Міністерство освіти і науки України
Кам’янець-Подільський національний університет імені Івана Огієнка
Фізико-математичний факультет
Кафедра комп’ютерних наук

Кваліфікаційна робота магістра
з теми: «**Вдосконалення методів підвищення конфіденційності та цілісності даних в криптовалютних блокчейнах**»

Виконав: здобувач вищої освіти групи KN1-M23 спеціальності 122 Комп’ютерні науки
Задорожний Володимир Володимирович
Керівник: Смалько Олена Аркадіївна, доцент, кандидат педагогічних наук
Рецензент: Сорич Віктор Андрійович, доцент, кандидат фізико -математичних наук

м. Кам’янець-Подільський – 2024 р.

АНОТАЦІЯ

Магістерська робота присвячена дослідженню методів забезпечення конфіденційності та цілісності даних у блокчейн-системах. У роботі розглянуто основні криптографічні методи, такі як хешування та шифрування, а також їхню роль у блокчейн-технологіях. Особливу увагу приділено вдосконаленню існуючих методів шифрування та хешування для підвищення продуктивності та надійності систем. Описано алгоритми оптимізації процесів шифрування та представлені фрагменти реалізації відповідних рішень. Практичне значення роботи полягає у можливості застосування отриманих результатів для покращення захисту даних у криптовалютних блокчейнах та інших децентралізованих системах.

Ключові слова: блокчейн, хешування, криптографія, алгоритми шифрування, конфіденційність, цілісність даних.

ANNOTATION

The master's thesis focuses on the study of methods for ensuring data confidentiality and integrity in blockchain systems. The work examines key cryptographic methods, such as hashing and encryption, and their role in blockchain technologies. Particular attention is given to improving existing encryption and hashing methods to enhance system performance and reliability. Algorithms for optimizing encryption processes are described, along with illustrative code implementations of the proposed solutions. The practical significance of the study lies in the applicability of the obtained results to improve data protection in cryptocurrency blockchains and other decentralized systems.

Keywords: blockchain, hashing, cryptography, encryption algorithms, confidentiality, data integrity.

ЗМІСТ

ВСТУП

Сучасний розвиток цифрових технологій значно збільшує обсяги даних, що зберігаються та передаються у мережах. Блокчейн-технології стали фундаментом для багатьох галузей, включаючи фінанси, логістику, охорону здоров’я, але зі зростанням їх популярності виникають численні виклики щодо забезпечення конфіденційності та цілісності даних. Незважаючи на використання надійних криптографічних алгоритмів, такі аспекти, як відстеження транзакцій, ризики компрометації даних та кібератаки, потребують пошуку нових підходів для підвищення рівня безпеки.

У контексті криптовалютних блокчейнів проблема конфіденційності стає особливо актуальною через прозорість мережі, де будь-який користувач має доступ до історії транзакцій. Водночас забезпечення цілісності даних є критичним для довіри до децентралізованих систем. Таким чином, дослідження методів, які вдосконалюють ці аспекти, має велике значення для подальшого розвитку галузі блокчейн-технологій.

Зі зростанням потужності обчислювальних систем та розвитком технологій аналізу даних виникають нові виклики для криптографічних алгоритмів, що лежать в основі блокчейнів. Хоча квантові комп'ютери поки що залишаються теоретичним інструментом, зростає ймовірність їхньої появи у майбутньому, що може поставити під загрозу безпеку існуючих алгоритмів, таких як RSA та ECC, які використовуються для шифрування й підпису даних. Водночас традиційні виклики блокчейн-технологій включають масштабування мережі через значний обсяг транзакцій, потребу в підвищеній швидкості обробки та оптимізації зберігання даних. Розвиток технологій аналізу, зокрема машинного навчання та штучного інтелекту, підсилює загрози анонімності користувачів, оскільки нові методи дозволяють ідентифікувати транзакційні зв'язки та потенційно розкривати учасників мережі. Тому дослідження нових алгоритмів шифрування, методів забезпечення конфіденційності та підвищення

ефективності систем є критично важливим для стійкого розвитку блокчейн-технологій.

Об’єктом дослідження є механізми забезпечення конфіденційності та цілісності даних у криптовалютних блокчейнах.

Предметом дослідження є вдосконалення методів криптографічного захисту даних у блокчейн-системах для підвищення їхньої конфіденційності, цілісності та ефективності обробки.

Метою дослідження є вдосконалення методів і алгоритмів криптографічного захисту даних у блокчейн-системах для підвищення їхньої конфіденційності, цілісності та ефективності обробки, враховуючи сучасні технології й процеси обробки інформації.

Завдання дослідження:

1. Дослідити сучасні моделі, методи та алгоритми криптографічного захисту даних, що використовуються у блокчейн-системах.
2. Виявити ключові загрози для конфіденційності, цілісності та ефективності обробки даних у блокчейн-мережах.
3. Описати та реалізувати алгоритми вдосконалених методів хешування та шифрування, що покращують конфіденційність та забезпечують цілісність даних у блокчейні.

Методи дослідження. Для досягнення поставленої мети використано методи теоретичного аналізу, моделювання криптографічних алгоритмів, експериментального дослідження їх продуктивності та порівняльного аналізу для оцінки ефективності запропонованих рішень.

Практичне значення роботи. Результати дослідження можуть бути використані для підвищення безпеки та ефективності криптографічного захисту даних у блокчейн-системах, зокрема у криптовалютних платформах, децентралізованих фінансових застосунках та інших інформаційних системах.

Апробація результатів досліджень проводилась упродовж двох міжнародних конференцій, зокрема під час 10-ї Міжнародної наукової

конференції «Сучасні проблеми математичного моделювання, прогнозування та оптимізації» пам’яті почесного професора Кам’янець-Подільського національного університету імені Івана Огієнка, д.т.н., професора, почесного академіка НАПНУ Анатолія Федоровича ВЕРЛАНЯ (м. Кам’янець-Подільський), а також Міжнародної науково-методичної Інтернет-конференції «Проблеми вищої математичної освіти: виклики сучасності» (м. Вінниця). За результатами роботи конференцій опубліковано тези: «Модель криптоблокчейну з посиленим захистом конфіденційності» [3], «Перспективні математичні концепції та технології для забезпечення цілісності даних у криптоблокчейнах» [4].

Структура роботи. Робота складається з трьох розділів. У першому розділі розглядаються основи шифрування в блокчейні та його роль у забезпеченні конфіденційності й цілісності. Другий розділ присвячено аналізу поширених методів захисту криптоблокчейнів та опису способів вдосконалення методів забезпечення конфіденційності та цілісності у розподілених базах даних. У третьому розділі описуються алгоритми вдосконалення деяких методів хешування та шифрування, що використовуються в блокчейнах. У чотирьох додатках наводяться коди програмних модулів з результатами їхньої роботи. Загальний обсяг роботи складає N сторінок. У списку використаних джерел міститься 32 інформаційних джерела.

РОЗДІЛ 1
ОСНОВИ ШИФРУВАННЯ ДЛЯ ЗАХИСТУ ДАНИХ У БЛОКЧЕЙНІ

1.1 Використання криптографії у блокчейні

Криптографія виступає ключовим елементом, на якому ґрунтується функціонування криптовалютних систем, забезпечуючи захист, збереження цілісності та конфіденційність даних у децентралізованих мережах. Вона дозволяє захистити транзакції від підробок, підтвердити їхню автентичність та узгодити дані між учасниками мережі без необхідності у довіреному посереднику. Серед основних криптографічних методів, які використовуються у криптовалютах, ключову роль відіграють хешування, криптографія з відкритим ключем, цифрові підписи, протоколи конфіденційності та механізми консенсусу. Кожен із цих методів забезпечує унікальний аспект безпеки, створюючи комплексний захист блокчейн-систем.

Одним із найважливіших компонентів криптографії у криптовалютах є хешування, яке використовується для забезпечення цілісності даних і підтримки структури блокчейна. Хеш-функції перетворюють будь-який обсяг даних у фіксований за довжиною хеш, який слугує унікальним ідентифікатором цих даних. У Bitcoin, наприклад, застосовується хеш-функція SHA-256, яка генерує 256-бітний результат, чутливий навіть до найменших змін у вихідних даних. Окрім забезпечення цілісності транзакцій, хешування відіграє критичну роль у механізмах консенсусу, таких як Proof of Work, де майнери розв’язують складні задачі, ґрунтуючись на хеш-функціях. Цей процес гарантує, що нові блоки додаються до блокчейна лише після виконання обчислювальної роботи, що захищає систему від атак.

Хешування тісно пов’язане з використанням криптографії з відкритим ключем, яка забезпечує автентифікацію і шифрування даних. У цій системі використовуються дві математично пов’язані ключі: публічний, доступний усім, і приватний, що зберігається в секреті. Одним із найбільш поширених алгоритмів, застосовуваних у криптовалютах, є ECDSA (Elliptic Curve Digital

Signature Algorithm), що базується на властивостях еліптичних кривих. Цей алгоритм використовується для створення цифрових підписів у Bitcoin, дозволяючи підтвердити, що транзакція була ініційована власником приватного ключа. Завдяки своїй ефективності ECDSA забезпечує високий рівень безпеки з мінімальними обчислювальними витратами, що є важливим для масштабованості блокчейнів. Розвиток криптографії з відкритим ключем також сприяв появі нових підходів, таких як EdDSA, який має підвищену стійкість до атак.

Цифрові підписи є ключовим інструментом автентифікації транзакцій у блокчейнах, забезпечуючи захист від несанкціонованих змін і підробок. Алгоритми цифрових підписів, такі як ECDSA і Schnorr-підписи, дозволяють гарантувати, що транзакція була створена власником приватного ключа, а також забезпечують механізми для об'єднання підписів у багатосторонніх угодах. Schnorr-підписи, наприклад, є вдосконаленим методом, який дозволяє зменшити розмір підписів і підвищити конфіденційність шляхом агрегування підписів кількох учасників. У Monero використовується спеціалізована технологія кільцевих підписів (Ring Signatures), яка забезпечує анонімність, приховуючи, хто саме з групи учасників підписав транзакцію. Таким чином, цифрові підписи слугують критичним елементом для забезпечення автентичності та конфіденційності.

Питання конфіденційності додатково вирішуються за допомогою протоколів конфіденційності, таких як Zero-Knowledge Proofs (ZKPs). Ці протоколи дозволяють учасникам підтвердити істинність певного твердження, не розкриваючи жодної додаткової інформації. Наприклад, у криптовалюті Zcash застосовуються zk-SNARKs, що дають змогу підтверджувати транзакції без розголошення інформації про відправника, отримувача або суму. Подібні підходи знаходять застосування у захисті даних у смарт-контрактах і децентралізованих фінансових платформах. Більш сучасні протоколи, такі як zk-STARKs, усувають необхідність у довірених налаштуваннях, що робить їх особливо перспективними для широкого використання в блокчейні.

Окремо слід розглянути механізми консенсусу, які є основою функціонування децентралізованих мереж. У Bitcoin використовується механізм Proof of Work, що забезпечує безпеку мережі за рахунок виконання обчислювально складних завдань. Альтернативні моделі, такі як Proof of Stake (PoS), пропонують енергоефективніший підхід, де нові блоки генеруються відповідно до частки володіння монетами. Інноваційні механізми, як-от Proof of Authority (PoA) і Proof of Space (PoSpace), розширюють можливості блокчейнів, забезпечуючи високу швидкість і захист даних. Ці моделі підтримують цілісність мережі та забезпечують надійну роботу навіть у складних умовах.

1.2 Загрози конфіденційності та цілісності даних у блокчейні

Блокчейн-технологія, завдяки своїй прозорості, забезпечує високу ступінь децентралізації та доступності даних. Проте саме ця прозорість створює суттєві ризики для конфіденційності учасників системи. Усі транзакції, що здійснюються у публічних блокчейнах, доступні для перегляду будь-якому користувачеві мережі. Це дозволяє зловмисникам проводити складний аналіз даних і, використовуючи сучасні методи, отримувати інформацію, що може порушити приватність користувачів.

Однією з головних загроз є можливість відстеження публічних адрес, які пов’язані з транзакціями. Унікальні ідентифікатори, що використовуються у блокчейні, з одного боку, забезпечують захист від фальсифікацій, але з іншого — відкривають можливості для аналізу даних. Наприклад, зловмисники можуть використовувати техніки кластеризації адрес, щоб об'єднувати транзакції за спільними параметрами і визначати, які з них належать одному користувачу. Хоча адреси не пов'язані безпосередньо з реальними іменами, цей зв'язок можна встановити за допомогою додаткових даних із зовнішніх джерел.

Атаки на транзакційні зв’язки є наступним кроком після ідентифікації публічних адрес. Зловмисники можуть використовувати аналіз графів транзакцій для виявлення ланцюжків переказів і зв’язків між різними учасниками мережі.

