -$ це стан блокчейну до блоку B_i;
- $state_i -$ це стан блокчейну після застосування транзакцій з блоку B_i.

У контексті консенсусу доказу частки, процес вибору блоку для додавання до блокчейну здійснюється на основі стейкінгу та валідації. Генезис-блок B_0 є фіксованим. Кожен блок у блокчейні повинен бути дійсним. Дійсність блоку перевіряється через кілька процедур, включаючи перевірку дійсності кожної

транзакції (підтвердження nonce та перевірка підпису відправника) та валідаторами. [22]

4.3 Криптографічна хеш-функція Keccak-256

Keccak-256 - це алгоритм хешування блокчейну Ethereum, який забезпечує безпеку даних. На даний момент цей хеш-алгоритмом ще ніколи не був зламаний. Для криптографічної хеш-функції Keccak-256 ймовірність колізії дуже низька (число можливих результатів більше, ніж кількість атомів у Всесвіті).

4.3.1 Математична модель хеш-функції Keccak-256

У рамках розробки стандарту SHA-3, алгоритм Keccak-256 це ключовий компонент, який використовує унікальну конструкцію та параметри. Зокрема, для $n = 256$ використовується конфігурація:

$$n = 256\colon \lfloor Keccak[r = 1088, c = 512] \rfloor_{256}$$

n: позначає довжину вихідного повідомлення

r: швидкість передачі даних

c: ємність

Keccak є губчастою функцією:

$$Keccak[r, c] \triangleq sponge[Keccak - f[r + c], pad10^*, r].$$

В криптографії губчаста функція або губчаста конструкція - це клас алгоритмів із скінченним внутрішнім станом, які приймають вхідний потік бітів будь-якої довжини та виробляють вихідний потік бітів будь-якої бажаної довжини.

Губчаста функція описується трьома компонентами

1) Пам’ять стану S, що містить b бітів
2) Функція $f\colon \{0,1\}^b \to \{0,1\}^b$ генерує пседовипадкову перестановку з 2^b станів S.
3) Фунція доповнення P.

S ділиться на дві частини: одна розміром r (швидкість передачі даних) та інша розміром c (ємність).

Функція P додає достатньо бітів до вхідного рядка, так що довжина доповненого вводу є цілим кратним r, тобто вхідний рядок поділяється на блоки по r бітів.

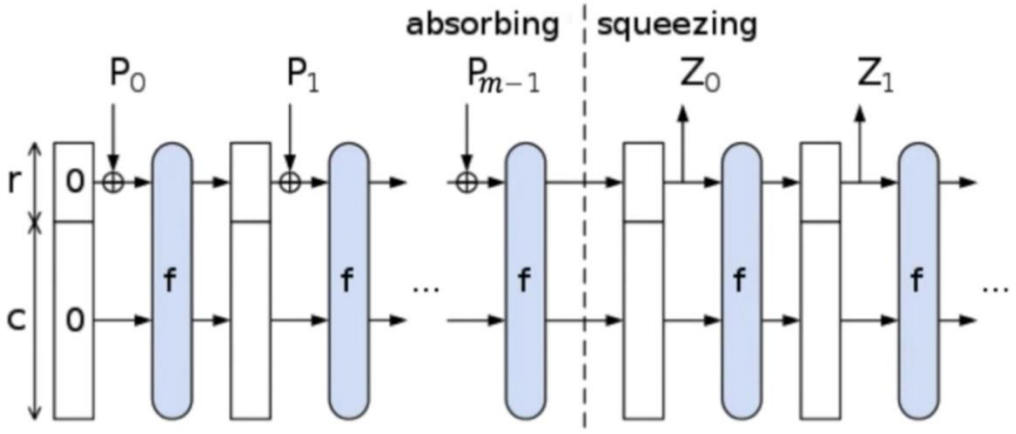


Рисунок 4.4

4.3.2 Алгоритм хеш-функції Keccak-256:

1) Доповнення повідомлення M:

$$P = M||pad[r](|M|)$$

- Доповнення (pad) $10 \dots 1$, довжина кратна r
- Розділення на $P_0, P_1, \dots, P_{m-1}$

На першому кроці вхідне повідомлення M доповнюється таким чином, щоб його загальна довжина стала кратною r. Після цього доповнене повідомлення розділяється на ряд блоків $P_0, P_1, \dots, P_{m-1}$, кожен з яких має довжину r.

Цей крок потрібен для того аби повідомлення було ефективно оброблене на наступних кроках алгоритму.

