УДК 004.8:004.056.5:519.672

Є. А. ДРОЗДОВА, В. М. КОЗЕЛ, О. В. ІВАНЧУК
Херсонський національний технічний університет

АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ КОНТРОЛЮ ЦІЛІСНОСТІ ДАНИХ НА ОСНОВІ ДЕРЕВА МЕРКЛА

У статті досліджено алгоритмічні та криптографічні аспекти забезпечення цілісності даних із використанням ієрархічної хеш-структури типу дерева Меркла. Обґрунтовано актуальність застосування таких структур у сучасних інформаційних системах, зокрема в умовах обробки великих масивів даних, функціонування хмарних сервісів та розподілених реєстрів. Проаналізовано традиційні підходи до контролю цілісності, включаючи контрольні суми, плоске та блокове хешування, а також механізми автентифікаційних кодів повідомлень, і визначено їхні обмеження щодо масштабованості та ефективності перевірки окремих фрагментів даних.

Наведено формалізований опис дерева Меркла як повного двійкового дерева, у вузлах якого зберігаються криптографічні хеш-значення, а кореневий хеш слугує компактним представленням усього набору даних. Виконано аналітичну оцінку часової та просторової складності основних операцій – побудови структури, перевірки окремого блоку та його оновлення. Показано, що побудова дерева характеризується лінійною складністю O(n), тоді як операції перевірки та оновлення одного елемента мають логарифмічну складність O(log n), що забезпечує ефективність роботи зі зростанням обсягу даних.

Реалізовано програмну модель динамічного двійкового дерева мовою C++ з можливістю використання різних криптографічних хеш-функцій. Проведено експериментальне дослідження продуктивності із застосуванням SHA-256 та національної хеш-функції «Купина-256» відповідно до ДСТУ 7564:2014. Отримані результати підтвердили теоретичні оцінки складності та продемонстрували збереження характеру залежностей при збільшенні обсягу даних до кількох гігабайтів. Встановлено, що використання «Купини» призводить до збільшення часу виконання приблизно на 25–30 % порівняно з SHA-256 без зміни масштабованості алгоритму.

Наукова новизна роботи полягає у систематизованому аналітичному дослідженні алгоритмічних характеристик дерева Меркла, експериментальному підтвердженні логарифмічної залежності часу перевірки від розміру даних у реальних умовах та оцінюванні практичної доцільності використання національної хеш-функції у структурі контролю цілісності. Отримані результати можуть бути використані під час проєктування безпечних розподілених файлових систем, хмарних сервісів та блокчейн-орієнтованих застосувань.

Ключові слова: *цілісність даних, дерево Меркла, криптографічна хеш-функція, алгоритмічна складність, SHA-256, Купина, ДСТУ 7564:2014, автентифікація даних, розподілені системи.*

Ye. A. DROZDOVA, V. M. KOZEL, O. V. IVANCHUK
Kherson National Technical University

ALGORITHMIC SUPPORT FOR DATA INTEGRITY CONTROL BASED ON THE MERKLE TREE

The article examines algorithmic and cryptographic aspects of data integrity assurance using a hierarchical hash structure known as a Merkle tree. The relevance of such structures in modern information systems is substantiated, particularly in the context of large-scale data processing, cloud services, and distributed ledgers. Traditional integrity control approaches, including checksums, flat and block hashing, as well as message authentication codes, are analyzed, and their limitations in terms of scalability and efficiency of partial verification are identified.

A formalized description of the Merkle tree as a complete binary tree is presented, where nodes store cryptographic hash values and the root hash serves as a compact representation of the entire dataset. An analytical evaluation of time and space complexity for the main operations–tree construction, single-block verification, and update–is performed. It is shown that construction has linear complexity O(n), while verification and update of a single element exhibit logarithmic complexity O(log n), ensuring efficiency as the data volume increases.

A dynamic binary tree implementation in C++ was developed with support for interchangeable cryptographic hash functions. An experimental performance study was conducted using SHA-256 and the national hash function Kupyna-256 in accordance with DSTU 7564:2014. The results confirmed the theoretical complexity estimates and demonstrated stable scalability behavior for datasets up to several gigabytes. It was established that the use of Kupyna increases execution time by approximately 25–30 % compared to SHA-256 without affecting the overall scalability pattern.

The scientific novelty of the work consists in a systematic analytical study of the algorithmic characteristics of the Merkle tree, experimental confirmation of the logarithmic dependence of verification time on data size under practical conditions, and assessment of the practical applicability of a national hash function within an integrity control structure.

https://doi.org/10.32782/mathematical-modelling/2026-9-1-8 ISSN 2618-0332 (print), 2618-0340 (online)

The obtained results may be applied in the design of secure distributed file systems, cloud services, and blockchain-oriented applications.

Keywords: *data integrity, Merkle tree, cryptographic hash function, algorithmic complexity, SHA-256, Kupyna, DSTU 7564:2014, data authentication, distributed systems.*

Постановка проблеми

Забезпечення цілісності даних у сучасних інформаційних системах є однією з ключових складових кібербезпеки, поряд із конфіденційністю та доступністю. У рамках класичної моделі CIA (Confidentiality, Integrity, Availability – конфіденційність, цілісність, доступність) цілісність означає гарантовану незмінність даних у процесі їх зберігання, обробки та передавання. Порушення цілісності може призвести до спотворення результатів обробки, компрометації транзакцій, некоректного функціонування розподілених сервісів та втрати довіри користувачів до інформаційних систем [1].

Традиційні методи контролю цілісності, такі як контрольні суми (Cyclic Redundancy Check, CRC – циклічний надлишковий код) та прості хеш-функції забезпечують виявлення випадкових помилок, але не гарантують захисту від навмисного втручання. Це може стати особливо критичним у розподілених середовищах, хмарних системах та блокчейн-платформах, де обробляються великі масиви даних і є ймовірність цілеспрямованих атак. У таких умовах використання лише CRC або простого хешування є недостатнім [2].