Особливо вразливими є сценарії, коли транзакції пов’язуються з метаданими, такими як IP-адреси або часові мітки, що можуть бути доступні через мережевий трафік. Об'єднання цієї інформації з відкритими даними блокчейна створює значні ризики для приватності користувачів, дозволяючи ідентифікувати їхні фінансові операції та особистості.

Ще однією суттєвою вразливістю є відсутність захисту метаданих. Навіть якщо вміст транзакцій шифрується або використовується технологія приховування суми операцій, допоміжна інформація залишається доступною. Наприклад, часові мітки, геолокаційні дані або інформація про мережеву активність можуть бути використані для побудови детальних профілів користувачів. Зокрема, аналіз частоти транзакцій і часу їхнього виконання може дати змогу встановити поведінкові моделі, які полегшують ідентифікацію.

Окрім того, багато публічних блокчейнів, таких як Bitcoin, не мають інтегрованих механізмів анонімізації. У таких умовах конфіденційність залежить виключно від поведінки користувача. У той час як спеціалізовані криптовалюти, такі як Monero або Zcash, реалізують технології анонімності, наприклад кільцеві підписи або zk-SNARKs, більшість блокчейнів дозволяють аналізувати реєстр без суттєвих обмежень. Це створює простір для масового збору даних та їхнього використання третіми сторонами для компрометації конфіденційності учасників мережі.

Цілісність даних у блокчейні є однією з ключових характеристик, яка забезпечує довіру до системи. Вона гарантує, що дані, записані у блокчейні, залишаються незмінними, доки це не санкціоновано відповідними механізмами консенсусу. Проте цілісність може бути поставлена під загрозу через різноманітні вразливості або цілеспрямовані атаки, спрямовані на порушення роботи мережі чи маніпуляцію даними.

Однією з найвідоміших загроз є атаки 51%, які можуть виникати в системах, що використовують Proof of Work (PoW) як механізм консенсусу. Якщо одна група майнерів отримує контроль над більш ніж половиною обчислювальної потужності мережі, вона отримує можливість змінювати історію

транзакцій, скасовувати платежі або навіть проводити атаки подвійного витрачання. Такі дії підривають довіру до блокчейна, оскільки його головна перевага — незмінність даних — стає сумнівною. Ця загроза особливо актуальна для блокчейнів із низьким рівнем децентралізації, де кілька великих майнінгових пулів можуть концентрувати значні обчислювальні ресурси.

Ще однією суттєвою вразливістю є атаки на механізми консенсусу, такі як Proof of Stake (PoS) або Proof of Authority (PoA). У PoS система вибору валідаторів залежить від частки володіння монетами, що може стати слабким місцем у разі централізації активів. У PoA ризики пов’язані з довірою до обмеженого кола авторитетних вузлів, які можуть стати мішенню атак або діяти недобросовісно. Вразливості консенсусу відкривають шлях для маніпуляцій у верифікації транзакцій, що порушує цілісність блокчейна.

Окрему категорію загроз становлять атаки на смарт-контракти, які реалізуються у блокчейні. Смарт-контракти є програмним кодом, що виконується автоматично за заданими умовами. Проте помилки у коді або недоліки в дизайні можуть стати причиною серйозних вразливостей. Наприклад, злам DAO у мережі Ethereum у 2016 році став результатом помилки у смарт-контракті, що дозволило зловмисникам вивести значну кількість коштів. Цей інцидент підкреслює важливість ретельного аудиту коду та використання формальних методів перевірки для забезпечення безпеки.

Хоча блокчейн за своєю природою є децентралізованою системою, вузли мережі та інфраструктура залишаються вразливими до атак типу DDoS (розподілених атак на відмову в обслуговуванні) або інших видів зловмисної активності. Атаки DDoS можуть перевантажувати вузли, перешкоджаючи їхньому нормальному функціонуванню і порушуючи доступність системи. Інфраструктурні вразливості також включають ризики перехоплення даних або компрометації обладнання, що може призвести до часткової втрати даних або впливу на їхню цілісність.

Деякі загрози блокчейн-систем мають подвійний характер, оскільки вони одночасно впливають на конфіденційність і цілісність даних. Такі атаки не лише

ставлять під сумнів безпеку конкретних транзакцій, але й можуть порушити функціонування всієї системи, знижуючи її надійність та стійкість. Комбіновані загрози виникають через вразливості у механізмах синхронізації, аутентифікації чи зберігання даних і потребують комплексного підходу до їхнього виявлення та нейтралізації.

Одним із яскравих прикладів комбінованих загроз є синхронізаційні атаки (Timejacking), які спрямовані на маніпуляцію часовими параметрами мережі. У блокчейні кожен вузол синхронізує свій локальний час із мережею для забезпечення коректності обробки транзакцій і формування блоків. Зловмисник, використовуючи помилкові часові мітки, може вплинути на синхронізацію окремих вузлів. Це призводить до розриву консенсусу, створення "сирітських блоків" або навіть форкування ланцюга. Такі наслідки не лише порушують цілісність даних, але й можуть розкрити транзакційну активність, створюючи додаткові загрози конфіденційності.

Ще одним поширеним видом комбінованих атак є повторні атаки (Replay Attacks), які полягають у повторному використанні даних транзакції в іншому контексті. Наприклад, зловмисник може захопити інформацію про транзакцію, таку як адреса, сума або хеш, і повторно відправити її в мережу, видаючи за нову операцію. Якщо система не використовує механізми захисту, такі як унікальні ідентифікатори або часові мітки, така атака може бути успішною. Результатом є не лише фінансові втрати для користувачів, але й компрометація конфіденційності, оскільки повторні транзакції дозволяють аналізувати взаємодії між учасниками мережі.

Особливу увагу заслуговує той факт, що комбіновані загрози, такі як Timejacking та Replay Attacks, можуть мати кумулятивний ефект. Маніпуляції з часовими мітками, наприклад, можуть створювати сприятливі умови для повторних атак, оскільки вузли мережі втрачають здатність ідентифікувати унікальні транзакції в умовах хаосу. Це ставить під загрозу не лише окремих користувачів, але й загальну стабільність блокчейна.

Для мінімізації ризиків комбінованих загроз необхідно впроваджувати комплексні захисні механізми. Синхронізаційні атаки можуть бути попереджені шляхом використання більш стійких протоколів часу, таких як Network Time Protocol (NTP), із додатковими перевірками достовірності часових міток. Для захисту від повторних атак необхідно впроваджувати унікальні ідентифікатори транзакцій, які включають криптографічно захищені часові мітки, а також використовувати двофакторну аутентифікацію на рівні передачі даних.

1.3 Механізми та методи забезпечення конфіденційності та цілісності даних в блокчейні

Цілісність даних у блокчейні є однією з основних властивостей, що забезпечує довіру до системи та її функціонування у децентралізованому середовищі. Цей принцип гарантує, що транзакції та записи у реєстрі залишаються незмінними та достовірними, доки це не санкціоновано заздалегідь визначеним механізмом консенсусу. Досягнення цілісності є результатом інтеграції кількох криптографічних методів і архітектурних рішень, які перешкоджають модифікації даних зловмисниками або їхній компрометації.

Одним із ключових механізмів забезпечення цілісності є хешування, що використовується для створення унікальних цифрових відбитків даних. Хеш-функції перетворюють вхідну інформацію будь-якого розміру у фіксований за довжиною хеш, який слугує унікальним ідентифікатором цих даних. Ці відбитки змінюються навіть при найменшій модифікації вхідних даних, що робить хешування ефективним захистом від фальсифікації. Найбільш поширеними хеш-функціями є SHA-256, яка забезпечує високий рівень криптографічної безпеки та застосовується в мережі Bitcoin, та Blake2, яка використовується в інших криптовалютах завдяки своїй швидкості та стійкості до атак. У блокчейні хешування також застосовується для збереження транзакцій у вигляді структурованих даних, які формують блоки.

Ще однією важливою структурою для забезпечення цілісності даних є Мерклові дерева (Merkle Trees). Це структура даних, яка дозволяє ефективно перевіряти транзакції в блоці, зберігаючи їх у вигляді хешів. Кожна транзакція спочатку хешується, після чого результати об'єднуються в дерево, кореневий хеш якого слугує унікальним ідентифікатором всього блоку. Якщо будь-яка транзакція змінюється, це одразу змінює кореневий хеш, сигналізуючи про порушення цілісності. Мерклові дерева значно оптимізують процес перевірки транзакцій у великих блокчейн-мережах, знижуючи обчислювальні витрати та зберігаючи цілісність даних.

Окрім хешування, цілісність даних гарантують криптографічні підписи, які забезпечують автентичність транзакцій. Цифрові підписи дозволяють підтвердити, що транзакція була ініційована саме її автором, а не стороннім зловмисником. Серед найбільш поширених алгоритмів цифрових підписів у блокчейнах — ECDSA (Elliptic Curve Digital Signature Algorithm) та Schnorr-підписи. Перший є стандартом у багатьох криптовалютах, включаючи Bitcoin, завдяки високій ефективності та криптографічній стійкості. Schnorr-підписи, у свою чергу, забезпечують вищу продуктивність і конфіденційність, дозволяючи агрегувати кілька підписів в одну транзакцію, що зменшує її розмір і підвищує швидкість обробки.

Невід'ємною частиною забезпечення цілісності даних є механізми консенсусу, які координують дії всіх учасників мережі. Найбільш поширеним є механізм Proof of Work (PoW), який використовується в Bitcoin. Він вимагає від майнерів виконання обчислювально складних задач для підтвердження транзакцій і додавання нових блоків, що унеможливлює підробку даних через високі витрати ресурсів. Альтернативою є механізм Proof of Stake (PoS), який базується на економічній зацікавленості учасників мережі: валідатори обираються відповідно до частки їхніх активів. У PoS ризик порушення цілісності зменшується завдяки штрафам за недобросовісну поведінку. Інший підхід, Proof of Authority (PoA), передбачає використання довірених вузлів для верифікації даних, що робить його ефективним для приватних блокчейнів.

Забезпечення конфіденційності даних є одним із ключових викликів для блокчейн-технологій, особливо у відкритих децентралізованих системах, де вся інформація про транзакції зазвичай доступна для перегляду будь-якому учаснику мережі. Така прозорість сприяє довірі, проте водночас створює ризики компрометації приватності користувачів, оскільки їхні фінансові операції та транзакційні зв’язки можуть бути відстежені. Щоб подолати ці загрози, впроваджуються різноманітні методи та механізми, які спрямовані на захист як самих даних, так і метаданих, пов'язаних із транзакціями.

Одним із базових методів захисту конфіденційності є шифрування, яке дозволяє зберігати дані у зашифрованому вигляді, забезпечуючи доступ до них лише авторизованим сторонам. Існують два основних типи шифрування, що застосовуються у блокчейнах. Симетричне шифрування, зокрема алгоритм AES (Advanced Encryption Standard), використовується для захисту невеликих обсягів даних, коли ключ доступний обом сторонам. Натомість асиметричне шифрування, представлене алгоритмами RSA (Rivest-Shamir-Adleman) та ECC (Elliptic Curve Cryptography), використовується для безпечної передачі даних між сторонами завдяки використанню пари ключів: відкритого і приватного. Ці підходи забезпечують високий рівень захисту навіть у разі передачі даних через публічні канали.

Докази з нульовим розголошенням, або Zero-Knowledge Proofs (ZKPs), є інноваційним підходом до захисту конфіденційності, що дозволяє одному учаснику мережі підтвердити істинність певного твердження без розкриття будь-якої додаткової інформації. У криптовалютах ZKPs знаходять широке застосування для приховування даних транзакцій. Наприклад, у Zcash використовуються zk-SNARKs (Succinct Non-Interactive Arguments of Knowledge), які дозволяють забезпечити повну анонімність транзакцій, приховуючи адреси відправника, отримувача та суми. Більш сучасна версія цієї технології, zk-STARKs (Scalable Transparent Arguments of Knowledge), відрізняється підвищеною масштабованістю та усуває потребу в довірених

налаштуваннях, що робить її перспективною для великих децентралізованих мереж.

Кільцеві підписи (Ring Signatures) є ще одним ефективним методом для захисту конфіденційності. Ця технологія працює за принципом створення групи можливих підписувачів, серед яких приховується справжній ініціатор транзакції. Використання кільцевих підписів, наприклад у Monero, забезпечує високий рівень анонімності, оскільки неможливо визначити, хто саме підписав транзакцію. Це унеможливлює відстеження фінансових операцій навіть за наявності додаткових даних.

Для збереження приватності суми транзакцій застосовуються конфіденційні транзакції (Confidential Transactions, CT). Цей метод дозволяє приховувати суми, не порушуючи при цьому коректності перевірки транзакцій. У таких криптовалютах, як Monero та Grin, CT дають змогу зберігати фінансову конфіденційність, навіть якщо інші аспекти транзакцій залишаються відкритими для мережі.

