Вінницький національний технічний університет
Міністерство освіти і науки України

Кваліфікаційна наукова
праця на правах рукопису

СЕЛЕЗНЬОВ ВІТАЛІЙ ІГОРОВИЧ

УДК 004.056 +004.631.54

ДИСЕРТАЦІЯ
МЕТОДИ ТА ЗАСОБИ МАЛОРЕСУРСНОГО ГЕШУВАННЯ ДЛЯ ПРИСТРОЇВ ІНТЕРНЕТУ РЕЧЕЙ

Спеціальність 125 – «Кібербезпека»

Галузь знань 12 – «Інформаційні технології»

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело _____ В. І. Селезньов

Науковий керівник: Лужецький Володимир Андрійович, доктор технічних наук, професор

Вінниця – 2026

АНОТАЦІЯ

Селезньов В. І. Методи та засоби малоресурсного гешування для пристроїв інтернету речей. – Кваліфікаційна наукова праця на правах рукопису. Дисертація на здобуття наукового ступеня доктора філософії в галузі знань 12 «Інформаційні технології» за спеціальністю 125 «Кібербезпека». – Вінницький національний технічний університет, МОН України, Вінниця, 2026.

Дисертаційну роботу присвячено розв’язанню актуальної науково-технічної задачі зменшення апаратних витрат на реалізацію засобів гешування даних для пристроїв Інтернету речей шляхом розробки нових методів малоресурсного гешування та моделей спеціалізованих засобів, що реалізують ці методи. Актуальність дослідження визначається необхідністю забезпечення контролю цілісності даних та автентифікації пристроїв у малоресурсних IoT-системах, зокрема RFID-мітках, сенсорних вузлах та вбудованих контролерах, які мають жорсткі обмеження щодо апаратної складності, енергоспоживання та продуктивності. Відомі малоресурсні геш-функції не завжди забезпечують задовільне поєднання апаратної складності, довжини геш-значення та пропускної здатності для таких пристроїв, що підтверджує необхідність розробки нових методів і засобів малоресурсного гешування.

Дисертаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. У вступі наведено актуальність теми дослідження та сучасний стан розв’язання поставленої задачі. Відповідно до проведеного аналізу сформульовано мету роботи та основні задачі дослідження, визначено об’єкт і предмет дослідження. Наведено наукову новизну та практичне значення одержаних результатів, а також відомості про апробацію, впровадження та публікації автора за темою дисертаційної роботи.

У першому розділі проведено аналіз сучасних методів та засобів гешування для пристроїв Інтернету речей. Розглянуто задачі безпеки IoT-систем та обґрунтовано роль криптографічних геш-функцій як одного з основних інструментів забезпечення контролю цілісності даних та автентифікації

пристроїв в умовах обмежених апаратних ресурсів. Визначено основні криптографічні вимоги до геш-функцій, зокрема стійкість до колізій, атак на перший та другий прообрази, а також властивість лавинного ефекту. Проаналізовано базові конструкції геш-функцій, зокрема конструкцію Меркла–Дамґарда та конструкцію «губка», визначено принципи їх роботи, переваги та обмеження. Виконано огляд і порівняльний аналіз відомих малоресурсних геш-функцій за криптографічними та технічними характеристиками. Показано, що розглянуті малоресурсні геш-функції не завжди забезпечують задовільне поєднання апаратної складності, довжини геш-значення та пропускної здатності для пристроїв Інтернету речей з обмеженими апаратними ресурсами, що підтверджує актуальність розробки нових методів малоресурсного гешування.

У другому розділі запропоновано методи малоресурсного гешування HDG, HDGm та HGD на основі ітеративної конструкції Меркла–Дамґарда. Спільною особливістю розроблених методів є використання функції ущільнення, яка передбачає використання простих операцій додавання за модулем 256, циклічного зсуву складових проміжного геш-значення та генератора псевдовипадкової послідовності. Метод HDG передбачає внесення байтів повідомлення до проміжного геш-значення у позиціях, які визначаються псевдовипадковою послідовністю, що забезпечує відносно високу швидкість гешування при мінімальних апаратних витратах. Для підвищення інтенсивності перемішування під час оброблення коротких повідомлень запропоновано удосконалений метод HDGm, у якому використано розширене правило вибору даних, що додаються до проміжного геш-значення. Метод HGD відрізняється тим, що проміжне геш-значення оновлюється шляхом внесення псевдовипадкових байтів, сформованих генератором, а керування цим процесом здійснюється бітами вхідного повідомлення.