2) Фаза поглинання (absorbing):

- $S = o^b, b = r + c = 1600$ - розмір стану
- Для кожного P_i:

- Доповнення (Pad) $P_i||0^{b-r}$
- $S = S \oplus P_i||0^{b-r}$
- $S = f(S)$

На другому кроці стан S ініціалізується як нульовий вектор розміром b бітів. Для кожного блоку P_i відбувається додавання XOR цього блоку до частини стану S, що відповідає r, із подальшим застосуванням функції перестановки $f(S)$. Ця фаза забезпечує поглинання вхідних даних у стан алгоритму.

3) Фаза вижимання (squeezing):

- $Z_0 = \lfloor S \rfloor_r$
- Якщо $|Z_0| \geq n$:
- $Z = Z_0$
- Вивід $\lfloor Z \rfloor_n$ як хеш − значення Keccak − n
- Якщо $|Z_0| < n$:
- $S = f(S)$
- $Z_1 = \lfloor S \rfloor_r$
- $Z = Z_0 || Z_1$

На третьому кроці зі стану S витягується частина даних розміром r бітів як перший блок виводу Z_0. Якщо довжина Z_0 достатня ($|Z_0| \geq n$), вона виводиться як хеш-значення.

У випадку якщо $|Z_0| < n$, виконується додатковий раунд перестановки $f(S)$ для генерації наступного блоку виводу Z_1 і так далі, доки не буде отримано достатньо даних для виводу.

4) f є функцією перестановки з n_r раундів R

Стан $S[w(5y + x) + z]$ відображається на $a[x][y][z]$, де $x = 0, \dots, 4, y = 0, \dots, 4, z = 0, \dots, 2^l - 1$.

Встановлюємо $l = 6$, так що $w = w^l = 64$ та $1600 = 5 \times 5 \times w$ and $n_r = 12 + 2l = 24$.

$R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$, де

θ: $a[x][y][z] \leftarrow a[x][y][z] + \sum_{y'}^{4} a[x-1][y'][z] + \sum_{y'}^{4} a[x+1][y'][z-1]$,

виконується побітове додавання для кожного елемента стану з елементами з попереднього і наступного рядка.

ρ: $a[x][y][z] \leftarrow a[x][y]\left[z - \frac{(t+1)(t+2)}{2}\right]$, здійснюється круговий зсув елементів стану.

π: : $a[x][y] \leftarrow a[x'][y'], with \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}$, переставляє елементи стану в певному порядку.

χ: $a[x] \leftarrow a[x] + (a[x+1] + 1)a[x+2]$, нелінійна трансформація кожного рядка стану.

ι: $a \leftarrow RC[i_r]$ додавання раундової сталої до стану.

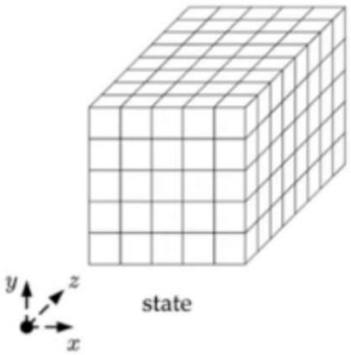


Рисунок 4.5 – візуалізація внутрішнього стану алгоритму.

Кожен квадрат у цій тривимірній структурі відповідає окремому біту або елементу стану. [21]