Окрім захисту самих транзакцій, значну увагу приділяють захисту метаданих, таких як часові мітки, геолокація та поведінкові моделі користувачів. Для цього використовуються додаткові протоколи, зокрема Tor та I2P, які приховують мережевий трафік, мінімізуючи ризики аналізу сторонніми особами. Такі технології унеможливлюють ідентифікацію джерела та призначення транзакцій, забезпечуючи додатковий рівень конфіденційності.

Отже, забезпечення конфіденційності у блокчейнах досягається завдяки інтеграції криптографічних методів і протоколів анонімності. Впровадження шифрування, доказів з нульовим розголошенням, кільцевих підписів та захисту метаданих створює багаторівневу систему захисту, яка ефективно протидіє сучасним загрозам. Подальший розвиток цих методів, зокрема вдосконалення ZKPs та гомоморфного шифрування, дозволить забезпечити ще вищий рівень конфіденційності у майбутніх децентралізованих системах.

Розвиток блокчейн-технологій та сучасних криптографічних рішень стимулює появу інноваційних підходів, які спрямовані на підвищення рівня

конфіденційності та цілісності даних. Ці підходи базуються на інтеграції новітніх алгоритмів, що забезпечують не лише стійкість до сучасних загроз, але й масштабованість і ефективність у виконанні транзакцій. Зокрема, серед ключових технологій виділяються гомоморфне шифрування, постквантова криптографія та алгоритми динамічного шифрування.

Одним із перспективних методів є гомоморфне шифрування, яке дозволяє виконувати обчислення над зашифрованими даними без їхнього розшифрування. Це відкриває нові можливості для забезпечення конфіденційності у фінансових транзакціях і смарт-контрактах, де учасники можуть взаємодіяти з даними без доступу до їхнього змісту. Наприклад, у контексті блокчейнів це дозволяє виконувати аналіз великих даних або проводити обчислення у смарт-контрактах, не розкриваючи конфіденційну інформацію. Такий підхід особливо актуальний для децентралізованих фінансових систем (DeFi), де конфіденційність та обчислювальна ефективність є критичними.

Ще одним важливим напрямком є постквантова криптографія, яка покликана забезпечити стійкість блокчейнів до загроз, пов’язаних із розвитком квантових обчислень. Квантові комп'ютери здатні вирішувати складні математичні задачі, які є основою сучасних криптографічних алгоритмів, таких як RSA чи ECDSA, що ставить під загрозу безпеку багатьох систем. Постквантові алгоритми, як-от Lattice-based cryptography, засновані на задачах решіткової алгебри, що є стійкими до квантових атак завдяки своїй складності. Інший перспективний підхід, NTRU, базується на багаточленах і також демонструє високий рівень безпеки. Впровадження постквантової криптографії у блокчейни забезпечить їхню стійкість до загроз майбутнього, дозволяючи захистити як дані, так і механізми консенсусу.

Окрім цього, у блокчейні активно досліджуються та впроваджуються алгоритми динамічного шифрування, які змінюються під час виконання транзакцій. Цей підхід ускладнює будь-які атаки в реальному часі, оскільки алгоритми адаптуються до змін у мережі чи вхідних даних. Наприклад, у динамічних системах ключі шифрування можуть оновлюватися автоматично

після кожної транзакції, що мінімізує ризики компрометації ключів або зламу алгоритму. Такі механізми є важливими для забезпечення безперервної безпеки у швидко змінюваних середовищах, характерних для децентралізованих мереж.

РОЗДІЛ 2
ЗАБЕЗПЕЧЕННЯ КОНФІДЕНЦІЙНОСТІ ТА ЦІЛІСНОСТІ ДАНИХ У КРИПТОВАЛЮТНИХ БЛОКЧЕЙНАХ

2.1 Поширені методи забезпечення конфіденційності та цілісності даних в блокчейні

Хеш-функції є математичними алгоритмами, які перетворюють вхідні дані довільної довжини у фіксоване хеш-значення. Вони широко застосовуються для перевірки цілісності даних, ідентифікації інформації, прискорення процесів пошуку та в криптографії. Основною метою використання хеш-функцій є забезпечення надійності та ефективності обробки даних.

До ключових властивостей хеш-функцій належить детермінованість, яка гарантує, що однакові вхідні дані завжди генерують однаковий хеш. Важливою характеристикою є також односторонність, що робить обчислення вихідних даних з хешу вкрай складним, забезпечуючи високий рівень безпеки. Хеш-функції повинні бути стійкими до колізій, тобто унеможливлювати знаходження двох різних наборів вхідних даних, які дають однакове хеш-значення.

Особливою рисою хеш-функцій є їхня чутливість до змін у вхідних даних, відома як ефект лавини: навіть найменша модифікація викликає суттєву зміну хешу. Крім того, незалежно від розміру вхідних даних, хеш-функції генерують результат фіксованої довжини, що забезпечує зручність і прогнозованість їх використання в різних додатках. Ці властивості роблять хеш-функції основою багатьох сучасних криптографічних та інформаційних систем.

SHA-256 є криптографічною хеш-функцією, яка використовується для створення хеш-значень фіксованої довжини (256 бітів) незалежно від розміру вхідних даних. Вона базується на послідовній обробці вхідних даних через серію раундів, які використовують комбінацію побітових операцій, логічних функцій і числових обчислень. Розглянемо цей процес детально.

Першим етапом є перетворення вхідних даних (наприклад, тексту чи файлу) у бінарну форму. Ця послідовність бітів розбивається на блоки фіксованого розміру. Для SHA-256 кожен блок має довжину 512 бітів. Якщо останній блок не відповідає потрібному розміру, застосовується доповнення. Доповнення складається з трьох частин:

- Один біт зі значенням "1".
- Нулі, щоб вирівняти довжину блоку до 448 бітів (залишаючи 64 біти для запису довжини вхідних даних).
- 64-бітове представлення довжини початкових даних у бітах.

Для виконання обчислень використовуються вісім початкових фіксованих значень, відомих як *ініціалізаційний вектор* (IV). Ці значення визначені стандартом SHA-256 і представлені у шістнадцятковій системі.

Ці значення слугують стартовими параметрами для обчислень і гарантують стійкість алгоритму. Кожен блок розширюється до 64 слів довжиною 32 біти кожне. Початкові 16 слів утворюються безпосередньо з вхідного блоку, а наступні 48 обчислюються за допомогою рекурентної формули:

$$W_i = \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} W_i = \sigma_1 W_{i-2} + W_{i-7} + \sigma_0 W_{i-15} + W_{i-16}$$

де:

- $\sigma_0(x) = ROTR(7,x) \oplus ROTR(18,x) \oplus (x \gg 3)$
- $\sigma_1(x) = ROTR(17,x) \oplus ROTR(19,x) \oplus (x \gg 10)$

Розширені дані використовуються для подальшої обробки у 64 раундах. Кожен раунд виконується з використанням таких логічних функцій:

- *Ch (Choice)*:

$$Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z)$$

Ця функція вибирає значення між y та z, залежно від x.

- *Maj (Majority)*:

$$Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$

Ця функція повертає значення більшості серед x, y, z.

- *Великі сигма-функції*:

$$\Sigma_0(x) = ROTR(2,x) \oplus ROTR(13,x) \oplus ROTR(22,x)$$

$$\Sigma_1(x) = ROTR(6,x) \oplus ROTR(11,x) \oplus ROTR(25,x)$$

На кожному раунді оновлюються проміжні значення a, b, c, d, e, f, g, h. Вони розраховуються за формулами:

$$T_1 = h + \Sigma_1(e) + Ch(e,f,g) + K_i + W_i$$

$$T_2 = \Sigma_0(a) + Maj(a,b,c)$$

Після цього значення оновлюються так:

$$h = g,\ g = f,\ f = e,\ e = d + T_1$$

$$d = c,\ c = b,\ b = a,\ a = T_1 + T_2$$

Константи K_i визначені стандартом і забезпечують унікальність кожного раунду.

Після завершення всіх раундів проміжні значення додаються до початкових значень:

$$H_i = H_i + a,\ H_{i+1} = H_{i+1} + b\ , \dots ,\ H_{i+7} = H_{i+7} + h$$

Результат являє собою 256-бітове хеш-значення.

Для прикладу розглянемо хешування тексту "hello" перетворюється у бінарну форму, розбивається на блоки і обробляється за наведеним алгоритмом. Результат для SHA-256:

2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b.

Це значення є унікальним для даного тексту і не може бути обернено до початкових даних завдяки властивостям хеш-функції. SHA-256 забезпечує надійність і безпеку хешування завдяки складній математичній структурі, яка унеможливлює зворотне обчислення, стійкість до колізій і високу чутливість до змін у вхідних даних. Це робить її невід'ємною частиною сучасних криптографічних систем.

RSA (Rivest-Shamir-Adleman) — це асиметричний криптографічний алгоритм, що базується на складності факторизації великих чисел. Алгоритм використовує дві ключові операції: шифрування за допомогою відкритого ключа та розшифрування за допомогою закритого ключа.

Генерація ключів RSA ґрунтується на виборі двох великих простих чисел p та q. Добуток цих чисел формує модуль n, що є основою для шифрування та розшифрування:

$$n = p \cdot q$$

Визначається функція Ейлера:

$$\phi(n) = (p-1)(q-1)$$

Далі обирається ціле число e (відкрита експонента), що є взаємно простим із $\phi(n)$, тобто:

$$\gcd\big(e, \phi(n)\big) = 1$$

За допомогою розширеного алгоритму Евкліда обчислюється закрита експонента d, яка задовольняє:

$$e \cdot d \equiv 1\big(mod\phi(n)\big)$$

Таким чином, відкритий ключ складається з пари (e, n), а закритий ключ із пари (d, n).

Для шифрування повідомлення M, де $\mathrm{M} \in \mathrm{Z}_n$ (тобто $0 \leq M < n$), застосовується відкрита експонента e:

$$C \equiv M^e (mod\, n)$$

де C — це зашифрований текст (ciphertext).

Розшифрування здійснюється за допомогою закритої експоненти d:

$$M \equiv C^d (mod\, n)$$

Завдяки властивостям модульної арифметики та оберненої експоненти, M можна коректно відновити з C.

Безпека RSA базується на складності факторизації числа n. Навіть якщо n є відомим, знаходження простих множників p та q для обчислення закритого ключа d є обчислювально складною задачею.

AES (Advanced Encryption Standard) — це симетричний алгоритм блочного шифрування, що працює з блоками розміром 128 біт та підтримує ключі довжиною 128, 192 або 256 біт. Алгоритм базується на замінах, перестановках і операціях у скінченному полі $GF(2^8)$.

Вхідні дані розбиваються на блоки по 128 біт, де кожен блок представлений матрицею 4×4 байтів (state matrix).

Для кожного раунду генерується раундовий ключ на основі початкового ключа. Процес включає застосування функцій обертання (RotWord), заміни байтів (SubWord) з використанням S-Box та додавання констант (Rcon):

$$W_i = W_{i-4} \oplus \text{SubWord}\big(\text{RotWord}(W_{i-1})\big) \oplus Rcon$$

Цей процес забезпечує ключі для кожного раунду.

Алгоритм AES виконує N_r раундів шифрування (10, 12 або 14 раундів залежно від довжини ключа). Кожен раунд складається з чотирьох основних операцій:

SubBytes. Кожен байт матриці state замінюється відповідним значенням із нелінійної таблиці S-Box, що забезпечує заплутаність (confusion).

Нехай x — байт у $GF(2^8)$. Операція SubBytes реалізується через інверсію у скінченному полі та афінне перетворення.

ShiftRows. Байти в кожному рядку матриці state циклічно зсуваються на різну кількість позицій.

MixColumns. Кожен стовпець матриці state множиться на фіксовану матрицю M у $GF(2^8)$:

$$\text{State}' = M \cdot \text{State}$$

Ця операція забезпечує дифузію (diffusion).

AddRoundKey. Матриця state додається за допомогою операції XOR до раундового ключа:

$$State' = State \oplus RoundKey$$

Перший раунд включає лише AddRoundKey, а фінальний раунд не виконує MixColumns.

Розшифрування AES є зворотним процесом, що включає операції InvSubBytes, InvShiftRows, InvMixColumns та AddRoundKey. Зворотний S-Box використовується для InvSubBytes, а зворотні матричні операції забезпечують коректне відновлення даних.

AES забезпечує високий рівень стійкості завдяки складним нелінійним і дифузійним операціям, а також розміру ключа. Його структура оптимізована для ефективної реалізації як у програмному, так і в апаратному середовищі.

Алгоритм RSA надає високий рівень безпеки завдяки складності факторизації великих чисел, але він менш продуктивний у порівнянні із симетричним шифруванням. AES, у свою чергу, є надзвичайно швидким і ефективним для шифрування великих обсягів даних, оскільки його математична реалізація оптимізована для сучасних обчислювальних платформ.