Апаратну реалізацію розроблених методів представлено функціональною та структурною моделями спеціалізованого процесора для гешування даних. Функціональна модель описує взаємодію зовнішнього контролера зі спеціалізованим процесором під час ініціалізації, гешування та виведення

результату. Структурна модель визначає основні функціональні блоки процесора, що забезпечують збереження проміжних геш-значень, формування псевдовипадкової послідовності, виконання додавання за модулем 256 та керування режимами роботи. Для методів HDG та HGD розроблено автоматні моделі спеціалізованих процесорів у середовищі Logisim-evolution, що дозволило перевірити коректність організації їх роботи та відповідність запропонованим малоресурсним методам гешування. Отримані оцінки апаратної складності показали, що розроблені спеціалізовані процесори забезпечують обчислення геш-значень довжиною 128–256 біт при складності 915–1680 GE для HDG та 972–1752 GE для HGD. Порівняння з відомими малоресурсними засобами гешування показало зменшення апаратних витрат на 14–24 % для HDG та на 8–20 % для HGD. За інтегральним показником FoM реалізації HDG та HDGm забезпечують найвищу апаратну ефективність серед розглянутих малоресурсних геш-функцій.

У третьому розділі розглянуто сценарії застосування розроблених методів та спеціалізованих процесорів для контролю цілісності даних у малоресурсних IoT-системах. Показано, що метод HDG-128 з LFSR зменшеної розрядності забезпечує найменші апаратні витрати при достатній пропускній здатності для контролю цілісності телеметричних повідомлень сенсорних вузлів, а метод HDG-256 з 32-розрядним LFSR є придатним для верифікації програмного забезпечення під час завантаження IoT-пристрою завдяки достатньому періоду генератора та побайтній потоковій обробці даних. Також удосконалено протокол автентифікації RFID-міток, який одночасно забезпечує взаємну автентифікацію сторін взаємодії та контроль цілісності даних, що зберігаються і передаються RFID-міткою. Для виконання однієї сесії автентифікації на стороні RFID-мітки достатньо трьох обчислень геш-функції та одного обчислення функції оновлення псевдовипадкового числа при двох обмінах повідомленнями. Аналіз безпеки показав стійкість протоколу до підробки зчитувача та RFID-мітки, повторного відтворення, активної атаки типу «людина посередині» і порушення синхронізації, а оцінка апаратної складності підтвердила можливість його

реалізації на пасивній RFID-мітці з використанням HDG-процесора.

У четвертому розділі описано проведення експериментальних досліджень розроблених методів та протоколу автентифікації. Для виконання експериментів розроблено програмну бібліотеку методів гешування мовою C++ з компіляцією у Python-модуль, а також програмний засіб для віддаленого статистичного тестування геш-функцій за методиками NIST STS та Dieharder. Коректність програмної реалізації перевірено зіставленням обчислюваних геш-значень з результатами моделювання спеціалізованих процесорів у середовищі Logisim-evolution. Експериментальне дослідження статистичних характеристик за пакетами NIST STS та Dieharder, а також дослідження лавинного ефекту виконано для кожного з розроблених методів з різними параметрами. За результатами дослідження визначено умови, за яких розроблені методи формують геш-значення з належними статистичними характеристиками, та встановлено, що метод HGD забезпечує середнє значення лавинного ефекту на рівні 50 %. Виконано порівняльне дослідження швидкості програмних реалізацій розроблених методів та відомих малоресурсних геш-функцій, за результатами якого встановлено, що для 128-бітних геш-значень методи HDG та HDGm перевершують відомі аналоги за швидкістю у 100–287 разів, а метод HGD – у 11–33 рази. Програмну реалізацію протоколу автентифікації RFID-міток виконано на основі бібліотеки розроблених методів гешування, що дозволило перевірити коректність роботи протоколу та його стійкість до характерних сценаріїв атак.

Результати досліджень впроваджено у ТОВ «Солар Стар АГРО», ТОВ «Каскад-Безпека» та у навчальний процес кафедри захисту інформації ВНТУ.

Ключові слова: гешування; геш-функція; спеціалізований криптопроцесор; малоресурсна криптографія; апаратна складність; криптоаналіз; криптостійкість; IoT; RFID; бездротові сенсорні мережі; контроль цілісності; протокол автентифікації; інформаційно-комунікаційна система; захист даних; кібербезпека.

ABSTRACT

Seleznov V. I. Lightweight Hashing Methods and Their Hardware Implementations for Internet of Things Devices. – Qualifying scientific work on the rights of the manuscript. Dissertation for obtaining the scientific degree of Doctor of Philosophy in the field of knowledge 12 «Information Technologies», speciality 125 «Cybersecurity». – Vinnytsia National Technical University, Ministry of Education and Science of Ukraine, Vinnytsia, 2026.

The dissertation solves a relevant scientific and technical problem of reducing the hardware complexity of data hashing implementations for Internet of Things devices through the development of new lightweight hashing methods and models of specialized processors that implement these methods. The relevance of the research is determined by the need to ensure data integrity verification and device authentication in resource-constrained IoT systems, in particular RFID tags, sensor nodes and embedded controllers, which are subject to strict constraints on hardware complexity, power consumption and performance. Existing lightweight hash functions do not always provide a satisfactory combination of hardware complexity, hash value length and throughput for such devices, which confirms the need for new lightweight hashing methods and their hardware implementations.