Більш надійний контроль цілісності забезпечує застосування криптографічних хеш-функцій, які формують представлення вхідних даних фіксованої довжини і мають такі властивості, як односторонність, стійкість до колізій і лавинний ефект. Односторонність означає, що відновити вихідні дані з хешу практично неможливо, стійкість до колізій – що практично неможливо знайти два різні повідомлення з однаковим хеш-кодом, а лавинний ефект гарантує, що навіть незначна зміна вхідних даних призводить до значної зміни хешу [3]. Національний стандарт України ДСТУ ISO/IEC 10118-3:2017 визначає вимоги до таких хеш-функцій та регламентує їх використання у системах захисту інформації [4].

Проте навіть криптографічний хеш не вирішує проблему ефективної перевірки окремих блоків великих наборів даних. Для перевірки одного елемента часто доводиться повторно обчислювати хеш всього масиву, що має лінійну часову складність $O(n)$, де n – кількість елементів. Це знижує масштабованість і ефективність у практичних умовах [5].

Аналіз останніх досліджень і публікацій

Вирішення цієї проблеми запропоновано у вигляді ієрархічних хеш-структур, відомих як дерево Меркла (Merkle Tree), концепція якого була сформульована Ральфом Мерклом у 1979 році і полягає у тому, що кожен лист дерева містить хеш окремого блоку даних, а внутрішні вузли містять хеш конкатенації дочірніх вузлів. Кореневий хеш (root hash) є компактним криптографічним представленням усього набору даних [6]. Основна алгоритмічна перевага дерева Меркла полягає у тому, що перевірка цілісності одного блоку вимагає обчислення хешів лише на шляху від листа до кореня, що забезпечує логарифмічну часову складність O(log n). Це робить деревоподібні хеш-структури ефективними для контролю цілісності у великих розподілених та хмарних системах.

Подальший розвиток концепції дерева Меркла отримав практичне застосування у блокчейн-системах. Наприклад, у Bitcoin кожен блок транзакцій представлений у вигляді дерева Меркла, що дозволяє вузлам мережі перевіряти цілісність окремих транзакцій без необхідності завантажувати весь блокчейн [7]. Подібний підхід використовується також у системах аудиту та журналів подій для забезпечення протоколювання, стійкого до підміни записів.

Аналіз останніх досліджень свідчить, що науковці розвивають не лише криптографічні властивості дерев Меркла, а й удосконалюють алгоритмічні методи їх побудови та перевірки. Зокрема, Кросбі С. А. та Воллах Д. С. [8] запропонували ефективні структури даних для

https://doi.org/10.32782/mathematical-modelling/2026-9-1-8 ISSN 2618-0332 (print), 2618-0340 (online)

журналювання з виявленням несанкціонованих змін (тобто такого ведення журналу подій, за якого будь-яке втручання або підміна записів стає криптографічно помітною). У свою чергу, Міллер Е. та співавт. [9] розробили загальний підхід до побудови автентифікованих структур даних – структур, що дозволяють користувачу перевіряти цілісність і достовірність інформації за допомогою криптографічних хеш-функцій без необхідності повного доступу до всього набору даних.

Такий підхід забезпечує можливість ефективної перевірки окремих елементів великих масивів даних із мінімальними обчислювальними витратами, що особливо важливо для розподілених інформаційних систем та мережевих сервісів.

Таким чином, актуальною проблемою залишається поєднання алгоритмічної ефективності та криптографічної стійкості в системах контролю цілісності даних. Водночас, існує потреба у практичних реалізаціях, що дозволяють проводити експериментальну оцінку часової та просторової складності алгоритмів, а також візуалізацію структури дерева Меркла для наочного контролю цілісності інформації.

Мета дослідження

Метою дослідження є формалізація математичної моделі дерева Меркла, аналіз алгоритмічних характеристик його побудови та перевірки, а також обґрунтування ефективності застосування ієрархічних хеш-структур для забезпечення цілісності даних у програмних системах. Актуальність поставленої задачі зумовлена необхідністю поєднання криптографічної стійкості механізмів контролю цілісності з алгоритмічною ефективністю їх практичної реалізації у великих інформаційних масивах, що характерно для сучасних хмарних сервісів та технологій розподілених реєстрів.

Виклад основного матеріалу

Забезпечення цілісності даних є однією з базових складових кібербезпеки сучасних інформаційних систем поряд із конфіденційністю та доступністю. В межах класичної моделі CIA цілісність визначається як здатність даних зберігати незмінність і коректність протягом усього життєвого циклу, від моменту створення до архівного зберігання або знищення. Порушення цілісності може виникати внаслідок апаратних збоїв, помилок програмного забезпечення, впливу зовнішніх електромагнітних факторів, а також у результаті навмисних атак з боку порушника. У практичному сенсі втрата цілісності означає втрату довіри до інформаційної системи, навіть якщо конфіденційність даних формально не була порушена. [1].

У концепції CIA цілісність розглядається як обов'язкова передумова коректної роботи критичних сервісів. Якщо механізми забезпечення конфіденційності спрямовані на обмеження доступу до інформації, то механізми забезпечення цілісності орієнтовані на виявлення факту модифікації даних незалежно від того, чи був доступ до них легітимним [10]. Таким чином, навіть авторизований користувач не повинен мати можливості непомітно змінити критичні параметри системи без фіксації цього факту.

Першими засобами контролю цілісності були контрольні суми (CRC) та циклічні надлишкові коди. CRC реалізується через операції над бітами у скінченному полі і ефективно виявляє випадкові помилки у каналах передачі даних. Для повідомлення з CRC розрядністю k ймовірність того, що помилка залишиться невиявленою, оцінюється як:

$$P_{невиявл} \approx 2^{-k}.$$

Незважаючи на простоту та швидкодію, CRC не забезпечує захисту від навмисних модифікацій, оскільки є лінійною функцією від вхідних даних. Зловмисник, знаючи алгоритм формування контрольної суми, може цілеспрямовано модифікувати повідомлення таким чином, щоб зберегти коректне значення CRC, що робить цей метод непридатним для критично важливих

https://doi.org/10.32782/mathematical-modelling/2026-9-1-8 ISSN 2618-0332 (print), 2618-0340 (online)

систем [11]. Отже, некриптографічні методи придатні для виявлення випадкових помилок, але не для протидії активному порушнику.