Post-Quantum Extended Diffie-Hellman (PQXDH) — це гібридний криптографічний протокол, який поєднує класичний алгоритм ECDH (Elliptic Curve Diffie-Hellman) із постквантовими алгоритмами, такими як CRYSTALS-Kyber або інші протоколи, що належать до постквантової криптографії. Його основна ідея полягає в тому, що для розрахунку спільного секрету використовується два незалежних ключі, один ключ — генерується традиційним методом (наприклад, ECDH), а інший ключ — генерується за допомогою постквантового алгоритму.

Для обчислення спільного секрету сторони обчислюють комбіновану функцію, що включає обидва компоненти. Зламати таку систему можливо лише у разі одночасного порушення обох криптографічних механізмів, що є надзвичайно складним завданням навіть для квантових комп'ютерів.

Блокчейн-системи широко покладаються на ECDH для обміну ключами та шифрування даних. Проте квантові комп'ютери можуть розв'язати задачу дискретного логарифму на еліптичних кривих за допомогою алгоритму Шора, що робить ECDH вразливим.

PQXDH забезпечує стійкість до квантових атак шляхом додавання постквантових механізмів до традиційного обміну ключами. Це гарантує, що навіть за умови зламу одного з компонентів (наприклад, ECDH), загальна безпека залишається непорушною. Застосування у блокчейнах нового покоління:

- Конфіденційність у транзакціях: PQXDH може використовуватися для обміну секретними ключами між учасниками транзакцій у блокчейнах, таких як Ethereum, Monero, або інших платформ із високими вимогами до конфіденційності.

- Захист смарт-контрактів: У смарт-контрактах PQXDH дозволяє створити стійкі до квантових атак механізми для обміну даними або підписування угод.
- Керування ключами: PQXDH забезпечує безпечний обмін ключами для систем підписів, захищених з'єднань та зберігання даних у блокчейнах.

Деякі сучасні блокчейн-платформи розглядають гібридний криптографічний підхід, де постквантові протоколи поєднуються із традиційними методами для поступового переходу до квантово-стійких систем. Наприклад: CRYSTALS-Kyber використовується разом із ECDH у PQXDH для обміну ключами. Впровадження PQXDH у блокчейни дозволяє забезпечити зворотну сумісність із традиційними системами, одночасно захищаючи від майбутніх загроз.

Переваги PQXDH у блокчейні заключаються в Стійкості до квантових атак, яка базується на поєднання класичної та постквантової криптографії робить PQXDH безпечним навіть за наявності квантових комп'ютерів. Захист транзакцій, який забезпечує надійний обмін ключами для конфіденційного шифрування даних у блокчейні. Масштабованість та ефективність завдяки, яким постквантові алгоритми, такі як CRYSTALS-Kyber, є досить продуктивними та підходять для впровадження у децентралізованих мережах.

PQXDH є перспективним методом для використання в блокчейн-системах завдяки своїй стійкості до квантових атак. Гібридний підхід, що об'єднує класичні алгоритми, такі як ECDH, з постквантовими методами, дозволяє блокчейнам зберегти конфіденційність, цілісність та надійність навіть у світі квантових обчислень. Активні дослідження та впровадження PQXDH є важливим кроком у підготовці блокчейн-технологій до ери квантових обчислень.

2.2 Способи вдосконалення методів забезпечення конфіденційності та цілісності даних в блокчейні

Однією з ключових операцій, які використовуються у SHA-256, є функції перетасування $\Sigma_0(x)$, $\Sigma_1(x)$, $\sigma_0(x)$, $\sigma_1(x)$. Вони побудовані на основі циклічних

зсувів (*ROTR*) і операцій *XOR* ($\oplus$), які сприяють рівномірному розподілу вхідних даних у вихідному хеші. Наприклад, функція $\Sigma_0(x)$ визначається як:

$$\Sigma_0(x) = ROTR(2,x) \oplus ROTR(13,x) \oplus ROTR(22,x)$$

де *ROTR(n, x)* позначає циклічний зсув числа x на n позицій праворуч. Ці операції забезпечують сильний ефект лавини, коли навіть незначна зміна у вхідних даних суттєво змінює вихідний хеш. Хоча побітові операції є ефективними, вони можуть бути оптимізовані на сучасних процесорах, які мають високу продуктивність для операцій множення, порівняно з побітовими зсувами.

Пропоноване вдосконалення полягає у заміні побітових циклічних зсувів математичними операціями множення за модулем великого простого числа p. Це дозволяє зберегти криптографічну стійкість і рівномірний розподіл даних, водночас підвищуючи ефективність алгоритму на сучасних апаратних архітектурах. Формула для $\Sigma_0(x)$ у вдосконаленій реалізації виглядає наступним чином:

$$\Sigma_0(x) = \left((x \cdot 2^2)\ mod\ p\right) \oplus \left((x \cdot 2^{13})\ mod\ p\right) \oplus \left((x \cdot 2^{22})\ mod\ p\right)$$

де p — велике просте число, вибране так, щоб відповідати криптографічним вимогам до хеш-функції.

Множення за модулем p створює ефект, схожий на циклічні зсуви, але додає новий шар випадковості, що посилює захист від деяких криптографічних атак, наприклад, атак на колізії. Просте число p виступає як додатковий параметр, який можна варіювати для досягнення різних рівнів стійкості та швидкості.

Операція $(x \cdot 2^n)$ має такі властивості:

- Вона створює унікальну послідовність значень для будь-якого x, подібно до побітового зсуву.
- Використання модуля p зменшує ймовірність утворення структур у вихідних даних, які могли б використовуватися для аналізу алгоритму.
- Модульна арифметика ефективно реалізується на сучасних процесорах, забезпечуючи високу продуктивність.

Вдосконалення функції $\Sigma_0(x)$ на основі модульної арифметики забезпечує кілька ключових переваг. Ефективність на апаратному рівні досягається завдяки оптимізації сучасних процесорів для операцій множення, що дозволяє обробляти великі обсяги даних швидше, ніж при використанні побітових операцій. Зменшення кількості кроків у процесі обчислень знижує загальну складність алгоритму, скорочуючи кількість необхідних інструкцій і підвищуючи продуктивність. Підвищення стійкості забезпечується завдяки використанню додаткового параметра p, який змінює структуру вихідних значень, ускладнюючи їхнє прогнозування та знижуючи ймовірність атак на основі симетрії. Нарешті, гнучкість налаштувань дозволяє адаптувати алгоритм під специфічні потреби, вибираючи різні значення p, що дозволяє балансувати між швидкістю виконання та криптографічною стійкістю функції.

Для демонстрації вдосконалення візьмемо $x = 123456$, $p = 2^{31} - 1$ (велике просте число):

$$\Sigma_0(x) = \left((123456 \cdot 2^2) mod (2^{31} - 1)\right) \oplus \left((123456 \cdot 2^{13}) mod (2^{31} - 1)\right) \oplus \left((123456 \cdot 2^{22}) mod (2^{31} - 1)\right)$$

Обчислення для кожного доданку:

$$\left((123456 \cdot 2^2)\ mod\ (2^{31} - 1)\right) = 493824,$$

$$\left((123456 \cdot 2^{13})\ mod (2^{31} - 1)\right) = 1017118720,$$

$$\left((123456 \cdot 2^{22})\ mod (2^{31} - 1)\right) = 1275068416.$$

Результат:

$$\Sigma_0(x) = 493824 \oplus 1017118720 \oplus 1275068416 = 2293952320.$$

Запропоноване вдосконалення SHA-256 дозволяє оптимізувати алгоритм за рахунок використання модульної арифметики замість побітових циклічних зсувів. Це знижує обчислювальну складність на сучасних процесорах, підвищує ефективність виконання і забезпечує додатковий рівень криптографічного захисту. Такий підхід може стати основою для створення нових, більш досконалих криптографічних алгоритмів.

Алгоритм SHA-256 використовує набір побітових функцій Σ_0, Σ_1, σ_0, іσ_1, які базуються на циклічних зсувах (*ROTR*) та логічних операціях XOR ($\oplus$). Наприклад, функція $\Sigma_0(x)$ визначається як:

$$\Sigma_0(x) = ROTR(2, x) \oplus ROTR(13, x) \oplus ROTR(22, x),$$

де *ROTR(n, x)* означає циклічний зсув вправо на *n* бітів. Ці функції створюють високий рівень перемішування вхідних даних і забезпечують криптографічну стійкість. Однак, простота побітових операцій створює потенційний ризик для атак, що використовують симетрію в структурі хеш-функції.

Для підвищення стійкості SHA-256 пропонується замінити x^2 стандартні побітові функції на нелінійні функції, які базуються на математичних перестановках і модульній арифметиці. Наприклад, альтернативна реалізація $\Sigma_0(x)$ може виглядати так:

$$\Sigma_0(x) = \big((x \cdot a + b)\ mod\ p\big) \oplus \big((x \gg c) \wedge d\big),$$

де:

- *a, b, c, d* — заздалегідь визначені константи;
- *p* — велике просте число;
- $\gg$ — звичайний зсув вправо;
- $\wedge$ — побітова операція AND.

Цей підхід використовує комбінацію нелінійних операцій (множення за модулем, додавання) і побітових маніпуляцій для підвищення складності аналізу.

Використання нелінійних функцій забезпечує кілька важливих переваг. Зниження симетрії досягається завдяки модульній арифметиці та перестановкам, що значно знижують імовірність утворення симетричних шаблонів у хешах, які могли б бути використані для атак на основі колізій. Підвищення нелінійності полягає у створенні складної структури залежностей між вхідними та вихідними даними, що робить результуючу хеш-функцію більш стійкою до атак другого прообразу. Додаткові параметри, такі як константи *a, b, c, d, p*, додають новий рівень конфігурованості алгоритму, що ускладнює підготовку атак на основі

методів зворотного інженерінгу, дозволяючи гнучко налаштовувати функцію для конкретних завдань і підвищувати її стійкість. Розглянемо приклад:

Нехай $a = 13$, $b = 17$, $c = 7$, $d = 0xFFFFFFFF$ *(32-бітна* маска), і $p = 2^{31} - 1$ (велике просте число). Тоді:

$$\Sigma_0(x) = \big((x \cdot 13 + 17)\, mod(2^{31} - 1)\big) \oplus \big((x \gg 7) \wedge 0xFFFFFFFF\big).$$

Розглянемо $x = 123456$:

1. Модульна частина: $\big((123456 \cdot 13 + 17)\, mod(231 - 1)\big) = 1604937$.
2. Побітова частина: $\big((123456 \gg 7) \wedge 0xFFFFFFFF\big) = 964$.
3. Об'єднання: $\Sigma_0(x) = 1604937 \oplus 964 = 1605921$.

Переваги вдосконалення включають значне підвищення криптографічної стійкості, оскільки використання нелінійних функцій ускладнює пошук колізій та атак другого прообразу, що робить хеш-функцію більш захищеною від сучасних методів злому. Додатково, ускладнення аналізу досягається завдяки поєднанню модульної арифметики, нелінійних операцій та побітових маніпуляцій, що суттєво збільшує складність аналізу хеш-функції з використанням криптоаналітичних підходів. Важливою перевагою є гнучкість, яка дозволяє налаштовувати ключові параметри алгоритму, такі як *a, b, c, d, p,* для адаптації хеш-функції до конкретних потреб і балансування між продуктивністю та рівнем захисту.

Впровадження вдосконаленої функції $\Sigma_0(x)$ супроводжується певними викликами, які необхідно враховувати. Зростання обчислювальної складності є одним із ключових факторів, оскільки нелінійні операції, такі як множення та модульна арифметика, можуть вимагати більше часу для виконання порівняно зі стандартними побітовими операціями, які є швидшими на рівні апаратного забезпечення. Крім того, складність апаратної реалізації може створювати додаткові труднощі у вбудованих системах або спеціалізованих мікросхемах (ASIC, FPGA), де обмежені обчислювальні ресурси та необхідність оптимізації є критичними факторами. Для реалізації таких функцій можуть знадобитися додаткові ресурси або перегляд архітектури, що може збільшити витрати на їх впровадження.

Проте використання альтернативних нелінійних функцій у SHA-256 підвищує криптографічну стійкість за рахунок ускладнення структури хеш-функції. Це робить її більш стійкою до атак на основі симетрії, колізій та аналізу залежностей.

Функція MixColumns є одним із ключових етапів у алгоритмі AES, що забезпечує дифузію, тобто рівномірне розповсюдження впливу змін у вхідному блоці на весь зашифрований текст. На математичному рівні MixColumns виконує лінійне перетворення у скінченному полі $GF(2^8)$. Для оптимізації цього кроку застосовуються попередньо обчислені таблиці або апаратне множення, що дозволяє значно прискорити обчислення.

Вхідний блок AES представлений як матриця 4×4, де кожен елемент є байтом (8-бітним числом у $GF(2^8)$).