The dissertation consists of an introduction, four chapters, conclusions, a list of references and appendices. The introduction presents the relevance of the research topic and the current state of the problem under consideration. Based on the conducted analysis, the aim of the work and the main research tasks are formulated, and the object and subject of the research are defined. The scientific novelty and practical significance of the obtained results are presented, along with information on the approbation, implementation and publications of the author on the topic of the dissertation.

The first chapter presents an analysis of contemporary hashing methods and their hardware implementations for Internet of Things devices. The security challenges of IoT systems are considered, and the role of cryptographic hash functions as a key tool for data integrity verification and device authentication under constrained hardware

resources is substantiated. The main cryptographic requirements for hash functions are determined, including collision resistance, resistance to preimage and second-preimage attacks, and the avalanche effect. The basic constructions of hash functions, namely the Merkle–Damgård construction and the sponge construction, are analyzed, and their operating principles, advantages and limitations are identified. A review and comparative analysis of known lightweight hash functions by their cryptographic and technical characteristics is performed. It is shown that the considered lightweight hash functions do not always provide a satisfactory combination of hardware complexity, hash value length and throughput for Internet of Things devices with constrained hardware resources, which confirms the relevance of developing new lightweight hashing methods.

The second chapter introduces the lightweight hashing methods HDG, HDGm and HGD, which are based on the iterative Merkle–Damgård construction. A common feature of the developed methods is the use of a compression function that employs simple operations of addition modulo 256, cyclic shifts of intermediate hash value bytes and a pseudorandom sequence generator. The HDG method adds message bytes to the intermediate hash value at positions determined by the pseudorandom sequence, which provides a relatively high hashing speed with minimal hardware cost. To increase the diffusion intensity when processing short messages, the HDGm method is proposed, in which an extended rule for selecting the data added to the intermediate hash value is employed. The HGD method differs in that the intermediate hash value is updated by adding pseudorandom bytes generated by the generator, while this process is controlled by the bits of the input message.

The hardware implementation of the developed methods is described through functional and structural models of a specialized hashing processor. The functional model captures the interaction between an external controller and the specialized processor during initialization, hashing and result output. The structural model defines the main functional blocks of the processor, which provide storage of intermediate hash values, generation of the pseudorandom sequence, computation of addition modulo 256 and control of operating modes. For the HDG and HGD methods, automaton

models of specialized processors were developed in the Logisim-evolution environment, which made it possible to verify the correctness of their operation and their correspondence to the proposed compression functions. The obtained estimates of hardware complexity show that the developed specialized processors compute hash values of 128–256 bits at a complexity of 915–1680 GE for HDG and 972–1752 GE for HGD. A comparison with known lightweight hash function implementations demonstrates a hardware cost reduction of 14–24 % for HDG and 8–20 % for HGD. According to the integral Figure of Merit (FoM), the HDG and HDGm implementations achieve the highest hardware efficiency among the considered lightweight hash functions.

The third chapter examines application scenarios of the developed methods and specialized processors for data integrity verification in resource-constrained IoT systems. It is shown that the HDG-128 method with a reduced-width LFSR provides the lowest hardware cost at sufficient throughput for integrity verification of telemetry messages from sensor nodes, while the HDG-256 method with a 32-bit LFSR is suitable for software verification during boot of an IoT device due to a sufficient generator period and byte-wise data processing. An RFID tag authentication protocol is also enhanced, providing both mutual authentication of the interacting parties and integrity verification of data stored and transmitted by an RFID tag. To complete a single authentication session on the RFID tag side, three hash function computations, one pseudorandom number update and two message exchanges are sufficient. The security analysis confirmed the resistance of the protocol to reader and tag impersonation attacks, replay attacks, an active man-in-the-middle attack and desynchronization, while the assessment of hardware complexity confirmed the feasibility of its implementation on a passive RFID tag using the HDG processor.

The fourth chapter presents the experimental studies of the developed methods and the authentication protocol. To carry out the experiments, a software library of the hashing methods was developed in C++ and compiled into a Python module, along with a software tool for remote statistical testing of hash functions using the NIST STS and Dieharder test suites. The correctness of the software implementation is verified

by comparing the computed hash values with the simulation results of the corresponding specialized processors in the Logisim-evolution environment. An experimental study of the statistical characteristics using the NIST STS and Dieharder test suites, as well as a study of the avalanche effect, was performed for each of the developed methods with various parameters. Based on the results obtained, the conditions under which the developed methods produce hash values with appropriate statistical characteristics have been determined, and it has been established that the HGD method achieves an average avalanche effect of 50 %. A comparative study of the speed of software implementations of the developed methods and known lightweight hash functions showed that, for 128-bit hash values, the HDG and HDGm methods are 100–287 times faster than the known counterparts, while the HGD method is 11–33 times faster. The software implementation of the RFID tag authentication protocol was performed on the basis of the library of the developed hashing methods, which made it possible to verify the correctness of the protocol operation and its resistance to typical attack scenarios.