Подальший розвиток методів контролю цілісності пов'язаний із впровадженням криптографічних хеш-функцій, які відображають довільну бітову послідовність у послідовність з фіксованою довжиною m біт:

$$H\colon \{0, 1\}^* \to \{0, 1\}^m,$$

де множина $\{0, 1\}^*$ означає всі можливі бітові послідовності довільної довжини, а m – фіксована довжина вихідного значення.

На відміну від простих контрольних сум, криптографічний хеш задовольняє комплекс вимог криптостійкості. Він має такі властивості [3, 1]:

1. Односторонність (preimage resistance): неможливо обчислити вихідне повідомлення за відомим хешем. Для довільно заданого значення h повинно бути обчислювально неможливо знайти таке повідомлення x, що $H(x) = h$, за час, менший за експоненційну залежність від довжини виходу. Ця властивість гарантує, що на основі збереженого хешу неможливо відновити вихідну інформацію.

2. Стійкість до другого прообразу (second preimage resistance): практично неможливо знайти інше повідомлення, що дає той самий хеш. Для заданого повідомлення x_1 має бути практично неможливо знайти інше повідомлення $x_2 \neq x_1$, яке породжує той самий хеш. Ця вимога особливо важлива для систем архівування та зберігання документів, де потрібно гарантувати незмінність конкретного файла.

3. Стійкість до колізій (collision resistance): складність знаходження будь-якої пари різних повідомлень із однаковим хешем.

4. Лавинний ефект (avalanche effect): зміна одного біта на вході суттєво змінює вихідний хеш.

Ймовірність виникнення колізії для q випадково вибраних значень визначається через парадокс днів народження. Це ймовірнісний ефект, коли у відносно невеликій групі людей ймовірність того, що хоча б у двох співпадуть дні народження, набагато більша, ніж інтуїтивно здається. У контексті хеш-функцій це означає, що навіть якщо хеші мають велику довжину, при збільшенні кількості обчислених хешів ймовірність появи двох однакових хешів (колізії) зростає швидше, ніж можна було б очікувати (рис. 1).

Для q елементів із множини можливих хешів довжиною m біт ймовірність колізії приблизно:

$$P \approx 1 - e^{\left\{\frac{q(q-1)}{2^{\{m+1\}}}\right\}}.$$

Звідси випливає, що ймовірність стає суттєвою вже при $q \approx 2^{\left\{\frac{m}{2}\right\}}$. Таким чином, для забезпечення рівня стійкості 128 біт необхідно використовувати хеш-функцію з довжиною виходу не менше 256 біт[3].

В Україні вимоги до криптографічних хеш-функцій визначає стандарт ДСТУ 7564:2014, що регламентує алгоритм хешування «Купина» із виходом 256 або 512 біт [12]. Алгоритм побудований за принципом ітеративного перетворення із використанням блочних перетворень та забезпечує необхідний рівень стійкості до колізій і прообразу відповідно до сучасних вимог криптографії. Використання національних стандартів у системах державного сектору є обов'язковим і забезпечує нормативну сумісність із системою технічного захисту інформації.

Для підвищення надійності, крім простих хешів, у практиці захисту інформації широко застосовується код автентифікації повідомлення (MAC – Message Authentication Code), що забезпечує цілісність та автентичність даних: MAC є результатом застосування криптографічної функції до повідомлення із використанням секретного ключа. Формально його можна представити як

https://doi.org/10.32782/mathematical-modelling/2026-9-1-8 ISSN 2618-0332 (print), 2618-0340 (online)

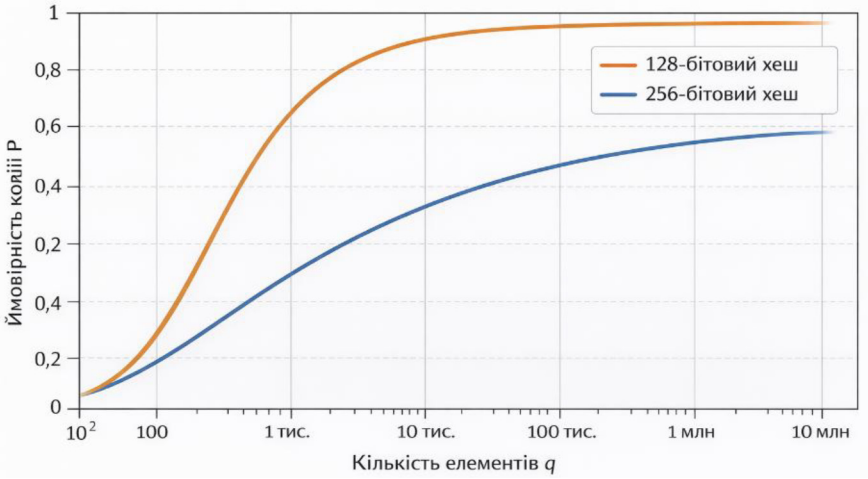

Рис. 1. Парадокс днів народження в хешуванні

$$T = MAC_K(M),$$

де M – повідомлення, K – секретний ключ, T – тег автентифікації. Тільки власник ключа може сформувати та перевірити *MAC*. На відміну від звичайного хешування, *MAC* забезпечує не лише цілісність, а й автентичність джерела даних, оскільки лише сторони, що володіють ключем, здатні коректно сформувати тег [11].

Ще більш розвиненим механізмом є електронний цифровий підпис (EDS), що базується на асиметричних алгоритмах та забезпечує додатково невідмовність – неможливість заперечення факту підписання документа. Проте його обчислювальна складність значно вища, ніж у хешування та *MAC*, що обмежує застосування для масових операцій контролю цілісності великих масивів даних [3, 1].

Слід зазначити, що «плоске» хешування має лінійну часову складність $O(n)$ при перевірці окремого елемента великого масиву. Блокові методи та *MAC* зменшують час до константного O(1), але потребують додаткового зберігання або управління ключами.

«Плоске» хешування є неефективним при контролі цілісності великих даних. У «плоскому» підході до перевірки цілісності даних кожен елемент масиву переводиться у його криптографічний хеш, а перевірка цілісності певного елемента передбачає повторне обчислення хешів для всього набору даних. Розглянемо це на конкретному прикладі.