4.3.3 Алгоритм криптографічної хеш-функція Keccak-256 на мові програмування Python

```python
# сталі для Keccak-256
RC = [
    0x0000000000000001, 0x0000000000008082, 0x800000000000808A,
    0x8000000080008000, 0x000000000000808B, 0x0000000080000001,
    0x8000000080008081, 0x8000000000008009, 0x000000000000008A,
    0x0000000000000088, 0x0000000080008009, 0x000000008000000A,
    0x000000008000808B, 0x800000000000008B, 0x8000000000008089,
    0x8000000000008003, 0x8000000000008002, 0x8000000000000080,
    0x000000000000800A, 0x800000008000000A, 0x8000000080008081,
    0x8000000000008080, 0x0000000080000001, 0x8000000080008008
]
# Таблиця індексів для перемішування байтів.
r = [
    [0, 36, 3, 41, 18],
    [1, 44, 10, 45, 2],
    [62, 6, 43, 15, 61],
    [28, 55, 25, 21, 56],
    [27, 20, 39, 8, 14]
]

1 usage
def keccak256(data):
    # Ініціалізуємо матрицю `S` розміром 5x5 з нулів.
    S = [[0] * 5 for _ in range(5)]

    # Перетворюємо вхідні дані в байти.
    data = bytearray(data)
    data.append(0x01)  # Додаємо 0x01 до вхідних даних із кінця для додавання додаткового байта 0x0

    # Визначаємо функцію для обертання байта (бітова перестановка).
    def rotate_left(x, n):
        return ((x << n) | (x >> (64 - n))) & 0xFFFFFFFFFFFFFFFF
```

```
    # Починаємо абсорбцію вхідних даних в матрицю `S`.
    for i in range(0, len(data), 72):
        chunk = data[i:i + 72]
        lanes = [0] * 25  # Перетворюємо частину вхідних даних в масив 64-бітних цілих чисел
        # Збільшили обсяг масиву lanes до 25, щоб було достатньо елементів.
        for j in range(min(25, len(chunk))):
            lanes[j] = chunk[j]
        # Виконуємо абсорбцію для кожного байта вхідних даних.
        # Виконуємо XOR з відповідним байтом вхідних даних та відповідним байтом матриці `S`.
        for j in range(min(25, len(chunk))):
            S[j % 5][j // 5] ^= lanes[j]


        for round in range(24):

            # тета функція
            C = [S[x][0] ^ S[x][1] ^ S[x][2] ^ S[x][3] ^ S[x][4] for x in range(5)]
            D = [C[(x - 1) % 5] ^ rotate_left(C[(x + 1) % 5], 1) for x in range(5)]
            for x in range(5):
                for y in range(5):
                    S[x][y] ^= D[x]

            # ро та пі функція
            new_S = [[0] * 5 for _ in range(5)]
            for x in range(5):
                for y in range(5):
                    new_S[y][(2 * x + 3 * y) % 5] = rotate_left(S[x][y], r[x][y])
            S = new_S

            # Хі функція
```

```
            # Хі функція
            for y in range(5):
                T = [S[x][y] for x in range(5)]
                for x in range(5):
                    S[x][y] = T[x] ^ ((~T[(x + 1) % 5]) & T[(x + 2) % 5])

            # йота функція
            S[0][0] ^= RC[round]

    # Виконуємо виведення хешу.
    # Отримуємо хеш з 25 64-бітних лейнів і конкатенуємо їх.
    hash_result = ''.join([format(S[x][y], '016x') for x in range(5) for y in range(5)])

    return hash_result

# Приклад
data = b'Example Keccak 256'
hashed_data = keccak256(data)
print(f'Хеш для вхідних даних "{data.decode()}": {hashed_data}')
```

Результат:

```
main
/Users/bogdanlozovoj/PycharmProjects/pythonProject6/venv/bin/python /Users/bogdanlozovoj/PycharmP
Хеш для вхідних даних "Example Keccak256": b46386c9e4c70ccd513645d5edd60ddd0aea7c021d886c4cfc4d39

Process finished with exit code 0
```

Отримали Хеш-значення:

b46386c9e4c70ccd513645d5edd60ddd0aea7c021d886c4cfc4d3985166ad5c9125b18
97004401aef12c30ad86f0f9afb7592735ed0574a734c81012b2
af49007a30f2ad3db6fa0f15212768ff41ba5e02499e7166600873f886bab6a0e8c87b4c

46c3a0307fd5777c8ed267846b11c228840b728d9930cad582bc04881e762193183a19
1425ac5b74da35e9b15b3dcf5ffead0f0341b2c8c530355736b93d890102eac1a637dc8
cd2e73be17ba1d0c3c10ff71c1e097885ac131a124c6692203b5435bd6d112fcd

Узагальнення:

Узагальнену ключову ідею Ethereum можна представити як:

$$H(B_n) = H(T_{r_x} + H(B_{n-1}) + State + Signature)$$

де:

$H(B_n) -$ хеш n-го блоку в блокчейні Ethereum;

$T_{r_x} -$ дані транзакції, що включені в блок;

$H(B_{n-1}) -$ хеш попереднього $(n-1)$-го блоку в блокчейні;

$State -$ стан системи Ethereum після застосування транзакцій;

$Signature -$ підпис валідатора, що підтверджує блок у рамках консенсусу доказу частки (Proof of Stake).

$H -$ криптографічна хеш-функція Keccak-256, яка використовується в Ethereum.