The research results of the dissertation were implemented at “Solar Star AGRO” LLC and “Kaskad-Bezpeka” LLC, and introduced into the educational process of the Department of Information Protection of Vinnytsia National Technical University.

Keywords: hashing; hash function; specialized cryptoprocessor; lightweight cryptography; hardware complexity; cryptanalysis; cryptographic resilience; IoT; RFID; Wireless Sensor Networks; data integrity; authentication protocol; information and communication system; data security; cybersecurity.

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ ТА ВІДОМОСТІ ПРО АПРОБАЦІЮ РЕЗУЛЬТАТІВ ДИСЕРТАЦІЇ

Статті в журналах, що включені до переліку наукових фахових видань України:

1. Селезньов В. І., Лужецький В. А. Метод малоресурсного гешування типу «Дані – Генератор». *Кібербезпека: освіта, наука, техніка,* 2023. Т. 2, № 22. С. 84–95. URL: https://doi.org/10.28925/2663-4023.2023.22.8495.
2. Лужецький В. А., Селезньов В. І. Огляд підходів та методів малоресурсного гешування даних. *Вісник Вінницького політехнічного інституту*, 2025. № 6. С. 99–112. URL: https://doi.org/10.31649/1997-9266-2025-183-6-99-112.
3. Luzhetskyi V., Seleznov V. Hardware implementation of the HDG hash function. *Bulletin of Cherkasy State Technological University*. 2025. Т. 30, № 2, С. 10–21. URL: https://doi.org/10.62660/bcstu/2.2025.10.
4. Лужецький В., Селезньов В., Хохлачова Ю. Протокол автентифікації засобів інтернету речей з використанням RFID-міток. *Кібербезпека: освіта, наука, техніка*. 2026. Т. 4, № 32. С. 987–1001. URL: https://doi.org/10.28925/2663-4023.2026.32.1205.

Публікації в матеріалах конференцій, тезах доповідей:

5. Селезньов В. І. Аналіз методів малоресурсного гешування. Матеріали LII науково-технічної конференції підрозділів ВНТУ. Вінниця, 21–23 червня 2023 р. URL: https://ir.lib.vntu.edu.ua/handle/123456789/38623 (дата звернення: 14.04.2026).
6. Селезньов В. І. Програмний засіб для віддаленого статистичного тестування методів малоресурсного гешування за допомогою пакету NIST STS 822. Матеріали LIII науково-технічної конференції підрозділів ВНТУ. Вінниця, 20–22 березня 2024 р. URL: https://ir.lib.vntu.edu.ua//handle/123456789/41890 (дата звернення: 14.04.2026).

7. Селезньов В. І., Лужецький В. А. Удосконалення методу малоресурсного гешування HDG. Матеріали XIII Міжнародної науково-технічної конференції «ITSec-2024 Безпека інформаційних технологій». Львів, 9–11 травня 2024 р. URL: https://sci.ldubgd.edu.ua/bitstream/123456789/14350/1/2024-ITSec_%D0%B7%D0%B1%D1%96%D1%80%D0%BD%D0%B8%D0%BA_v_1.pdf (дата звернення: 14.04.2026).
8. Лужецький В. А., Селезньов В. І. Метод малоресурсного гешування типу «генератор-дані». Матеріали міжнародної науково-практичної конференції «ІТКМ-2024». Івано-Франківськ, 21–24 травня 2024 р. URL: https://journal.comp-sc.if.ua/test/public/journals/zbirnyk_2024.pdf (дата звернення: 14.04.2026).
9. Селезньов В. І. Засіб для гешування даних методом HDG. Матеріали LIV Всеукраїнської науково-технічної конференції підрозділів ВНТУ. Вінниця, 24–27 березня 2025 р. URL: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/48653/24074.pdf (дата звернення: 14.04.2026).

Авторські свідоцтва, дипломи, патенти:

10. Селезньов В. І., Лужецький В. А. Свідоцтво про реєстрацію авторського права на твір № 127025 «Комп'ютерна програма «Засіб для віддаленого статистичного тестування методів малоресурсного гешування за допомогою пакету NIST STS 822». Державна організація «Український національний офіс інтелектуальної власності та інновацій». Дата реєстрації 31 липня 2024 р.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IoT – Інтернет речей (англ., Internet of Things)

BLE – енергоефективна технологія Bluetooth (англ., Bluetooth Low Energy)

RFID – радіочастотна ідентифікація (англ., Radio Frequency Identification)

GE – вентильні еквіваленти (англ., gate equivalent)