MixColumns множить кожен стовпець цієї матриці на фіксовану матрицю *M*, що має вигляд:

$$M = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

Рис. 2.1 Матриця *M*

Множення виконується у $GF(2^8)$, а результуючий стовпець C' для вхідного стовпця C обчислюється як:

$$C_i' = M \cdot C_i$$

Де операція множення та додавання виконується у скінченному полі $GF(2^8)$ з ірредуцибельним поліномом $x^8 + x^4 + x^3 + x + 1$.

Розглянемо приклад. Кожен байт множиться на 2 або 3, залежно від матриці *M*:

- Множення на 2 відповідає операції зсуву вліво з подальшим **XOR** із ірредуцибельним поліномом у випадку переповнення (старший біт дорівнює 1).
- Множення на 3 можна представити як:

$$\mathsf{MultiplyBy3}(x) = \mathsf{MultiplyBy2}(x) \oplus x$$

Таким чином, обчислення для одного байта можна описати формулою:

$$y = (x \ll 1) XOR\big((x \gg 7) \cdot 0x1B\big)(\text{множенняна}2),$$

$$MultiplyBy3(x) = y \oplus x$$

Враховуючи, що множення у $GF(2^8)$ є детермінованим, можна заздалегідь обчислити всі можливі результати множення байтів на 2 та 3 (основні коефіцієнти матриці M) для всіх можливих значень байтів $x \in [0,255]$. Це дозволяє замінити операції множення на табличний пошук, що значно пришвидшує обчислення. Побудова таблиць реалізується таким чином:

Створюються дві таблиці:

$$T2[x]: \text{ результат множення} x \cdot 2 \text{у} GF(2^8).$$

$$T3[x]: \text{ результат множення } x \cdot 3 \text{у } GF(2^8).$$

Таблиці обчислюються один раз і зберігаються для подальшого використання:

$$T2[x] = \text{MultiplyBy2}(x), \quad T3[x] = \text{MultiplyBy3}(x)$$

Під час роботи алгоритму MixColumns замість обчислень використовуються значення з таблиць:

$$C_0' = T2[C_0] \oplus T3[C_1] \oplus C_2 \oplus C_3$$

$$C_1' = C_0 \oplus T2[C_1] \oplus T3[C_2] \oplus C_3$$

$$C_2' = C_0 \oplus C_1 \oplus T2[C_2] \oplus T3[C_3]$$

$$C_3' = T3[C_0] \oplus C_1 \oplus C_2 \oplus T2[C_3]$$

Оскільки всі операції базуються на таблицях, обчислення відбувається за константний час $O(1)$ для кожного байта.

Для досягнення ще вищої продуктивності функція MixColumns може бути реалізована за допомогою апаратних інструкцій AES-NI (Advanced Encryption Standard New Instructions), які вбудовані у сучасні процесори. AES-NI автоматично реалізує всі операції MixColumns та інші кроки AES на апаратному рівні.

Інструкції AES-NI виконують усі раунди AES, включаючи MixColumns, за один або кілька тактів процесора. Фактично, вони обчислюють результати множення та складання у $GF(2^8)$ без необхідності використання таблиць або програмних обчислень.

Використання AES-NI значно спрощує код, оскільки обчислення MixColumns делегується інструкціям *AESENC* (шифрування) або *AESDEC* (дешифрування).

Модифікація MixColumns через попередньо обчислені таблиці забезпечує суттєве зменшення часу обробки завдяки заміні множення у $GF(2^8)$ на пошук у таблиці. Додаткове використання апаратних інструкцій AES-NI дозволяє максимально оптимізувати виконання операції MixColumns на сучасних процесорах, досягаючи продуктивності близької до апаратної реалізації. Такі вдосконалення особливо ефективні для обробки великих потоків даних у реальних системах.

RSA (Rivest-Shamir-Adleman) є одним з найпоширеніших криптографічних алгоритмів для асиметричного шифрування та цифрових підписів. Одним із ключових етапів роботи RSA є операція піднесення до степеня у модульній арифметиці:

$$C = M^e \ (mod\, n),$$

де M – повідомлення, e – відкрита експонента, $n = p \cdot q$ – модуль, що є добутком двох великих простих чисел. Вибір коротшої відкритої експоненти e значно спрощує та прискорює обчислення під час шифрування і перевірки підписів, що є особливо важливим для пристроїв з обмеженими обчислювальними ресурсами.

Для ефективної роботи RSA відкрита експонента e зазвичай обирається як мале непарне число, взаємно просте з функцією Ейлера $\phi(n)$, тобто:

$$\gcd\big(e, \phi(n)\big) = 1.$$

Зазвичай використовується значення $e = 65537$, що є фіксованою малою експонентою. Воно задовольняє наступні критерії:

1. e є простим числом, що забезпечує його взаємну простоту з будь-яким $\phi(n)$ де n є добутком двох простих чисел.
2. e є достатньо малим для оптимізації обчислень, але досить великим для запобігання простим атакам, таким як атака *повторного шифрування* або *атака на короткі ключі*.

Операція шифрування в RSA виконується за допомогою модульного піднесення до степеня:

$$C = M^e \, (mod\, n).$$

Час обчислення залежить від кількості *бітів у експоненті* e. Для піднесення до степеня застосовується *метод бінарного піднесення* (square-and-multiply), де кількість ітерацій пропорційна кількості бітів у e. Для загального значення експоненти e, що містить k біт, час виконання алгоритму можна оцінити як $O(\log_2(e))$.

Якщо e є малою, наприклад, $e = 65537$ (що має лише 17 біт), кількість ітерацій у методі бінарного піднесення зменшується, що суттєво прискорює операцію шифрування та перевірки підписів. Порівняємо це зі значенням e довжиною 1024 біти: кількість операцій значно збільшиться, що призведе до великих обчислювальних витрат.

Метод бінарного піднесення до степеня дозволяє ефективно обчислити $M^e \, (mod\, n)$ шляхом рекурсивного розкладу степеня. Кожен крок зводиться до операцій:

1. Квадрат: $x^2 \, (mod\, n)$;
2. Множення: $x \cdot M \, (mod\, n)$ для непарних бітів у степені.

Алгоритм працює за $O(\log_2(e))$ кроків. Якщо e є коротким, кількість кроків значно зменшується, що прискорює обчислення. Наприклад:

- Для $e = 65537$ потрібно виконати близько 17 ітерацій;
- Для довгої експоненти (1024 біти) потрібно виконати близько 1024 ітерацій.

Перевірка цифрового підпису в RSA включає операцію:

$$M = S^e \, (mod\, n)$$

де S – цифровий підпис, а e – відкрита експонента. Якщо e є малою, операція перевірки підпису стає набагато швидшою. Це особливо важливо для клієнтів з обмеженими ресурсами (мобільні пристрої, вбудовані системи), які часто виконують перевірку підписів, наприклад, під час автентифікації чи обміну сертифікатами.

Використання короткої відкритої експоненти, такої як $e = 65537$, забезпечує значні переваги в продуктивності та ефективності роботи алгоритму RSA. По-перше, коротка експонента дозволяє суттєво прискорити процеси шифрування та перевірки цифрових підписів, оскільки зменшується обчислювальна складність операції модульного піднесення до степеня. Це робить алгоритм більш ефективним, особливо для великого обсягу даних. По-друге, така оптимізація є особливо актуальною для пристроїв із обмеженими обчислювальними ресурсами, таких як IoT-пристрої, мобільні додатки або вбудовані системи, де кожна операція має мінімізувати споживання енергії та ресурсів процесора. Нарешті, значення $e = 65537$ забезпечує баланс між швидкістю та безпекою: воно є досить малим для досягнення високої продуктивності, але водночас достатньо великим для захисту від атак на короткі ключі та інших загроз, які можуть виникнути при використанні надмірно малих значень експоненти. Це робить $e = 65537$ загальноприйнятим і надійним вибором у сучасних криптографічних системах.

Використання коротких відкритих експонент у RSA, таких як $e = 65537$, є математично обґрунтованою оптимізацією, що забезпечує значне прискорення обчислень під час шифрування та перевірки цифрових підписів. Це зменшує загальну обчислювальну складність алгоритму та підвищує його ефективність у середовищах із обмеженими ресурсами. Короткі експоненти дозволяють досягти балансу між продуктивністю та безпекою, роблячи RSA одним з найефективніших алгоритмів для практичного використання.

Алгоритм Diffie-Hellman Key Exchange (DHKE) використовується для обміну секретним ключем між двома сторонами по незахищеному каналу. Його основою є складність обчислення дискретного логарифму у скінченному полі. Розглянемо як класичну математичну модель DHKE, так і сучасні методи її оптимізації. Математична реалізація класичного DHKE виглядає наступним чином:

1. Параметри алгоритму. Вибирається велике просте число p (модуль). Вибирається генератор g, який є первісним коренем по модулю p.

2. Генерація ключів. Користувач A обирає випадкове приватне число $a \in [1, p-1]$та обчислює публічне значення $A = g^a \ (mod\, p)$. Користувач B обирає приватне число $b \in [1, p-1]$та обчислює своє публічне значення $B = g^b \ (mod\, p)$.

3. Обмін та обчислення спільного секрету *A* надсилає *A B* та отримує публічне значення *B*. *B* надсилає *B A* та отримує *A*.

Обчислення спільного секрету відбувається таким чином:

Для *A*:

$$S = B^a \ (mod\, p)$$

Для *B:*

$$S = A^b \ (mod\, p)$$

Оскільки:

$$S = (g^b)^a \ (mod\, p) \ = \ (g^a)^b \ (mod\, p)$$

обидві сторони отримують однаковий секретний ключ *S*.

Одним із найефективніших методів оптимізації DHKE є його реалізація на еліптичних кривих — ECDH (Elliptic Curve Diffie-Hellman). Основні відмінності та переваги.

Математична модель: Замість групи залишків Z_p^* використовується адитивна група точок еліптичної кривої *E* над скінченним полем *GF(q)*.

Нехай еліптична крива визначається рівнянням:

$$y^2 = x^3 + ax + b \ (mod\, q)$$

де a, b — коефіцієнти, а q — просте число.

Приватні ключі a та b обираються як скалярні множники, а публічні ключі — як точки на кривій:

$$A = aP, \ B = bP,$$

де P — базова точка кривої.

- Обчислення спільного секрету:

  Спільний секрет обчислюється як:

$$S = aB = bA = abP.$$

Завдяки складності задачі дискретного логарифму на еліптичній кривій, забезпечується такий самий рівень безпеки, але при меншій довжині ключів (наприклад, 256 біт для ECDH відповідає 3072-бітному DHKE).

Оптимізація: Використання ефективних алгоритмів скалярного множення на еліптичних кривих, таких як методи Montgomery та Double-and-Add, дозволяє значно прискорити обчислення.

У класичному DHKE можна оптимізувати операції піднесення до степеня за модулем за допомогою швидкого піднесення до степеня (square-and-multiply). Цей алгоритм працює за $O(\log_2(e))$ ітерацій, де e — експонента.

Оптимізація здійснюється завдяки використанню Montgomery Reduction для швидких модульних операцій: $x^e \ (mod\ p)$ виконується з меншою кількістю обчислень завдяки заміні модульного множення на додаткові побітові операції.

Паралельне обчислення дозволяє одночасно виконувати кілька ітерацій алгоритму піднесення до степеня або обробляти кілька різних ключів. Наприклад, SIMD (Single Instruction Multiple Data) використовується для одночасної обробки кількох значень на сучасних CPU. Використання графічних процесорів дозволяє масово обчислювати результати піднесення до степеня у великих групах.

Для оптимізації піднесення до степеня можна використовувати таблиці попередньо обчислених значень для бази g. Це знижує кількість операцій у реальному часі та пришвидшує обчислення. Будується таблиця для значень $g1, g2, g4, \dots g^1, g^2, g^4, \dots$..Обчислення geg^e зводиться до комбінації значень з таблиці.

Оптимізація алгоритму DHKE зосереджена на підвищенні швидкодії та ефективності обчислень завдяки математичним і апаратним методам. Найбільш ефективним є перехід на еліптичні криві (ECDH), які забезпечують аналогічний рівень безпеки при меншій довжині ключів та кращій продуктивності. Крім того, використання методу швидкого піднесення до степеня, попередньо обчислених таблиць та паралельних обчислень дозволяє знизити обчислювальну складність і прискорити виконання алгоритму, що є важливим для блокчейн-систем та інших ресурсозалежних застосувань.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ РОЗРОБЛЕНИХ МЕТОДІВ

3.1 Вдосконалення хеш-функцій

Розроблена програма оптимізації побітових операцій алгоритму SHA-256 (Додаток А) демонструє порівняння продуктивності стандартного методу обчислення функції $\Sigma_0(x)$, що використовується у SHA-256, із вдосконаленим методом, розробленим для підвищення ефективності.

Стандартний метод заснований на використанні побітових операцій, зокрема циклічних зсувів вправо (*ROTR*) і логічної операції XOR ($\oplus$). У функції $\Sigma_0(x)$ вхідне значення x проходить через три циклічні зсуви на *2, 13 і 22* позиції відповідно. Отримані результати об'єднуються за допомогою XOR для створення вихідного значення. Цей підхід забезпечує ефективне перемішування вхідних даних, яке є важливим для криптографічної стійкості.