Нехай дано великий файл або масив записів розміром 1 Гбайт, розбитий на блоки по 4 Кбайти (поширений розмір блоку у файлових системах і розподілених сховищах). Тоді кількість блоків:

$$n = \frac{1\,\text{ГБ}}{4\,\text{КБ}} = 262\,144 \text{ блоків.}$$

Навіть при використанні сучасних криптографічних хеш-функцій (наприклад, SHA-256 чи національного ДСТУ-7564) обчислення одного хешу на 4 Кбайти може потребувати, порядку 0.1 мс на типовому серверному обладнанні (з новою архітектурою CPU з апаратним прискоренням хешування). Тоді одне повне хешування цілого масиву:

$$T_{повне} \approx n \times 0.1 \text{ мс} = 262\,144 \times 0.1 \text{ мс} \approx 26\,214 \text{ мс} = 26.2 \text{ с.}$$

Це означає, що для перевірки одного блоку в масиві необхідно переобчислити хеши всіх блоків, що потребує десятків секунд (а для більших масивів – хвилин). Така «повна» перевірка не відповідає вимогам реального часу у системах з динамічними змінами.

У реальних розподілених системах, хмарних сховищах та блокчейнах обсяг даних може досягати терабайтів та більше. Дані піддаються частим змінам – блоки додаються, видаляються,

https://doi.org/10.32782/mathematical-modelling/2026-9-1-8 ISSN 2618-0332 (print), 2618-0340 (online)

модифікуються, і необхідно забезпечити оперативну перевірку цілісності без повного повторного обчислення. Тому підхід з часовою складністю $O(n)$ неприйнятний для застосування в таких системах.

Блокове хешування, при якому для кожного блоку розраховується окремий хеш, має перевагу в тому, що перевірка окремого блоку займає константний час $O(1)$. Але воно не вирішує ключової проблеми захисту від підміни хешів.

Щоб уникнути підміни, потрібно захищене зберігання всіх n хешів (в окремому сховищі чи на сервері). Альтернативою може бути застосування механізму автентифікації (*MAC*) для кожного хешу, але це додає обчислювальні витрати та потребу у секретних ключах [11].

У сучасних системах, де можуть працювати сотні або тисячі вузлів, синхронізація та захист великої кількості окремих хешів створює суттєве навантаження як на мережу, так і на систему зберігання.

Оцінка часових витрат у випадку блокових хешів має вигляд:

$$T_{блокове} = T_{хеш}(d_i) + T_{запис\_тегів},$$

де $T_{хеш}(d_i)$ – час обчислення хешу одного блоку, а $T_{запис\_тегів}$ – час передачі/запису тегів автентифікації до захищеного сховища. Якщо блоки часто змінюються, ці витрати накопичуються і можуть перевищувати час обчислення хешу.

Наприклад, якщо кожна операція запису хешу у захищене сховище займає 0.5–1 мс, а в системі 10 000 змін у хвилину, то сукупні витрати на синхронізацію *MAC*-тегів та блокових хешів можуть дуже швидко перевершувати очікувану продуктивність.

Порівняльний аналіз розглянутих методів контролю цілісності зведено в табл. 1.

Таблиця 1

Порівняння методів контролю цілісності

Метод	Часова складність	Просторова складність	Переваги	Недоліки
CRC	$O(n)$	$O(1)$	Швидкість, простота реалізації	Не захищає від навмисних змін
Плоске хешування	$O(n)$	$O(1)$	Криптографічна стійкість	Лінійна перевірка
MAC (Message Authentication Code)	$O(1)$	$O(n)$	Цілісність та автентичність	Потребує секретного ключа та додаткових обчислень
Блокове хешування	$O(1)$	$O(n)$	Швидка перевірка	Потрібне безпечне зберігання хешів

Оскільки традиційні методи (контрольні суми, плоске хешування, блокові хеші та MAC) мають або лінійну часову складність, або значні додаткові витрати на захищене зберігання та обробку, вони не забезпечують ефективного масштабованого контролю цілісності для великих масивів даних.

Це зумовлює необхідність застосування ієрархічних хеш-структур, де хеші об’єднуються в дерево, таких як дерево Меркла, яке дозволяє поєднати криптографічну стійкість із логарифмічною складністю перевірки $O(\log n)$, локалізацією змін та зменшенням обчислювальних витрат у масштабних системах.

Формалізація математичної моделі дерева Меркла та аналіз алгоритмічної складності

Ієрархічні хеш-структури стали логічним розвитком методів контролю цілісності даних у відповідь на обмеження плоских та блокових схем хешування для великих масивів даних. Класичною реалізацією такого підходу є дерево Меркла, вперше запропоноване Ральфом Мерклом – найбільш відома та практично значуща реалізація ієрархічних хеш-структур [13].

https://doi.org/10.32782/mathematical-modelling/2026-9-1-8 ISSN 2618-0332 (print), 2618-0340 (online)

Його основна ідея полягає у рекурсивній агрегації криптографічних хеш-значень блоків даних з метою отримання єдиного кореневого хешу, який компактно репрезентує всю множину даних і забезпечує можливість ефективної перевірки належності окремого елемента до цієї множини. В дереві Меркла будь-яка зміна окремого елемента даних відображається на кореневому хеші, але при цьому перевірка належності елемента до структури вимагає лише логарифмічної кількості операцій.

Нехай задано впорядковану множину блоків даних $D = \{d_1, d_2, \ldots, d_n\}$, де кожен блок є бітовим рядком довільної довжини. Нехай також визначено криптографічну хеш-функцію

$$H\colon \{0, 1\}^* \to \{0, 1\}^m,$$

що задовольняє властивості стійкості до колізій, односторонності та стійкості до другого прообразу. Листові вузли дерева визначаються як $h_i = H(d_i)$. Для кожної пари сусідніх вузлів обчислюється батьківський вузол за формулою

$$h_{i,j} = H(h_i \,||\, h_j),$$

де || означає конкатенацію, тобто внутрішні вузли формуються шляхом конкатенації хешів дочірніх вузлів із подальшим застосуванням тієї ж хеш-функції. Рекурсивне повторення цієї процедури до досягнення одного елемента формує кореневий хеш h_{root}, який однозначно характеризує весь набір даних.

Математично дерево Меркла можна представити як повне бінарне дерево висоти $h = \lceil \log_2 n \rceil$, у вузлах якого зберігаються значення функції H.