SPN – мережа підстановок-перестановок (англ., substitution-permutation network)

WSN – бездротові сенсорі мережі (англ., Wireless Sensor Networks)

FoM – коефіцієнт ефективності апаратного пристрою (англ., Figure of Merit)

ASIC – інтегральна схема для специфічного застосування (англ. Application-specific integrated circuit)

FPGA – програмована користувачем вентильна матриця (англ. Field-Programmable Gate Array)

LFSR – регістр зсуву з лінійним зворотним зв’язком (англ. linear feedback shift register)

CLI – інтерфейс командного рядка (англ., Command Line Interface)

RAM – оперативна пам’ять (англ., Random Access Memory)

ВСТУП

Актуальність роботи. З набуттям широкого розповсюдження технологій Інтернету речей (IoT) постає задача забезпечення безпеки для пристроїв та засобів, що потребують порівняно низького енергоспоживання та мають незначні обчислювальні потужності. Серед таких пристроїв RFID-мітки для ідентифікації об’єктів, бездротові сенсори, датчики промислових систем керування та інші вбудовані системи [1-3]. Забезпечення цілісності та автентичності даних у таких пристроях потребує застосування криптографічних механізмів, зокрема функцій гешування. У зв’язку з тим, що звичайні геш-функції потребують достатньо великих апаратних витрат на реалізацію і, як наслідок, споживають багато енергії, тому з’явилась необхідність у так званих малоресурсних або «легких» геш-функціях, які призначені для використання у пристроях з обмеженими ресурсами.

За останні роки світова наукова спільнота досягла значного прогресу в цьому напрямі. Вагомий внесок у розвиток теорії та практики гешування даних для пристроїв з обмеженими обчислювальними можливостями зробили вчені багатьох країн: Jean-Philippe Aumasson (Швейцарія) [4, 5], Jian Guo [6, 7], Thomas Peyrin (Сінгапур) [6], Andrey Bogdanov (Данія/Бельгія) [8, 9, 10], Gregor Leander (Німеччина) [8], Aleksandra Mileva (Північна Македонія) [11], Bernardi Pranggono [12], William J Buchanan (Великобританія) [13], Dag Johansen (Норвегія) [14], Susila Windarta (Індонезія) [12], Deena Nath Gupta [15, 17], Mangalampalli K S Manyam (Індія) [16], Ling Huang (Китай) [18], Shoichi Hirose (Японія) [19, 20] та інші.

Серед українських вчених, які сприяли фундаментальним дослідженням у галузі прикладної криптографії, що охоплюють теорію та практику побудови криптографічних примітивів, зокрема в контексті їх адаптації для малоресурсних пристроїв, необхідно, в першу чергу, відзначити І. Д. Горбенка [21], О. О. Кузнєцова [21], С. П. Євсеєва [22], В. А. Лужецького [23], І. О. Розломій [24], Ю. В. Баришева [25], А. О. Фесенка [26], О. А. Смірнова та Т. В. Смірнову [27, 28].

Сучасні методи та засоби гешування надають достатній рівень безпеки, але вимагають значних апаратних витрат. Навіть спеціалізовані геш-функції, розроблені з урахуванням обмежених апаратних ресурсів, не завжди можуть бути використані в найпростіших вбудованих пристроях. Малоресурсні геш-функції в переважній більшості побудовані з використанням конструкцій Меркла–Дамґарда та типу «губка», які допускають широкий спектр реалізацій з різною організацією внутрішніх перетворень, унаслідок чого апаратні витрати відповідних геш-функцій можуть істотно відрізнятися [12]. Для значної частини подібних геш-функцій при формуванні геш-значення довжиною 256 біт складність їх апаратної реалізації перевищує декілька тисяч вентильних еквівалентів (GE), що хоча і відповідає стандарту малоресурсної криптографії [29, 30], але є критичним обмеженням для ряду класів пристроїв з обмеженими обчислювальними ресурсами [2, 31].

Саме тому актуальність дослідження полягає у необхідності зменшення апаратних витрат на реалізацію засобів гешування даних, зокрема шляхом розробки методів малоресурсного гешування та структурних моделей засобів, що реалізують ці методи.

Зв'язок роботи з науковими програмами, планами, темами. Дисертаційне дослідження виконано на кафедрі захисту інформації Вінницького національного технічного університету відповідно до плану науково-дослідної роботи кафедри. Окремі результати дослідження пов'язані з науково-дослідною темою 51-К3 «Методологія створення КСЗІ для об'єктів критичної інфраструктури», що виконувалася на кафедрі протягом 2022–2025 рр. Зв'язок із зазначеною темою полягає в розробці методів і засобів малоресурсного гешування для забезпечення контролю цілісності даних та автентифікації пристроїв у складі комплексних систем захисту інформації.