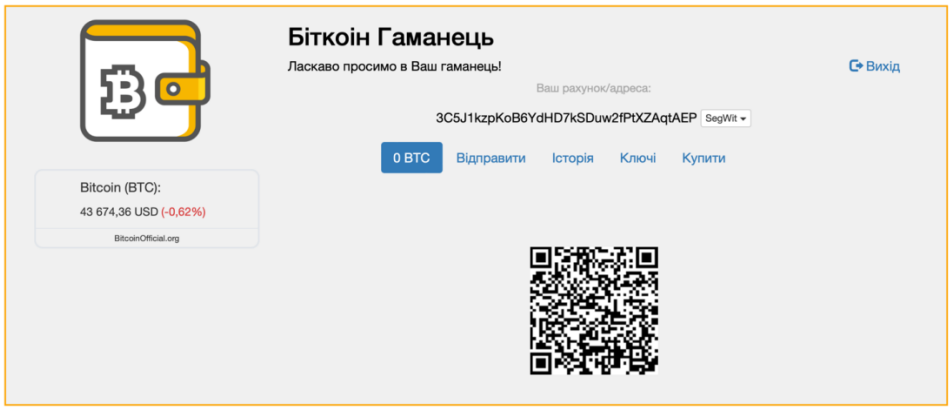
Ця формула відображає процес створення нового блоку в блокчейні Ethereum, описує хешування даних транзакцій, хеш попереднього блоку, оновлений стан системи після транзакцій, та підпис валідатора.

4.4 Порівняння Bitcoin і Ethereum та створення криптовалютних гаманців

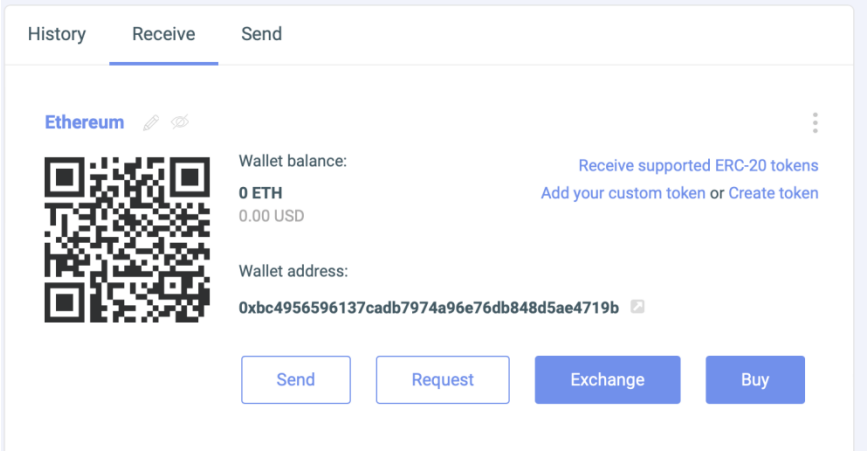
Обидві криптовалюти використовують сучасні методи для забезпечення безпеки, але роблять це за допомогою різних підходів. Bitcoin використовує рандомізацію для створення приватних ключів, що є критично важливим для захисту кожного гаманця. Ethereum, в свою чергу, застосовує рандомізацію у своїх Proof of Stake (доказ частки) протоколах, що забезпечує додаткову безпеку на рівні мережі та знижує ризики централізації. Крім того, обидві криптовалюти демонструють високу стійкість проти криптоаналітичних атак, оцінювану через криптографічну складність хеш-функцій. Також, здатність обох систем протистояти спробам зламу покращується зі збільшенням кількості вузлів у мережі. В цілому, Bitcoin і Ethereum мають свої унікальні переваги у забезпеченні безпеки, засновані на різних принципах та підходах, що робить їх

ключовими гравцями в екосистемі криптовалют. Після ретельного аналізу та порівняльного вивчення характеристик цих криптовалют я вирішив створити криптогаманці для Bitcoin та Ethereum. Вибір цих валют обумовлений їх значною популярністю, інноваційністю та, що найважливіше, різними підходами до забезпечення безпеки та децентралізації.

Криптовалюний гаманець Bitcoin:



Криптовалюний гаманець Ethereum:



Впровадження практичного елементу створення гаманців дозволить мені не лише глибше зрозуміти теоретичні аспекти, на яких базуються ці криптовалюти, але й набути реального досвіду використання та управління криптовалютними активами. Це допоможе максимально наблизити теоретичні знання до практичного застосування, що є важливим аспектом цілісного розуміння теми криптовалют.