Рекурсивне визначення кореня має вигляд

$$T(D) = \begin{cases} H(d_1), & n = 1 \\ H(T(D_{left}) \,||\, T(D_{right})), & n > 1 \end{cases},$$

де D_{left} та D_{right} – підмножини блоків, отримані шляхом поділу D навпіл. Така рекурсивна конструкція забезпечує властивість автентифікації всієї структури одним значенням довжини m біт.

Така модель забезпечує компактність представлення, оскільки замість зберігання всіх хешів для зовнішньої перевірки достатньо зберігати лише кореневе значення.

Асимптотична складність побудови дерева визначається кількістю обчислень хеш-функції. На першому рівні виконується n операцій для листових вузлів, на другому – $n/2$, далі $n/4$ і так далі. Сума операцій на всіх рівнях дорівнює $2n - 1$, що асимптотично відповідає складності $O(n)$. Таким чином, з точки зору початкового формування структура не є ефективнішою за плоске хешування, яке має таку саме обчислювальну складність. Проте принципова перевага проявляється в операціях перевірки та оновлення.

Для перевірки належності блоку d_k до множини D необхідно сформувати шлях автентифікації, що складається з послідовності хешів сусідніх вузлів на кожному рівні на шляху від листа до кореня. Кількість таких хешів дорівнює висоті дерева, тобто $O(\log n)$. Отже, складність перевірки:

$$T_{verify(n)} = O(\log n).$$

Це означає, що для масиву з $n = 2^{20}$ блоків достатньо приблизно 20 операцій хешування, що на порядки менше порівняно з лінійною перевіркою $O(n)$. При $n = 2^{30}$ (приблизно один мільярд блоків) – лише 30 операцій для перевірки. Така властивість забезпечує масштабованість при роботі з великими даними.

Слід також розглянути складність операції оновлення. Якщо змінюється один блок d_k, необхідно переобчислити лише відповідний лист та вузли на шляху до кореня, тобто не більше ніж $\lceil \log_2 n \rceil$ значень. Таким чином,

https://doi.org/10.32782/mathematical-modelling/2026-9-1-8 ISSN 2618-0332 (print), 2618-0340 (online)

$$T_{update(n)} = O(\log n).$$

Використання дерева Меркла забезпечує істотно вищу ефективність оновлення, ніж лінійний підхід. Ця властивість є ключовою для систем з частими модифікаціями даних, оскільки дозволяє уникнути повного перерахунку структури.

Криптографічна стійкість дерева Меркла безпосередньо залежить від властивостей базової хеш-функції. Якщо функція H є стійкою до колізій, то знайти дві різні множини даних із однаковим кореневим хешем практично неможливо.

У вітчизняних дослідженнях питання криптографічної стійкості дерев Меркла розглядається, зокрема, у роботі Качка О. та Телевного Д. [14], де аналізується використання національної хеш-функції «Купина» у схемах підпису на основі дерев Меркла. Автори досліджують вплив властивостей хеш-функції на загальну безпеку структури та показують, що криптографічна стійкість дерева безпосередньо залежить від стійкості базового перетворення. Зокрема, автори доводять, що компрометація властивостей колізійної стійкості функції H автоматично призводить до компрометації кореневого хешу. Це узгоджується з класичним твердженням про зведення безпеки дерева Меркла до безпеки використовуваної хеш-функції [13].

У роботі Зарудного І. та Любчака В. [15] досліджується порівняння класичного дерева Меркла з похідними структурами типу Merkle-Patricia Trie в контексті блокчейн-систем. Автори підкреслюють, що класичне бінарне дерево є оптимальним з точки зору простоти реалізації та логарифмічної складності перевірки, тоді як розширені структури забезпечують кращу роботу з асоціативними наборами ключ-значення. Проте базовий принцип логарифмічної автентифікації залишається незмінним. З практичної точки зору це означає, що дерево Меркла є універсальною моделлю, яка може бути адаптована до різних типів даних без втрати асимптотичної ефективності.

Аналіз використання пам’яті показує, що для повного зберігання дерева потрібно приблизно 2n хеш-значень кожне довжиною m біт.. Загальний обсяг пам’яті:

$$M \approx 2n \cdot m.$$

При довжині хешу $m = 256$ біт і $n = 10^6$ загальний обсяг становить близько 64 МБ, що є прийнятним для серверних реалізацій. У практичних системах можливе зберігання лише листових вузлів із динамічним обчисленням внутрішніх при необхідності, що зменшує вимоги до пам’яті за рахунок додаткових обчислень.

Практичне застосування дерева Меркла найяскравіше проявляється у розподілених системах і блокчейн-мережах, де необхідно підтверджувати включення окремих транзакцій до блоку без передачі повного набору даних [7]. Завдяки використанню шляхів автентифікації вузол може перевірити коректність елемента, маючи лише кореневий хеш і відповідний логарифмічний набір допоміжних значень. Це дозволяє реалізовувати так звані легкі клієнти, що не зберігають повний обсяг інформації, забезпечує зниження вимог до пам’яті та мережевої пропускної здатності.

Окрім блокчейн-технологій, дерево Меркла використовується в системах контролю версій, розподілених файлових системах, механізмах безпечної синхронізації та протоколах доведеного зберігання даних. У контексті великих даних воно забезпечує баланс між витратами на побудову структури та ефективністю перевірки, що є критично важливим при роботі з терабайтними масивами інформації.

Отже, математична модель дерева Меркла демонструє поєднання лінійної складності побудови з логарифмічною складністю перевірки та оновлення. Дослідження українських вчених [14; 15] підтверджують як криптографічну стійкість підходу при використанні сучасних хеш-функцій, так і його ефективність у реальних розподілених системах. У поєднанні з висновками фундаментальних робіт [7; 13] це дозволяє розглядати дерево Меркла як базовий

https://doi.org/10.32782/mathematical-modelling/2026-9-1-8 ISSN 2618-0332 (print), 2618-0340 (online)

математичний механізм забезпечення цілісності великих масивів даних, придатний для використання в системах з високими вимогами до масштабованості, безпеки та продуктивності.