Мета та задачі дослідження. Метою дослідження є зменшення апаратної складності на реалізацію засобів гешування для пристроїв Інтернету речей шляхом розробки методів малоресурсного гешування та структурних моделей засобів, що реалізують ці методи.

Для досягнення поставленої мети в дисертаційній роботі сформовано та розв’язано такі задачі:

- аналіз сучасних методів малоресурсного гешування даних та їх застосування для контролю цілісності даних та автентифікації в IoT-системах;
- розробка методів малоресурсного гешування даних;
- розробка моделей спеціалізованих процесорів для гешування даних;
- розробка протоколу автентифікації RFID-міток з контролем цілісності даних;
- експериментальні дослідження розроблених методів гешування та протоколу автентифікації.

Об’єктом дослідження є процеси контролю цілісності даних та автентифікації пристроїв Інтернету речей.

Предметом дослідження є методи малоресурсного гешування та моделі апаратних засобів гешування даних для пристроїв Інтернету речей.

Методи дослідження. При розв’язанні поставлених задач в дисертаційній роботі були використані теорія алгоритмів та методи криптографічних перетворень – при розробці методів малоресурсного гешування даних; методи математичної статистики – при експериментальному дослідження статистичних властивостей розроблених методів гешування; методи теорії цифрових автоматів – при побудові спеціалізованого процесора для гешування даних та моделюванні апаратних пристроїв для гешування.

Наукова новизна одержаних результатів. В результаті виконаних досліджень отримано такі наукові результати:

- вперше запропоновано метод малоресурсного гешування типу «дані-генератор», який, на відміну від відомих, передбачає формування проміжних геш-значень шляхом нагромадження байтів вхідних даних які керуються псевдовипадковою послідовністю, сформованою генератором, що забезпечує зменшення апаратних витрат на реалізацію

геш-функції на 14–24% порівняно з відомими малоресурсними засобами;

- вперше запропоновано моделі спеціалізованого процесора для гешування даних, які, на відміну від відомих, містять нагромаджувальний суматор та генератор псевдовипадкової послідовності, що забезпечує підвищення інтегрального коефіцієнта ефективності апаратної реалізації в 2,7–4,5 рази порівняно з відомими малоресурсними апаратними засобами;
- удосконалено протокол автентифікації RFID-міток, який, на відміну від відомих, використовує лише геш-функції та генерування псевдовипадкового числа як криптографічні примітиви, що забезпечує розширення функціональних можливостей шляхом одночасної взаємної автентифікації сторін і контролю цілісності даних, які зберігаються та передаються RFID-міткою.

Практичне значення одержаних результатів. Практичне значення одержаних результатів полягає в тому, що:

- розроблені моделі спеціалізованих процесорів гешування даних можуть бути використані як основа для створення апаратних засобів Інтернету речей з обмеженими ресурсами, які реалізують контроль цілісності даних та автентифікацію пристроїв;
- розроблена програмна бібліотека для моделювання методів гешування може бути використана для дослідження характеристик методів малоресурсного гешування;
- розроблений програмний засіб для віддаленого статистичного тестування методів гешування за методикою NIST STS та Dieharder може бути використаний для експериментальної перевірки статистичних властивостей геш-функцій;
- запропоновані методи гешування та протокол автентифікації можуть бути використані при проєктуванні засобів контролю цілісності та

автентифікації даних у малоресурсних IoT-пристроях.

Розроблені моделі спеціалізованих процесорів для малоресурсного гешування та протокол автентифікації з контролем цілісності даних використано при створенні системи ідентифікації на основі RFID-міток (ТОВ «Солар Стар АГРО», Вінницька обл., акт про використання результатів дисертаційної роботи від 19.04.2026).

Розроблені методи малоресурсного гешування впроваджено у систему IoT-моніторингу внутрішніх ресурсів підприємства для перевірки цілісності телеметричних даних (ТОВ «Каскад-Безпека», м. Вінниця, акт про впровадження результатів дисертаційної роботи від 05.04.2026).

Методи малоресурсного гешування, структури спеціалізованих процесорів та програмний засіб для статистичного тестування впроваджено у навчальний процес кафедри захисту інформації Вінницького національного технічного університету у дисципліни «Прикладна криптологія» та «Криптографічний захист інформації» для студентів спеціальності 125 «Кібербезпека та захист інформації» (акт про впровадження результатів дисертаційної роботи від 10.04.2026).

Особистий внесок здобувача.

Основні наукові результати дисертаційної роботи отримано автором особисто під час проведення досліджень у Вінницькому національному технічному університеті. У наукових працях, що опубліковані у співавторстві, особисто здобувачеві належить: аналіз методів малоресурсного гешування, результати порівняння малоресурсних геш-функцій [32, 36]; метод малоресурсного гешування типу «дані–генератор», результати статистичного тестування геш-функції HDG [33]; метод малоресурсного гешування типу «генератор–дані», оцінка складності апаратної реалізації методу [39]; удосконалений метод малоресурсного гешування HDG, результати статистичного тестування [38]; моделі спеціалізованого процесора для гешування та алгоритми роботи блоків HDG-процесора, оцінки апаратної складності [34]; програмний засіб для віддаленого статистичного тестування

методів малоресурсного гешування за допомогою пакету NIST STS 822 [37, 41]; протокол автентифікації для RFID-міток з контролем цілісності даних, аналіз безпеки протоколу [35].