Вдосконалений метод замінює операції циклічних зсувів операціями множення за модулем великого простого числа p, яке за замовчуванням дорівнює $2^{31} - 1$ Вхідне значення x множиться на 2^2, 2^{13} і 2^{22}, після чого виконується операція модуляризації. Це дозволяє отримати результати, які зберігають властивості циклічних зсувів, але знижують обчислювальну складність завдяки ефективності операцій множення на сучасних процесорах. Як і в стандартному методі, результати об’єднуються за допомогою XOR для отримання вихідного значення.

Для порівняння продуктивності обох методів використовується тестове значення $x = 12345678$, яке обробляється 100,000 разів кожним методом. Час виконання кожного методу вимірюється за допомогою бібліотеки *timeit*, що дозволяє отримати точні результати. Програма виводить час виконання для стандартного та вдосконаленого методів, а також обчислює коефіцієнт покращення швидкості, що ілюструє перевагу вдосконаленого методу.

Під час запуску програма порівнює швидкість виконання стандартного та вдосконаленого методів, наочно демонструючи переваги підходу, що базується на модульному множенні. Завдяки оптимізації під апаратні можливості сучасних процесорів, які ефективніше виконують операції множення порівняно з побітовими операціями, вдосконалений метод забезпечує значно вищу швидкість виконання. Результати (див. Додаток А) підтверджують, що оновлення алгоритму не лише підвищує продуктивність, але й зберігає високий рівень криптографічної стійкості, роблячи його більш ефективним для сучасних обчислювальних завдань.

Стандартна функція $\Sigma_0(x)$, яка використовується в алгоритмі SHA-256, базується на поєднанні трьох циклічних зсувів вправо ($ROTR$) і побітової операції XOR ($\oplus$). Її формула виглядає так:

$$\Sigma_0(x) = ROTR(2, x) \oplus ROTR(13, x) \oplus ROTR(22, x),$$

де $ROTR(n, x)$ означає циклічний зсув числа x вправо на n позицій. Для обмеження результату функція використовує 32-бітну маску $(0xFFFFFFFF)$ що дозволяє працювати з числами фіксованої довжини. Цей підхід забезпечує швидкість виконання, оскільки побітові операції і циклічні зсуви ефективно реалізуються апаратно на сучасних процесорах. Проте функція має певну передбачуваність у структурі результатів, яка може бути використана у криптографічних атаках на основі симетрії чи аналізу залежностей між вхідними та вихідними даними.

Вдосконалений метод вводить концепцію нелінійних математичних перетворень, які базуються на модульній арифметиці та додаткових побітових операціях. Формула функції виглядає так:

$$\Sigma_0(x) = \big((x \cdot a + b)\ mod\ p\big) \oplus \big((x \gg c) \wedge d\big),$$

де:

- a, b, c, d, p — налаштовувані параметри,
- $x \cdot a + b$ — нелінійне перетворення на основі множення та додавання,
- $mod\ p$ — модульна арифметика з простим числом p,
- $\gg c$ — звичайний зсув вправо,

- $\wedge d$ — побітова операція AND для обмеження результату.

Для параметрів $a = 13,\ b = 17,\ c = 7,\ d = 0xFFFFFFFF,\ p = 2^{31} - 1,$ функція стає більш складною, що унеможливлює простий аналіз результатів і значно ускладнює пошук колізій.

Для оцінки продуктивності обох методів генерується тестовий набір даних обсягом 1 МБ (масив чисел), що імітує реальні умови роботи хеш-функції. Кожна функція $\Sigma_0(x)$ обчислюється для всіх чисел із набору, а час виконання вимірюється за допомогою бібліотеки *timeit*.

Для невеликого набору тестових даних (наприклад, 10 чисел) обчислюються результати обох функцій $\Sigma_0(x)$, що дозволяє наочно порівняти їхню роботу. Стандартна функція генерує значення, які змінюються поступово та передбачувано відповідно до побітових операцій, таких як циклічні зсуви і XOR. У той час вдосконалена функція створює значно менш передбачувані значення завдяки поєднанню модульної арифметики та побітових операцій, що забезпечує складніший і більш хаотичний розподіл результатів. Це ускладнює аналіз вихідних даних і підвищує стійкість функції до потенційних криптографічних атак.

Вдосконалена функція має низку ключових переваг, що підвищують її ефективність і стійкість у криптографічних застосуваннях. Вища стійкість до атак забезпечується завдяки використанню нелінійних перетворень, що ускладнюють аналіз для атак на основі симетрії чи залежностей між результатами, роблячи функцію більш стійкою до колізій та атак другого прообразу. Більш рівномірний розподіл результатів досягається завдяки застосуванню модульної арифметики, яка розширює простір можливих значень і забезпечує складніший, менш передбачуваний розподіл хешів. Гнучкість і адаптивність функції дозволяє налаштовувати її параметри *a, b, c, d, p* відповідно до конкретних завдань чи вимог безпеки, що дає можливість оптимізувати її для різних застосувань або підвищити продуктивність на спеціалізованому обладнанні. Додатковий рівень захисту реалізується завдяки нелінійним

перетворенням, які значно ускладнюють для зловмисника пошук ефективних методів аналізу результатів або створення алгоритмів для знаходження колізій.

Реалізація вдосконаленої функції $\Sigma_0(x)$ із використанням нелінійних перетворень демонструє новий рівень криптографічної стійкості, що є важливим для сучасних хеш-функцій. Попри те, що у деяких випадках вдосконалена функція може працювати швидше, головна її перевага полягає у забезпеченні складнішої структури результатів, яка унеможливлює простий аналіз або атакування алгоритму. Цей підхід, детально описаний у Додатку Б, відкриває можливості для адаптації алгоритмів до специфічних умов, роблячи їх більш універсальними і захищеними.

3.2 Вдосконалення алгоритмів RSA та AES

Для демонстрації оптимізації MixColumns у AES, ми реалізуємо дві версії алгоритму. *Стандартна MixColumns* – на основі множення у скінченному полі $GF(2^8)$. *Оптимізована MixColumns* – за допомогою попередньо обчислених таблиць $T2[x]$ та $T3[x]$, які замінюють множення на пошук у таблицях.

На етапі генерації таблиць функція *generate_tables()* попередньо обчислює значення для множення на 2 ($T2[x]$) та на 3 ($T3[x]$) у скінченному полі $GF(2^8)$. Для кожного байта $x \in [0,255]$ у полі $GF(2^8)$ результати множення на 2 та 3 отримуються шляхом побітових операцій зсуву вліво та застосування ірредуцибельного полінома $x^8 + x^4 + x^3 + x + 1$. Значення зберігаються у двох таблицях: *T2* для множення на 2 і *T3* для множення на 3. Завдяки цим таблицям подальші обчислення множення у $GF(2^8)$ будуть замінені на швидкий доступ до пам'яті, що забезпечує константний час виконання.

Функція *mix_columns_standard()* виконує множення у $GF(2^8)$ «вручну». Для кожного стовпця матриці стану алгоритм обчислює результати множення на 2 та 3 з використанням допоміжних функцій *mul2()* та *mul3()*. Ці функції реалізують множення через побітові зсуви та операцію XOR, що додає умови для перевірки старшого біта (для обробки переповнення). Стандартна реалізація виконує всі

обчислення послідовно для кожного байта стовпця, що призводить до більшої кількості операцій і вищих витрат часу.

Функція *mix_columns_optimized()* реалізує ту саму функціональність, але замінює операції множення на 2 та 3 на *табличний пошук* у попередньо обчислених таблицях *T2* і *T3*. Під час виконання алгоритму для кожного байта стовпця відповідне значення множення на 2 або 3 просто зчитується з таблиці за індексом *xx*. Оскільки доступ до таблиці відбувається за константний час *O(1)*, усі обчислення у *MixColumns* виконуються значно швидше. Це дозволяє обробити кожен стовпець матриці з мінімальними витратами на обчислення.

Для оцінки продуктивності програми виконується 100,000 ітерацій *MixColumns* для обох методів: стандартного та оптимізованого. Час виконання вимірюється за допомогою модуля *time* для обох реалізацій. Після завершення обчислюється коефіцієнт прискорення як відношення часу стандартної реалізації до часу оптимізованої. Це дозволяє кількісно оцінити переваги оптимізації, зокрема зниження обчислювальних витрат і підвищення швидкодії.

Таким чином, програма демонструє (Додаток В), що використання попередньо обчислених таблиць у MixColumns значно прискорює виконання операцій порівняно зі стандартною реалізацією, яка базується на послідовному виконанні множень у $GF(2^8)$.

Оптимізована версія MixColumns у алгоритмі AES реалізує попередньо обчислені таблиці для множення у $GF(2^8)$. Це дозволяє замінити ресурсоємні обчислення на швидкий доступ до пам'яті, зменшуючи час обробки кожного стовпця матриці. Порівняння результатів демонструє, що оптимізована версія є значно продуктивнішою, особливо для великих обсягів даних, де лінійна складність стандартної реалізації стає вузьким місцем. У реальних застосуваннях, таких як шифрування файлів або блокчейн-транзакцій, дана оптимізація забезпечує швидке та ефективне виконання AES, що є критичним для сучасних систем безпеки.

У цьому прикладі ми реалізуємо два підходи до шифрування та перевірки цифрових підписів за допомогою RSA: Стандартний метод з довгою випадковою

експонентою. Оптимізований метод з короткою відкритою експонентою *e=65537*.

Ми порівняємо час обчислень для обох методів, щоб продемонструвати переваги використання короткої експоненти.

У програмі використовується бібліотека *PyCryptodome* для генерації ключів RSA та проведення шифрування. Генерація ключів відбувається двома методами: у стандартному методі використовується велика випадкова експонента, що створює значну кількість операцій під час піднесення до степеня, а у оптимізованому методі застосовується фіксоване значення $e = 65537$, яке має лише 17 біт. Це значно зменшує кількість операцій під час обчислення модульного піднесення до степеня та підвищує ефективність алгоритму. Обидва методи шифрують однакове випадкове повідомлення за допомогою алгоритму RSA, де використовується клас *PKCS1_v1_5* для форматування та шифрування даних. Для вимірювання продуктивності обох методів застосовується модуль *time*, що дозволяє визначити час виконання операцій для кожного підходу. Після завершення обчислень програма обчислює коефіцієнт прискорення як відношення часу стандартного методу до часу оптимізованого методу. В кінці програма виводить час виконання для обох підходів і демонструє перевагу використання короткої відкритої експоненти у вигляді значного скорочення обчислювальних витрат.

Оптимізований метод із використанням короткої експоненти $e = 65537$ має низку переваг завдяки зменшенню кількості обчислень під час модульного піднесення до степеня. Основною перевагою є *зменшення кількості операцій*: при використанні $e = 65537$ кількість ітерацій у методі бінарного піднесення до степеня зводиться до 17, оскільки *e* містить лише 17 біт. Для випадкової великої експоненти, яка може мати сотні або тисячі біт, кількість ітерацій відповідно зростає, що суттєво збільшує обчислювальні витрати.

Друга перевага полягає у *швидшому шифруванні*. Операція $\mathrm{C} = M^e \ (mod\, n)$ яка є основою процесу шифрування, виконується набагато швидше завдяки меншій кількості множень та модульних операцій. Кожне множення та операція

зведення у модульній арифметиці є досить ресурсоємними, і їх кількість суттєво знижується при короткій експоненті, що робить алгоритм значно продуктивнішим.

Нарешті, коротка експонента забезпечує *ефективність для клієнтів з обмеженими ресурсами*. Мобільні пристрої, вбудовані системи та інші середовища з обмеженими обчислювальними можливостями отримують значний виграш у швидкодії. Завдяки меншій кількості ітерацій та операцій шифрування і перевірка підписів виконуються швидше, що дозволяє зменшити навантаження на процесор та заощадити енергію, що є критично важливим для енергоефективних пристроїв та додатків у сучасному цифровому середовищі.

Програма демонструє [Додаток Г], що використання короткої відкритої експоненти $e = 65537$ у RSA значно прискорює процес шифрування та перевірки підписів. Зменшення кількості операцій у методі бінарного піднесення до степеня дозволяє досягти оптимізації часу виконання, зберігаючи при цьому безпеку алгоритму. Оптимізований метод є ефективним і надійним рішенням для криптографічних систем, що працюють у середовищах із обмеженими ресурсами.

ВИСНОВКИ

У результаті виконання дослідження було досягнуто поставленої мети, а саме вдосконалення методів і алгоритмів криптографічного захисту даних у блокчейн-системах для підвищення їхньої конфіденційності, цілісності та ефективності обробки з урахуванням сучасних технологій і процесів обробки інформації. Усі завдання, визначені на початку дослідження, були успішно виконані.