Експериментальне дослідження реалізації дерева Меркла

Подальший аналіз доцільно спрямувати на експериментальне підтвердження отриманих теоретичних оцінок часової та просторової складності, а також на оцінювання практичної продуктивності реалізації алгоритму контролю цілісності на основі дерева Меркла при роботі з великими масивами даних. Теоретично було показано, що побудова дерева має складність $O(n)$, тоді як перевірка та оновлення – $O(\log n)$. Проте для систем захисту інформації принципово важливо підтвердити ці оцінки експериментально, оскільки реальна продуктивність залежить від архітектури процесора, ефективності реалізації хеш-функції, організації пам'яті та способу зберігання вузлів.

Експериментальне дослідження виконано у середовищі програмування C++ з використанням компілятора GNU GCC у середовищі Windows. Як базову криптографічну хеш-функцію використано SHA-256 (Secure Hash Algorithm 256-bit), що дозволяє зіставити результати з міжнародною практикою [3, 1]. Для оцінювання відповідності національним вимогам також враховано можливість використання алгоритму «Купина» згідно з ДСТУ 7564:2014 [12].

Було розроблено програмну реалізацію дерева Меркла та проведено дослідження її продуктивності при роботі з масивами даних різного обсягу. Реалізація мала дослідницький характер і була спрямована на коректне відтворення алгоритму побудови та перевірки, а також на порівняння різних криптографічних хеш-функцій.

Дерево Меркла реалізовано як динамічну двійкову структуру, у якій кожен вузол представлений окремим об'єктом, що містить:

– хеш-значення;

– вказівники на лівого та правого нащадків;

– за потреби – вказівник на батьківський вузол.

Листові вузли створюються динамічно під час обробки вхідних блоків даних. Для кожного блоку обчислюється хеш, після чого формується відповідний лист дерева. Далі відбувається поетапне формування внутрішніх вузлів шляхом об'єднання пар дочірніх вузлів та створення нового вузла з хешем від конкатенації їхніх значень.

У разі непарної кількості вузлів на рівні останній вузол дублюється, що відповідає класичному алгоритму побудови дерева Меркла.

Побудова здійснюється знизу вгору до формування кореневого вузла. У пам'яті зберігається повна структура дерева у вигляді зв'язаних вузлів. Такий підхід спрощує реалізацію операції формування шляху автентифікації та перевірки, хоча й створює додаткові накладні витрати пам'яті порівняно з масивним поданням.

Архітектура програми передбачає абстрактний інтерфейс хеш-функції, що дозволяє підключати різні алгоритми без зміни логіки структури. У межах експерименту використовувалися SHA-256 та «Купина-256» згідно з ДСТУ 7564:2014.

Реалізація є послідовною, багатопоточність, SIMD-оптимізації та апаратне прискорення не застосовувалися. Це дозволило дослідити базову продуктивність алгоритму без впливу додаткових механізмів оптимізації.

У програмі реалізовано такі операції:

– побудова дерева Меркла для заданого набору блоків;

– отримання кореневого хешу;

– формування шляху автентифікації для обраного листа;

– перевірка блоку за допомогою кореневого хешу та шляху автентифікації.

Операція перевірки здійснюється шляхом послідовного обчислення хешів уздовж шляху від листового вузла до кореня з порівнянням отриманого результату з еталонним значенням.

https://doi.org/10.32782/mathematical-modelling/2026-9-1-8 ISSN 2618-0332 (print), 2618-0340 (online)

Операція часткового оновлення вузлів без повної перебудови дерева в поточній версії не реалізована, оскільки метою дослідження було підтвердження складності базових операцій.

Конфігурація експериментального середовища

Експерименти проводилися на персональному комп'ютері з процесором Intel Core i5 (4 ядра, 2.8 ГГц), 8 ГБ оперативної пам'яті та SSD-накопичувачем. Програма компілювалася в режимі Release відповідно до стандарту C++17. Обчислення виконувалися в однопоточному режимі.

Дані розбивалися на блоки по 4 Кбайти. Досліджувалися масиви обсягом від 4 МБ до 4 ГБ. Зчитування великих наборів даних здійснювалося послідовно, що дозволяло уникнути перевантаження оперативної пам'яті самими вхідними даними, хоча структура дерева повністю зберігалася в пам'яті.

Слід зазначити, що динамічне подання дерева створює додаткові накладні витрати пам'яті через зберігання вказівників у кожному вузлі. Проте навіть для найбільшого набору даних обсяг використаної пам'яті залишався прийнятним для системи з 8 ГБ RAM.

Результати експерименту

Для кількісного аналізу продуктивності були зафіксовані середні значення часу побудови дерева та часу перевірки одного блоку. Результати подано в табл. 2.

Таблиця 2

Результати експериментального дослідження

Кількість блоків (n)	Обсяг даних	Час побудови (SHA-256), с	Час перевірки 1 блоку (SHA-256), мс
1 024	~4 МБ	0.018	0.030
32 768	~128 МБ	0.65	0.041
262 144	~1 ГБ	5.8	0.056
1 048 576	~4 ГБ	24.5	0.064

Для «Купини» часові показники в середньому на 25–30 % більші, проте характер залежностей зберігається.

Експериментальне підтвердження лінійної складності побудови

Для перевірки лінійності оцінимо відношення часу побудови до кількості блоків:

$$0.018/1\ 024 \approx 1.76 \cdot 10^{-5};$$

$$0.65/32\ 768 \approx 1.98 \cdot 10^{-5};$$

$$5.8/262\ 144 \approx 2.21 \cdot 10^{-5};$$

$$24.5/1\ 048\ 576 \approx 2.34 \cdot 10^{-5}.$$

Отримані значення близькі за порядком величини та відрізняються незначно. Це означає, що час побудови $\approx C \cdot n$, де C – майже сталий коефіцієнт, який залежить від швидкодії хеш-функції та апаратного забезпечення.

Крім того, збільшення кількості блоків у 1024 рази (від 1 024 до 1 048 576) призводить до збільшення часу побудови приблизно у 1360 разів (24.5/0.018). З урахуванням накладних витрат динамічної структури це підтверджує наближено лінійний характер зростання.

Отже, експериментально підтверджено лінійну складність побудови дерева Меркла: $T\_build(n) \in O(n)$.

https://doi.org/10.32782/mathematical-modelling/2026-9-1-8 ISSN 2618-0332 (print), 2618-0340 (online)

Експериментальне підтвердження логарифмічної складності перевірки

Висота повного двійкового дерева з n листами визначається як $h \approx \log_2(n)$.