Апробація результатів дисертації. Основні наукові та практичні результати досліджень доповідалися та обговорювалися на конференціях:

- LII Всеукраїнська науково-технічна конференція підрозділів Вінницького національного технічного університету, м. Вінниця, 21-23 червня 2023 р;
- LIII Всеукраїнська науково-технічна конференція підрозділів Вінницького національного технічного університету, м. Вінниця, 20-22 березня 2024 р.;
- XIII Міжнародна науково-технічна конференція «ITSec-2024 Безпека інформаційних технологій», м. Львів, 9-11 травня 2024 р.;
- Міжнародна науково-практичної конференції МНПК «ІТКМ-2024», м. Івано-Франківськ, 21–24 травня 2024 р.;
- LIV Всеукраїнська науково-технічна конференція підрозділів Вінницького національного технічного університету, м. Вінниця, 24-27 березня 2025 р.

Публікації. Основні результати дисертації опубліковано в 10 роботах, з яких 4 статті у фахових виданнях, 5 тез доповідей та 1 свідоцтво авторського права на твір.

Структура і обсяг дисертації. Робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг дисертації – 172 сторінки, з них основний зміст – 129 сторінок, 23 таблиця, 40 рисунків. Список використаних джерел містить 110 найменувань.

РОЗДІЛ 1
АНАЛІЗ СУЧАСНИХ МЕТОДІВ ТА ЗАСОБІВ ГЕШУВАННЯ ДЛЯ ІОТ

1.1 Пристрої Інтернету речей та задачі забезпечення їх безпеки

Інтернет речей являє собою мережу взаємопов’язаних фізичних об'єктів, здатних збирати та обмінюватися даними без прямого втручання людини. За даними аналітичної компанії IoT Analytics, кількість підключених пристроїв сягнула 21,1 мільярда наприкінці 2025 року з прогнозованим зростанням до 39 мільярдів до 2030 року [3, 42]. Таке зростання пов’язане з масовим впровадженням IoT у промисловості, транспорті, енергетиці, медицині, міському господарстві, сучасній робототехніці та інших прикладних сферах [26, 42].

Різноманітність IoT-застосувань породжує велику кількість типів пристроїв з різними характеристиками. Пасивні RFID-мітки працюють без власного джерела живлення, отримуючи енергію від зчитувача для ідентифікації товарів. Бездротові сенсорні мережі (WSN) об’єднують автономні сенсори для моніторингу параметрів довкілля та промислових процесів. Носимі сенсорні пристрої збирають біометричні дані користувачів. Розумні лічильники автоматизують облік споживання енергоресурсів [2, 26, 44]. Переважна більшість таких пристроїв належить до категорії малоресурсних обчислювальних систем з жорсткими апаратними та енергетичними обмеженнями.

Для значної частини пристроїв Інтернету речей характерними є невисока продуктивність мікроконтролерів, обмежений обсяг оперативної та енергонезалежної пам’яті, а також необхідність працювати в режимі мінімального енергоспоживання. Особливо критичними ці обмеження є для RFID-міток, автономних сенсорних вузлів та інших вбудованих пристроїв, які тривалий час функціонують від батарей або за рахунок енергозбирання. У таких умовах навіть порівняно нескладні з точки зору універсальних обчислювальних платформ криптографічні перетворення можуть виявитися надто витратними [1, 36, 45].

Додатковою особливістю IoT є те, що передавання даних часто здійснюється через бездротові канали зв’язку, які є потенційно доступними для підслуховування, перехоплення та активного впливу. Багато пристроїв розміщуються у фізично незахищеному середовищі, працюють без постійного адміністрування та взаємодіють із великою кількістю інших вузлів. Унаслідок цього проблема безпеки в IoT є складнішою, ніж у традиційних комп’ютерних системах, оскільки поєднує мережеві загрози, фізичну доступність обладнання та обмеженість ресурсів, доступних для реалізації механізмів захисту [1, 2, 44].

За таких умов для пристроїв Інтернету речей ключового значення набуває забезпечення безпеки інформації, що зберігається, обробляється та передається мережею. Практика побудови сучасних IoT-рішень показує, що для таких систем критичними є не лише задачі конфіденційності, а насамперед достовірність джерела даних, захист від несанкціонованої модифікації повідомлень та підтвердження справжності учасників взаємодії. Саме тому у багатьох сучасних дослідженнях у сфері IoT особлива увага приділяється механізмам взаємної автентифікації, встановлення сеансових ключів і перевірки цілісності даних у середовищах з обмеженими ресурсами [46, 47, 48].