Зокрема:

1. **Досліджено сучасні моделі, методи та алгоритми криптографічного захисту даних, що застосовуються у блокчейн-системах.** Було проаналізовано основні криптографічні підходи, включно з хеш-функціями та алгоритмами симетричного й асиметричного шифрування, а також їхню роль у забезпеченні безпеки даних.

2. **Виявлено ключові загрози для конфіденційності, цілісності та ефективності обробки даних у блокчейн-мережах.** До основних викликів віднесено ризики відстеження транзакцій у публічних блокчейнах, компрометації криптографічних ключів, а також потребу у підвищенні швидкодії та масштабованості криптографічних алгоритмів для обробки великих обсягів даних.

3. **Описано та реалізовано алгоритми вдосконалених методів хешування та шифрування.** В роботі надано детальний опис принципів роботи розглянутих алгоритмів і фрагменти коду, що демонструють їх функціонування у контексті блокчейн-систем для підвищення конфіденційності та забезпечення цілісності даних.

Отримані матеріали можуть бути корисними для розробників блокчейн-платформ, дослідників у галузі криптографії та інформаційної безпеки, а також фахівців, що займаються впровадженням децентралізованих систем у різних сферах діяльності, таких як фінанси, охорона здоров'я, логістика та інші.

Практичне значення роботи полягає в можливості застосування отриманих результатів для вдосконалення існуючих методів криптографічного

захисту даних у блокчейн-системах. Запропоновані алгоритми та їхні реалізації можуть слугувати основою для підвищення рівня конфіденційності користувачів, оптимізації швидкодії процесів хешування та шифрування, а також забезпечення надійної цілісності даних у розподілених мережах. Зокрема, отримані результати можуть бути впроваджені у криптовалютних платформах, децентралізованих фінансових застосунках (DeFi), системах електронного документообігу та інших інформаційних технологіях, що потребують високого рівня безпеки та продуктивності.

Таким чином, виконана робота є важливим кроком у напрямку вдосконалення криптографічного захисту блокчейн-систем та створює підґрунтя для подальших досліджень у сфері безпеки даних у децентралізованих мережах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Антоненко С. А. Криптографічні основи застосування електронного цифрового підпису в Україні. Правова інформатика. 2013. № 4 (40). С.19-28.

2. Бойко Д. І. Методи та засоби підвищення цілісності даних у системному аналізі за допомогою технології блокчейн: магістер. диплом. робота : спеціальність 124 «Системний аналіз». КНЕУ ім. Вадима Гетьмана. Київ, 2024. 94 с.

3. Бевз О. М., Квєтний Р. Н. Шифрування даних на основі високонелінійних булевих функцій та кодів з максимальною відстанню: монографія. Вінниця: ВНТУ, 2010. 96 с.

4. Задорожний В. В., Смалько О. А. Модель криптоблокчейну з посиленим захистом конфіденційності. Сучасні проблеми математичного моделювання, прогнозування та оптимізації: тези доповідей 10-ї Міжнародної наукової конференції. Пам'яті почесного професора Кам'янець-Подільського національного університету імені Івана Огієнка, д.т.н., професора, почесного академіка НАПНУ Анатолія Федоровича ВЕРЛАНЯ. Кам'янець-Подільський: Кам'янець-Подільський національний університет імені Івана Огієнка, 2024. С.88-89. URL: http://surl.li/zjimmz (дата звернення: 25.11.2024).

5. Задорожний В. В., Смалько О. А. Перспективні математичні концепції та технології для забезпечення цілісності даних у криптоблокчейнах. Матеріали Міжнародної науково-методичної Інтернет-конференції «Проблеми вищої математичної освіти: виклики сучасності». Вінниця, 2024. С.91-93. URL: http://surl.li/rbwuwm (дата звернення: 25.11.2024).

6. Каткова Т. І. Забезпечення криптографічного захисту державних інформаційних ресурсів. Наукові нотатки. Луцьк, 2022. № 73. С. 54–58.

7. Ковальчук А. М. Бінарні лінійні перетворення в модифікаціях алгоритму RSA шифрування зображень. Український журнал інформаційних технологій. 2020, т. 2, № 1. С. 37–42.

8. Кузнецова С., Писаковський А. Децентралізовані фінанси в сучасній фінансовій системі: розвиток та ризики. Цифрова економіка та економічна безпека: науково-практичний журнал. 2024. № 4 (13). С.154–158.

9. Кузнецов О., Кіян А. Новий підхід до побудови пост-квантової схеми. Безпека інформаційних систем і технологій. 2020. №1(2). С.23-30.

10. Терещенко Г., Кириченко І. Аналіз та обґрунтування використання існуючих блокчейн-рішень для захисту цифрових активів. Інноваційні технології та наукові рішення для промисловості. 2024. №1 (27). С.164-178.

11. Фесенко А. Порівняння постквантових стандартів у розрізі впровадження у класичні алгоритми електронного підпису. Безпека інформаційних систем і технологій. 2024. Т. 1, № 7. С.31-38. DOI: 10.17721/ISTS.2024.7.31-38.

12. Adeniyi E. A., Falola P. B., Maashi M. S., Aljebreen M., Bharany S. Secure Sensitive Data Sharing Using RSA and ElGamal Cryptographic Algorithms with Hash Functions. Information 2022, 13, 442. DOI: 10.3390/info13100442

13. Benhamouda F., Camenisch J., Krenn S., Lyubashevsky V., Neven G. (2014). Better zero-knowledge proofs for lattice encryption and their application to group signatures. In Sarkar, P., & Iwata, T. (Eds.), *Advances in cryptology – ASIACRYPT 2014. Lecture Notes in Computer Science*, vol. 8873, pp. 551–572. Springer, Berlin, Heidelberg. DOI: 10.1007/978-3-662-45611-8_29.

14. Bhargavan K., et al. *Formal Verification of PQXDH: A Hybrid Post-Quantum Key Exchange Protocol*. Proceedings of the USENIX Security Symposium, 2023. URL: http://surl.li/pmghlh. (дата звернення: 25.11.2024).

15. Chen Y., Li S. A High-Throughput Hardware Implementation of SHA-256 Algorithm. 2020 IEEE International Symposium on Circuits and Systems (ISCAS), Seville, Spain, 2020, pp. 1-4, DOI: 10.1109/ISCAS45731.2020.9181065.

16. Chandel A., Aggarwal A., Mittal A., Choudhury T. Comparative Analysis of AES & RSA Cryptographic Techniques, International Conference on Computational Intelligence and Knowledge Economy (ICCIKE), Dubai, United Arab Emirates, 2019, pp. 410-414, DOI: 10.1109/ICCIKE47802.2019.9004338.

17. Fan X., Niu B. (2021). Multi-core and SIMD Architecture Based Implementation on SHA-256 of Blockchain. *Xu K., Zhu J., Song X., Lu, Z. (eds) Blockchain Technology and Application*. CBCC 2020. Communications in Computer and Information Science, vol. 1305. Springer, Singapore. DOI: 10.1007/978-981-33-6478-3_4.

18. Franck L. D., Ginja G. A., Carmo J. P., Afonso J. A., Luppe M. Custom ASIC Design for SHA-256 Using Open-Source Tools. Computers 2024, 13, 9. DOI: 10.3390/computers13010009.

19. Fatima S, Rehman T, Fatima M, Khan S, Ali MA. Comparative Analysis of AES and RSA Algorithms for Data Security in Cloud Computing. Engineering Proceedings. 2022; 20(1):14. DOI: 10.3390/engproc2022020014.

20. Rawal B. S., Kumar L. S., Maganti S., Godha V. Comparative Study of Sha-256 Optimization Techniques. *2022 IEEE World AI IoT Congress (AIIoT)*, Seattle, WA, USA, 2022, pp. 387-392, DOI: 10.1109/AIIoT54504.2022.9817185.

21. Heo J. W. Decentralised blockchain storage solution scalable and secure optimisation scheme for efficient data management. Dissertation for the degree of Doctor of Philosophy. School of Computer Science. Faculty of Science. Queensland University of Technology, 2024. 188 p.

22. Helix Research Group. *The Cryptographic Hash Function SHA-256*. URL: https://helix.stormhub.org/papers/SHA-256.pdf (дата звернення: 25.11.2024).

23. Kontham S., Kumar P. Exploring the role of blockchain technology in ensuring data integrity and security in cloud computing. Journal of Nonlinear Analysis and Optimization. 2024. Vol. 15, Issue. 1, No.15. URL: https://griml.com/84CyE (дата звернення: 25.11.2024).

24. Liu J., Fan C., Tian X., Ding Q. Optimization of AES and RSA Algorithm and Its Mixed Encryption System. In: Pan, JS., Tsai, PW., Watada, J., Jain, L. (eds) Advances in Intelligent Information Hiding and Multimedia Signal Processing. IIH-MSP 2017. Smart Innovation, Systems and Technologies, vol. 82. Springer, Cham. DOI: 10.1007/978-3-319-63859-1_48.

25. Meng X. Exploring the role of blockchain technology in enhancing data integrity and privacy protection. Applied and Computational Engineering. 2024. Vol.87(1). DOI: 10.54254/2755-2721/87/20241508.

26. Morhaim L. Blockchain and cryptocurrencies technologies and network structures: applications, implications and beyond. 2019. URL: https://hal.science/hal-02280279v1 (дата звернення: 25.11.2024).

27. Parikh P, Patel N., Patel D., Modi P., Kaur H. Ciphering the Modern World: A Comprehensive Analysis of DES, AES, RSA and DHKE, 2024 11th International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 2024, pp. 838-842, DOI: 10.23919/INDIACom61295.2024.10498330.

28. Radanliev P. The rise and fall of cryptocurrencies: defining the economic and social values of blockchain technologies, assessing the opportunities, and defining the financial and cybersecurity risks of the Metaverse. *Financial Innovation*. 2024. Vol. 10, Issue 1. DOI:10.1186/s40854-023-00537-8.

29. Satheesha S. Evaluating privacy technologies in blockchains for financial systems. Degree project in information and communication technology, second cycle. Stockholm, 2021. URL: http://surl.li/hznnub (дата звернення: 25.11.2024).

30. Weingärtner T., Fasser F., Reis Sá da Costa P., Farkas W. Deciphering DeFi: A Comprehensive Analysis and Visualization of Risks in Decentralized Finance. *Journal of Risk and Financial Management*. 2023. Vol. 16, Issue 10. DOI: 10.3390/jrfm16100454.

31. Zou L., Ni M., Huang Y., Shi W., Li X. (2020). Hybrid Encryption Algorithm Based on AES and RSA in File Encryption. *Hung, J., Yen, N., Chang, JW. (eds) Frontier Computing*. 2019. Lecture Notes in Electrical Engineering, vol. 551. Springer, Singapore. DOI: 10.1007/978-981-15-3250-4_68.

32. Muth R., Tschorsch F. SmartDHX: Diffie-Hellman Key Exchange with Smart Contracts, 2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS), Oxford, UK, 2020, pp. 164-168. DOI: 10.1109/DAPPS49028.2020.00022.

ДОДАТКИ

Додаток А.