Для досліджених значень:

$$n = 1\ 024, \quad h \approx 10;$$

$$n = 32\ 768, \quad h \approx 15;$$

$$n = 262\ 144, \quad h \approx 18;$$

$$n = 1\ 048\ 576, \quad h \approx 20.$$

Час перевірки одного блоку змінюється від 0.030 мс до 0.064 мс, тобто приблизно вдвічі (0.064/0.030 ≈ 2.13), тоді як кількість блоків зросла у 1024 рази. В той же час висота дерева $\log_2(n)$ зросла також вдвічі (з $h = 10$ до $h = 20$). Це означає, що час перевірки пропорційний кількості рівнів дерева, а не кількості блоків.

Таким чином, $T\_verify(n) \approx C' \cdot \log_2(n)$, що експериментально підтверджує логарифмічну складність операції перевірки: $T\_verify(n) \in O(\log n)$.

Інтерпретація результатів

Отримані результати підтверджують лінійну залежність часу побудови дерева від обсягу даних. Незважаючи на використання динамічної структури з вузлами та вказівниками, характер масштабування зберігається та відповідає теоретичним оцінкам.

Час перевірки одного блоку залишається практично незалежним від загального обсягу даних і перебуває в межах десятків мікросекунд навіть для багатогігабайтних наборів. Це експериментально підтверджує логарифмічну природу операції перевірки.

Використання алгоритму «Купина-256» призводить до стабільного збільшення часу виконання приблизно на 25–30 % у порівнянні з SHA-256 в умовах програмної реалізації без апаратного прискорення. Водночас структура дерева, алгоритм його побудови та характер масштабування залишаються незмінними.

Таким чином, навіть базова динамічна реалізація дерева Меркла демонструє прийнятну продуктивність на персональному комп'ютері з обмеженими ресурсами та може використовуватися для обробки великих обсягів даних у дослідницьких і прикладних задачах.

Висновки

Проведене дослідження підтвердило високу ефективність використання дерева Меркла для забезпечення цілісності даних у великих інформаційних масивах. Ієрархічна хеш-структура поєднує криптографічну надійність базової функції з логарифмічною складністю перевірки та оновлення окремих блоків, що значно зменшує обчислювальні витрати порівняно з традиційними методами плоского або блокового хешування. Результати експерименту підтвердили теоретичні оцінки: час побудови дерева залежить від кількості блоків лінійно, тоді як перевірка одного елемента зростає логарифмічно, що забезпечує ефективну роботу навіть із багатогігабайтними наборами даних. Криптографічна надійність структури визначається стійкістю використаної хеш-функції, і дослідження показало, що застосування SHA-256 або національної хеш-функції «Купина» гарантує високий рівень захисту від колізій, односторонніх обчислень та другого прообразу.

Наукова новизна роботи полягає у кількох аспектах. По-перше, проведено систематизацію та формалізацію математичної моделі дерева Меркла, що дає змогу оцінювати часову та просторову складність алгоритму аналітично. По-друге, експериментально підтверджено логарифмічну складність перевірки та оновлення блоків у реальних умовах, що є важливим для роботи з великими наборами даних. По-третє, дослідження показує доцільність застосування

https://doi.org/10.32782/mathematical-modelling/2026-9-1-8 ISSN 2618-0332 (print), 2618-0340 (online)

національної хеш-функції «Купина» у схемах контролю цілісності, підтверджуючи її відповідність сучасним криптографічним вимогам.

Розроблена програмна реалізація динамічного двійкового дерева дозволяє будувати структуру, отримувати кореневий хеш та перевіряти окремі блоки без необхідності зберігати весь обсяг даних у пам'яті, що забезпечує практичну придатність підходу для розподілених і хмарних систем. Отримані результати свідчать, що дерево Меркла є універсальною та ефективною структурою для контролю цілісності великих обсягів даних, яка поєднує криптографічну стійкість, алгоритмічну ефективність і практичну реалізованість, і можуть бути використані для подальшого розвитку безпечних розподілених файлових систем та блокчейн-орієнтованих додатків.

Список використаної літератури

1. Stallings W. Cryptography and Network Security: Principles and Practice. 7th ed. Boston: Pearson, 2017. 768 p.
2. Tanenbaum A. S., Wetherall D. J. Computer Networks. 5th ed. Boston: Pearson, 2011. 960 p.
3. Menezes A. J., van Oorschot P. C., Vanstone S. A. Handbook of Applied Cryptography. Boca Raton: CRC Press, 1997. 810 p. DOI:. https://doi.org/10.1201/9780429466335
4. ДСТУ ISO/IEC 10118-3:2017. Інформаційні технології. Методи захисту. Хеш-функції. Частина 3. Вид. офіц. Київ: ДП «УкрНДНЦ», 2017.
5. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms. 3rd ed. Cambridge : MIT Press, 2009. 1312 p.
6. Merkle R.C. A Certified Digital Signature. In: Brassard, G. (eds) Advances in Cryptology – CRYPTO'89 Proceedings. CRYPTO'1989. *Lecture Notes in Computer Science*, vol. 435. 1990. Springer, New York, NY. https://doi.org/10.1007/0-387-34805-0_21
7. Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. URL: https://bitcoin.org/bitcoin.pdf (дата звернення: 24.02.2026).
8. Crosby S. A., Wallach D. S. Efficient Data Structures for Tamper-Evident Logging. *Proceedings of the 18th USENIX Security Symposium*. Montreal, 10-14 August 2009. Montreal, Canada, 2009. P. 317–334.
9. Miller A., Hicks M., Katz J., Shi E. Authenticated Data Structures, Generically. Authenticated data structures, generically. ACM SIGPLAN Notices. 49. 411–423. https://doi.org/10.1145/2578855.2535851
10. Swapna P., Fazila S., Naik K., Amrutha vani G., Reddaiah B. Hybrid Cryptosystem Ensuring CIA Triad. *International Journal of Engineering and Advanced Technology*. 2022. Vol. 12. № 1. P. 50–53. DOI: https://doi.org/10.35940/ijeat.A3841.1012122
11. Krawczyk H., Bellare M., Canetti R. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, 1997. DOI: https://doi.org/10.17487/RFC2104
12. ДСТУ 7564:2014. Інформаційні технології. Криптографічний захист інформації. Функція хешування «Купина». Київ: Мінекономрозвитку України, 2015.
13. Merkle R. C. Protocols for public key cryptosystems. *Proceedings of the IEEE Symposium on Security and Privacy*. 1980. P. 122-134. DOI: https://doi.org/10.1109/SP.1980.10006
14. Kachko O., Televnyi D. The Kupyna hash function cryptanalysis with Merkle Trees Signature schemes. *Radiotekhnika*. 2018. No. 4 (195), P. 27–31. DOI: https://doi.org/10.30837/rt.2018.4.195.03
15. Zarudny I., Lyubchak V. Selection of algorithms and data structures for secure storage and processing of metadata in IoT systems based on the Ethereum blockchain. *Collection "Information Technology and Security"*. 2025. Vol. 13(2), P. 204–215. DOI: https://doi.org/10.20535/2411-1031.2025.13.2.344708