ВИСНОВОК

У цій магістерській дисертації проведено детальний аналіз криптографічної безпеки криптовалют, з акцентом на технології блокчейну та їх застосуванні у сфері криптовалют. Особлива увага приділена вступу, де описано сучасний стан технологій, та першому розділу, який зосереджується на концепції та структурі блокчейну. Досліджено важливість рандомізованих алгоритмів, які зменшують ймовірність колізій, тим самим підсилюючи захист криптовалют. Ключову роль у забезпеченні безпеки відіграють хеш-функції, які є фундаментальним елементом математичної моделі криптовалют. Були математично описані алгоритми криптографічних хеш-функцій MD5, SHA256, Keccak256, крім того побудовані алгоритми цих функцій на мові програмування Python. Новизна дослідження полягає у створенні унікальної математичної моделі криптовалют, яка включає в себе розгляд новітніх методів забезпечення безпеки та ефективності. Через детальний огляд криптографічних методів та їхнього застосування у криптовалютах, дисертація підкреслює критичну роль криптографії у забезпеченні безпеки, цілісності та приватності цифрових транзакцій.

Загалом, в цій роботі були висвітлені ключові аспекти блокчейну, криптографії та їх застосування у фінансовому секторі, відкриваючи нові перспективи для подальших досліджень та розвитку в цій області.

Література:

1. Стефанчук Р. Інформаційні технології та право: quo vadis? *Право України*. 2018. № 1. С. 30–50. DOI: https://doi.org/10.33498/louu-2018-01-030
2. Спасітєлєва С. О., Бурячок В. Л. Перспективи розвитку додатків блокчейн в Україні. *Кібербезпека: освіта, наука, техніка*. 2018. № 1. С. 35–48.
3. Базанов С. Криптовалюта: терміни та скорочення. URL: https://medium.com/bitcoin-review/bitcoin-криптовалюты-термины-и-сокращения27293b8413cc. (дата звернення: 10.11. 2021).
4. Zibin Zheng and Shaoan Xie, Hong-Ning Dai. Blockchain challenges and opportunities: a survey Int. J. Web and Grid Services, Vol. 14, No. 4, 2018: https://allquantor.at/blockchainbib/pdf/zheng2018blockchain.pdf
5. Lamport L., Pease M., Shostak R. The Byzantine Generals Problem : *ACM Transactions on Programming Languages and Systems*, 2010. P. 382–401.
6. https://www.bitbon.space/ua/knowledge-base/distributed-ledger-technologies-blockchain/technological-aspects-of-blockchain/classification-of-blockchains
7. Григор'єв В. В. Технологія блокчейн як фактор зростання економіки / Тенденції та перспективи розвитку, 2019. С. 486–491.
8. https://cryptomus.com/uk/blog/how-does-a-blockchain-technology-work
9. https://www.h-x.technology/ua/blog-ua/blockchain-security-issues-ua
10. https://www.h-x.technology/ua/blog-ua/what-is-blockchain-security-examples-issues-and-solutions-ua
11. https://www.ssldragon.com/blog/sha-256-algorithm/#:~:text=The%20key%20features%20of%20the,be%20less%20than%20264%20bits.
12. https://helix.stormhub.org/papers/SHA-256.pdf
13. Dylan Yaga, Peter Mell, Nik Roby, Karen Scarfone. NISTIR 8202 Blockchain Technology Overview.
14. https://komodoplatform.com/en/academy/cryptographic-hash-function/

15. https://atlasvpn.com/blog/birthday-attack#:~:text=A%20birthday%20attack%20is%20a,known%20as%20the%20birthday%20paradox.
16. https://it-enterprise.com/knowledge-base/architecture-security/ellektronnaja-cifrovaja-podpis
17. Що таке Смарт-Контракти – [Електронний ресурс]. – https://cryptobook.pro/shcho-take-smart-kontrakty.html.
18. Proof of Work vs Proof of Stake – [Електронний ресурс]. – ‘https://www.sitepoint.com/proof-of-stake-vs-proof-of-work/.
19. https://blockchain.ieee.org/images/files/pdf/techbriefs-2022-q3/the-mathematics-behind-blockchain.pdf
20. https://canopee-group.com/wp-content/uploads/2020/05/Blockchain-Coperneec.pdf
21. https://wiki.rugdoc.io/docs/introduction-to-ethereums-keccak-256-algorithm/
22. Mathematics and Data Structures in Blockchain and Ethereum. Seied veria Hoseini
23. Algorithm Design. Jon Kleinberg, Eva Tardos 800 p.