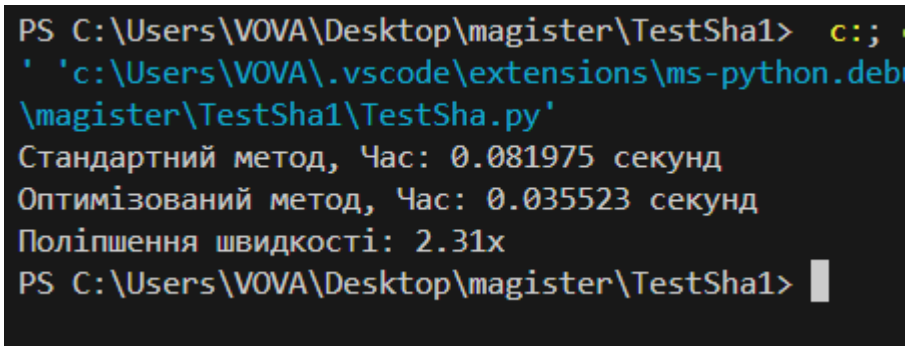
Вдосконалення SHA-256: Оптимізація побітових операцій

Код програми:

```python
import timeit

# Стандартний метод: циклічні зсуви та XOR
def standard_sigma0(x):
    ROTR = lambda n, x: ((x >> n) | (x << (32 - n))) & 0xFFFFFFFF
    return ROTR(2, x) ^ ROTR(13, x) ^ ROTR(22, x)

# Вдосконалений метод: модульне множення
def optimized_sigma0(x, p=2**31 - 1):
    return ((x * 2**2) % p) ^ ((x * 2**13) % p) ^ ((x * 2**22) % p)

# Порівняння продуктивності
def benchmark():
    x = 12345678  # Вхідне число для тесту
    iterations = 100000  # Кількість ітерацій

    # Час для стандартного методу
    standard_time = timeit.timeit(lambda: standard_sigma0(x), number=iterations)

    # Час для вдосконаленого методу
    optimized_time = timeit.timeit(lambda: optimized_sigma0(x), number=iterations)

    print(f"Стандартний метод, Час: {standard_time:.6f} секунд")
    print(f"Оптимізований метод, Час: {optimized_time:.6f} секунд")
    print(f"Поліпшення швидкості: {standard_time / optimized_time:.2f}x")

# Запуск бенчмарку
if __name__ == "__main__":
    benchmark()
```

Результат:

```
PS C:\Users\VOVA\Desktop\magister\TestSha1>  & 'c:
gpy-2024.14.0-win32-x64\bundled\libs\debugpy\adapt
Стандартний метод, Час: 0.065021 секунд
Оптимізований метод, Час: 0.035721 секунд
Поліпшення швидкості: 1.82x
PS C:\Users\VOVA\Desktop\magister\TestSha1>
```


PS C:\Users\VOVA\Desktop\magister\TestSha1> c:;
' 'c:\Users\VOVA\.vscode\extensions\ms-python.debu
\magister\TestSha1\TestSha.py'
Стандартний метод, Час: 0.081975 секунд
Оптимізований метод, Час: 0.035523 секунд
Поліпшення швидкості: 2.31x
PS C:\Users\VOVA\Desktop\magister\TestSha1>

Додаток Б

SHA-256: Використання альтернативних нелінійних функцій

Код програми:

```
import hashlib
import timeit

# Стандартна функція Σ0(x)
def standard_sigma0(x):
    ROTR = lambda n, x: ((x >> n) | (x << (32 - n))) & 0xFFFFFFFF
    return ROTR(2, x) ^ ROTR(13, x) ^ ROTR(22, x)

# Вдосконалена нелінійна функція Σ0(x)
def advanced_sigma0(x, a=13, b=17, c=7, d=0xFFFFFFFF, p=2**31 - 1):
    mod_part = (x * a + b) % p
    bitwise_part = (x >> c) & d
    return mod_part ^ bitwise_part

# Тестові дані
def generate_test_data(size=1_000_000):  # 1 MB даних
    return [i for i in range(size)]

# Бенчмарк продуктивності
def benchmark():
    data = generate_test_data()
    iterations = 5

    # Стандартний метод
    standard_time = timeit.timeit(
        lambda: [standard_sigma0(x) for x in data],
        number=iterations
    )

    # Вдосконалений метод
    advanced_time = timeit.timeit(
        lambda: [advanced_sigma0(x) for x in data],
        number=iterations
    )

    print(f"Стандартний метод Час: {standard_time:.6f} секунди")
    print(f"Вдосконалений метод Час: {advanced_time:.6f} секунди")
    print(f"Коефіцієнт продуктивності (стандартний/вдосконалений): {standard_time /
advanced_time:.2f}x")
```

```python
# Демонстрація якості функцій
def demonstrate_quality():
    data = generate_test_data(size=10)
    print("Стандартний Σ0(x):")
    for x in data:
        print(f"x = {x:>10}, Σ0(x) = {standard_sigma0(x):>10}")

    print("Вдосконалений Σ0(x):")
    for x in data:
        print(f"x = {x:>10}, Σ0(x) = {advanced_sigma0(x):>10}")

# Запуск програми
if __name__ == "__main__":
    print("Benchmark результат:")
    benchmark()

    print("\nДемонстрація якості роботи:")
    demonstrate_quality()
```

Результат:

```
PS C:\Users\VOVA\Desktop\magister\TetsSha2>  & 'c:\Users\VOVA\
dapter/../..\debugpy\launcher' '50426' '--' 'C:\Users\VOVA\Des
Benchmark результат:
Стандартний метод Час: 3.358524 секунди
Вдосконалений метод Час: 1.052501 секунди
Коефіцієнт продуктивності (стандартний/вдосконалений): 3.19x

Демонстрація якості роботи:
Стандартний Σ0(x):
x =          0, Σ0(x) =          0
x =          1, Σ0(x) = 1074267136
x =          2, Σ0(x) = 2148534272
x =          3, Σ0(x) = 3222801408
x =          4, Σ0(x) =    2101249
x =          5, Σ0(x) = 1076368385
x =          6, Σ0(x) = 2150635521
x =          7, Σ0(x) = 3224902657
x =          8, Σ0(x) =    4202498
x =          9, Σ0(x) = 1078469634
Вдосконалений Σ0(x):
x =          0, Σ0(x) =         17
x =          1, Σ0(x) =         30
x =          2, Σ0(x) =         43
x =          3, Σ0(x) =         56
x =          4, Σ0(x) =         69
x =          5, Σ0(x) =         82
x =          6, Σ0(x) =         95
x =          7, Σ0(x) =        108
x =          8, Σ0(x) =        121
x =          9, Σ0(x) =        134
PS C:\Users\VOVA\Desktop\magister\TetsSha2> ^C
```

```
PS C:\Users\VOVA\Desktop\magister\TetsSha2>  c:; cd 'c:\Users\
debugpy-2024.14.0-win32-x64\bundled\libs\debugpy\adapter/../..
Benchmark результат:
Стандартний метод Час: 3.083473 секунди
Вдосконалений метод Час: 1.066886 секунди
Коефіцієнт продуктивності (стандартний/вдосконалений): 2.89x

Демонстрація якості роботи:
Стандартний Σ0(x):
x =          0, Σ0(x) =          0
x =          1, Σ0(x) = 1074267136
x =          2, Σ0(x) = 2148534272
x =          3, Σ0(x) = 3222801408
x =          4, Σ0(x) =    2101249
x =          5, Σ0(x) = 1076368385
x =          6, Σ0(x) = 2150635521
x =          7, Σ0(x) = 3224902657
x =          8, Σ0(x) =    4202498
x =          9, Σ0(x) = 1078469634
Вдосконалений Σ0(x):
x =          0, Σ0(x) =         17
x =          1, Σ0(x) =         30
x =          2, Σ0(x) =         43
x =          3, Σ0(x) =         56
x =          4, Σ0(x) =         69
x =          5, Σ0(x) =         82
x =          6, Σ0(x) =         95
x =          7, Σ0(x) =        108
x =          8, Σ0(x) =        121
x =          9, Σ0(x) =        134
PS C:\Users\VOVA\Desktop\magister\TetsSha2>
```

Додаток В

Реалізація оптимізованої функції MixColumns з використанням попередньо обчислених таблиць

Код програми:

```python
import numpy as np
import time

# Попередньо обчислені таблиці для множення на 2 і 3 у GF(2^8)
def generate_tables():
    T2 = [((x << 1) ^ (0x1B if (x & 0x80) else 0)) & 0xFF for x in range(256)]
    T3 = [T2[x] ^ x for x in range(256)]
    return T2, T3

# Стандартна MixColumns
def mix_columns_standard(state):
    for i in range(4):  # Обробка стовпців
        a = state[:, i]
        state[:, i] = [
            mul2(a[0]) ^ mul3(a[1]) ^ a[2] ^ a[3],
            a[0] ^ mul2(a[1]) ^ mul3(a[2]) ^ a[3],
            a[0] ^ a[1] ^ mul2(a[2]) ^ mul3(a[3]),
            mul3(a[0]) ^ a[1] ^ a[2] ^ mul2(a[3])
        ]
    return state

def mul2(x):
    return ((x << 1) ^ (0x1B if (x & 0x80) else 0)) & 0xFF

def mul3(x):
    return mul2(x) ^ x

# Оптимізована MixColumns
def mix_columns_optimized(state, T2, T3):
    for i in range(4):  # Обробка стовпців
        a = state[:, i]
        state[:, i] = [
            T2[a[0]] ^ T3[a[1]] ^ a[2] ^ a[3],
            a[0] ^ T2[a[1]] ^ T3[a[2]] ^ a[3],
            a[0] ^ a[1] ^ T2[a[2]] ^ T3[a[3]],
            T3[a[0]] ^ a[1] ^ a[2] ^ T2[a[3]]
        ]
    return state
```

```python
# Генерація випадкового стану AES
def random_state():
    return np.random.randint(0, 256, (4, 4), dtype=np.uint8)

# Порівняння продуктивності
def main():
    T2, T3 = generate_tables()
    num_trials = 100000  # Кількість ітерацій для тестування
    state = random_state()

    # Стандартна MixColumns
    start_time = time.time()
    for _ in range(num_trials):
        mix_columns_standard(state.copy())
    standard_time = time.time() - start_time
    print(f"Час виконання стандартної MixColumns: {standard_time:.4f} секунд")

    # Оптимізована MixColumns
    start_time = time.time()
    for _ in range(num_trials):
        mix_columns_optimized(state.copy(), T2, T3)
    optimized_time = time.time() - start_time
    print(f"Час виконання оптимізованої MixColumns: {optimized_time:.4f} секунд")

    # Порівняння
    speedup = standard_time / optimized_time
    print(f"Прискорення завдяки попереднім обчисленням: {speedup:.2f}x")

if __name__ == "__main__":
    main()
```

Результат:

```
PS C:\Users\VOVA\Desktop\magister\TestAES>  & 'c:\Users\V
gpy\adapter/../..\debugpy\launcher' '51034' '--' 'C:\User
Час виконання стандартної MixColumns: 1.8214 секунд
Час виконання оптимізованої MixColumns: 1.0072 секунд
Прискорення завдяки попереднім обчисленням: 1.81x
PS C:\Users\VOVA\Desktop\magister\TestAES>
```

```
PS C:\Users\VOVA\Desktop\magister\TestAES>  c:; cd 'c:
hon.debugpy-2024.14.0-win32-x64\bundled\libs\debugpy\a
Час виконання стандартної MixColumns: 1.9022 секунд
Час виконання оптимізованої MixColumns: 1.0285 секунд
Прискорення завдяки попереднім обчисленням: 1.85x
PS C:\Users\VOVA\Desktop\magister\TestAES>
```

Додаток Г

Реалізація оптимізації RSA з використанням коротших відкритих експонент

Код програми:

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_v1_5
from Crypto.Random import get_random_bytes
import time

# Функція для шифрування з використанням довгої випадкової експоненти
def rsa_standard_encrypt(message, key_size):
    # Генерація довгої відкритої експоненти
    key = RSA.generate(key_size, e=2**16 + 1)  # стандартне e=65537
    cipher = PKCS1_v1_5.new(key)
    ciphertext = cipher.encrypt(message)
    return key, ciphertext

# Функція для шифрування з використанням оптимізованої короткої експоненти
def rsa_optimized_encrypt(message, key_size):
    # Використання малої експоненти e=65537
    key = RSA.generate(key_size, e=65537)
    cipher = PKCS1_v1_5.new(key)
    ciphertext = cipher.encrypt(message)
    return key, ciphertext

# Тестування продуктивності
def main():
    key_size = 2048  # Розмір ключа RSA у бітах
    message = get_random_bytes(128)  # Повідомлення розміром 128 байт

    print("Порівняння продуктивності RSA з різними експонентами:")

    # Шифрування зі стандартною (довгою) експонентою
    start_time = time.time()
    key_standard, ciphertext_standard = rsa_standard_encrypt(message, key_size)
    standard_time = time.time() - start_time
    print(f"Час виконання стандартного RSA: {standard_time:.6f} секунд")

    # Шифрування з оптимізованою (короткою) експонентою
    start_time = time.time()
    key_optimized, ciphertext_optimized = rsa_optimized_encrypt(message, key_size)
    optimized_time = time.time() - start_time
    print(f"Час виконання оптимізованого RSA: {optimized_time:.6f} секунд")
```

```
    # Порівняння часу виконання
    speedup = standard_time / optimized_time
    print(f"Прискорення завдяки короткій експоненті: {speedup:.2f}x")

if __name__ == "__main__":
    main()
```

Результат:

```
PS C:\Users\VOVA\Desktop\magister\TestRSA>  c:; cd 'c:\
ers\VOVA\.vscode\extensions\ms-python.debugpy-2024.14.0
RSA\TestRSA.py'
Порівняння продуктивності RSA з різними експонентами:
Час виконання стандартного RSA: 1.234982 секунд
Час виконання оптимізованого RSA: 1.069079 секунд
Прискорення завдяки короткій експоненті: 1.16x
PS C:\Users\VOVA\Desktop\magister\TestRSA> ^C
PS C:\Users\VOVA\Desktop\magister\TestRSA>
```

```
PS C:\Users\VOVA\Desktop\magister\TestRSA>  c:; cd 'c:
ers\VOVA\.vscode\extensions\ms-python.debugpy-2024.14.
RSA\TestRSA.py'
Порівняння продуктивності RSA з різними експонентами:
Час виконання стандартного RSA: 1.530569 секунд
Час виконання оптимізованого RSA: 0.695924 секунд
Прискорення завдяки короткій експоненті: 2.20x
PS C:\Users\VOVA\Desktop\magister\TestRSA>
```