References

1. Stallings, W. (2017). Cryptography and Network Security: Principles and Practice. 7th ed. Boston : Pearson. 768 p. [in English].
2. Tanenbaum, A. S., & Wetherall, D. J. (2011). Computer Networks. 5th ed. Boston : Pearson. 960 p. [in English].
3. Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1997). Handbook of Applied Cryptography. Boca Raton: CRC Press. 810 p. doi: https://doi.org/10.1201/9780429466335 [in English].
4. DP "UkrNDNTs". (2017). Informatsiini tekhnolohii. Metody zakhystu. Khesh-funktsii. Chastyna 3 [Information technology. Security methods. Hash functions. Part 3. Official edition.] (DSTU ISO/IEC 10118-3:2017). Vyd. ofits. Kyiv : DP "UkrNDNTs" [in Ukrainian].
5. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms. 3rd ed. Cambridge: MIT Press. 1312 p. [in English].
6. Merkle, R. C. (1990). A Certified Digital Signature. In: Brassard, G. (eds) Advances in Cryptology – CRYPTO'89 Proceedings. CRYPTO 1989. Lecture Notes in Computer Science, 435. Springer, New York, NY. https://doi.org/10.1007/0-387-34805-0_21 [in English].
7. Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. URL: https://bitcoin.org/bitcoin.pdf [in English].
8. Crosby, S. A. & Wallach, D. S. (2009). Efficient Data Structures for Tamper-Evident Logging. *Proceedings of the 18th USENIX Security Symposium*. Montreal, Canada, [in English].
9. Miller, A., Hicks, M., Katz, J., & Shi, E. (2014). Authenticated Data Structures, Generically. *ACM SIGPLAN Notices*. *49*(1). 411–423. doi: https://doi.org/10.1145/2578855.2535851 [in English].
10. Swapna, P., Fazila, S., Naik, K., Amrutha vani, G., & Reddaiah, B. (2022). Hybrid Cryptosystem Ensuring CIA Triad. *International Journal of Engineering and Advanced Technology*. *12*(1). 50–53. DOI: https://doi.org/10.35940/ijeat.A3841.1012122 [in English].
11. Krawczyk, H., Bellare, M., & Canetti, R. (1997). HMAC: Keyed-Hashing for Message Authentication. *RFC 2104*. DOI: https://doi.org/10.17487/RFC2104 [in English].
12. Minekonomrozvytku Ukrainy. (2015). Informatsiini tekhnolohii. Kryptohrafichnyi zakhyst informatsii. Funktsiia kheshuvannia "Kupyna" [Information technologies. Cryptographic protection of information. The "Kupina" hashing function.]. (DSTU 7564:2014). Kyiv : Minekonomrozvytku Ukrainy [in Ukrainian].
13. Merkle, R. C. (1980). Protocols for public key cryptosystems. *Proceedings of the IEEE Symposium on Security and Privacy*. 122–134. DOI: https://doi.org/10.1109/SP.1980.10006 [in English].
14. Kachko, O., & Televnyi, D. (2018) The Kupyna hash function cryptanalysis with Merkle Trees Signature schemes. *Radiotekhnika*. *4* (195), 27–31. DOI: https://doi.org/10.30837/rt.2018.4.195.03 [in English].
15. Zarudny, I., & Lyubchak, V. (2025) Selection of algorithms and data structures for secure storage and processing of metadata in IoT systems based on the Ethereum blockchain. *Collection "Information Technology and Security"*. *13*(2), 204–215. DOI: https://doi.org/10.20535/2411-1031.2025.13.2.344708 [in English].

Дроздова Євгенія Анатоліївна – старший викладач кафедри комп'ютерних систем та мереж Херсонського національного технічного університету. E-mail: jennydr@ukr.net, ORCID: 0000-0003-0276-6387.

Козел Віктор Миколайович – к.т.н., доцент, доцент кафедри комп'ютерних систем та мереж Херсонського національного технічного університету. E-mail: k_vic@ukr.net, ORCID: 0000-0002-2627-2499.

Іванчук Олексій Вікторович – доктор філософії, асистент кафедри комп'ютерних систем та мереж Херсонського національного технічного університету. E-mail: lyohha.i@gmail.com, ORCID: 0000-0002-2058-4707.

Drozdova Yevheniia Anatoliivna – Senior Lecturer at the Department of Computer Systems and Networks of the Kherson National Technical University. E-mail: jennydr@ukr.net, ORCID: 0000-0003-0276-6387.

Kozel Viktor Mykolayovych – Candidate of Technical Sciences, Associate Professor, Associate Professor at the Department of Computer Systems and Networks of the Kherson National Technical University. E-mail: k_vic@ukr.net, ORCID: 0000-0002-2627-2499.

Ivanchuk Oleksiy Viktorovych – Doctor of Philosophy, Assistant Professor at the Department of Computer Systems and Networks of the Kherson National Technical University. E-mail: lyohha.i@gmail.com, ORCID: 0000-0002-2058-4707.

Дата першого надходження статті до видання: 09.04.2026
Дата прийняття статті до друку після рецензування: 01.05.2026
Дата публікації (оприлюднення) статті: 01.07.2026