Контроль цілісності даних є однією з базових вимог безпеки IoT-систем. Телеметричні повідомлення від сенсорів, службові команди, ідентифікаційні параметри та інші дані, що циркулюють у мережі, повинні бути захищені від навмисної або випадкової зміни. Порушення цілісності в IoT-середовищі може призвести не лише до спотв орення інформації, але й до помилкових рішень на рівні систем автоматизованого керування, моніторингу чи обліку. Тому засоби перевірки цілісності мають бути невід’ємною складовою архітектури безпеки таких систем [15, 45, 49].

Не менш важливою є автентифікація пристроїв. Без надійного підтвердження справжності вузла мережі неможливо виключити підміну пристрою, підключення несанкціонованого обладнання або повторне використання перехоплених повідомлень. Для IoT-середовища це особливо суттєво, оскільки пристрої часто взаємодіють автоматично, без участі

користувача, а отже рішення про довіру до джерела повідомлення повинно прийматись криптографічними засобами [46, 49]. Саме тому автентифікація в IoT зазвичай розглядається разом із задачею контролю цілісності, а не окремо від неї.

Водночас застосування відомих криптографічних алгоритмів у таких умовах є проблематичним. Традиційні алгоритми, що широко використовуються в інформаційних системах загального призначення, орієнтовані на потужніші апаратні платформи і не завжди придатні для безпосереднього використання в малоресурсних пристроях. Причиною цього є високі вимоги до пам’яті, часу виконання, енергоспоживання та площі апаратної реалізації. У зв’язку з цим для IoT формується окремий напрям малоресурсної криптографії, в межах якого розробляються спеціалізовані примітиви та протоколи, адаптовані до обмежень вбудованих систем [1, 2, 45].

Серед криптографічних примітивів, що мають найбільше значення для розв’язання задач контролю цілісності та автентифікації, важливе місце займають функції гешування. Геш-функції використовуються для формування компактних контрольних значень, побудови кодів автентифікації повідомлень, реалізації протоколів типу challenge–response, а також у складі інших криптографічних механізмів [22, 50]. У випадку малоресурсних IoT-пристроїв геш-функція є особливо привабливою, оскільки за відносно простої структури може використовуватись для розв’язання одразу кількох задач безпеки.

Разом з тим, навіть спеціалізовані малоресурсні криптографічні перетворення не завжди забезпечують необхідний компроміс між рівнем криптографічної стійкості, продуктивністю та апаратними витратами. З огляду на це для пристроїв Інтернету речей необхідні такі методи гешування, які поєднують достатній рівень криптографічної стійкості з малими апаратними та енергетичними витратами. Саме тому специфіка IoT-середовища вимагає окремого розгляду вимог до криптографічних геш-функцій, орієнтованих на роботу в умовах обмежених ресурсів.

1.2 Основні вимоги до криптографічних функцій гешування даних

Криптографічне гешування є одним із базових механізмів сучасної прикладної криптографії та використовується у широкому спектрі задач інформаційної безпеки, зокрема для контролю цілісності даних [22, 50], автентифікації, побудови цифрових підписів [14], кодів автентифікації повідомлень (MAC) [2, 11, 26], протоколів автентифікації [46], а також у складі більш складних криптографічних примітивів і протоколів [11, 22]. Незважаючи на простоту загальної ідеї, геш-функції є складними за внутрішньою організацією алгоритмами, вимоги до яких визначаються як криптографічними властивостями, так і умовами практичного застосування.

У загальному випадку гешуванням називають процес перетворення повідомлення довільної довжини у бітову послідовність фіксованої довжини, яку називають геш-значенням цього повідомлення [22, 51].

Формально геш-функцію $hash$ визначають як відображення:

$$hash: \{0,\ 1\}^* \to \{0,\ 1\}^l, \tag{1.1}$$

де $\{0,\ 1\}^*$ – множина бітових послідовностей довільної довжини, а l – фіксована довжина геш-значення в бітах, причому $l \geq 1$. При цьому для будь-якого повідомлення M геш-значення h визначається як $h = hash(\mathrm{M})$, де $h \in \{0,\ 1\}^l$.

Криптографічною вважається геш-функція, яка задовольняє низку вимог, що визначають безпеку та можливість застосування такої геш-функції у криптографічних системах [22, 52]. Першочерговою вимогою є стійкість до знаходження першого прообразу, яка полягає у практичній складності відновлення вхідного повідомлення за відомим геш-значенням цього повідомлення. Таку вимогу також визначають як однобічність (незворотність) процесу гешування, оскільки вона забезпечує незворотний характер перетворення та унеможливлює використання результату обчислень $hash(\mathrm{M}) = h$ для отримання початкового повідомлення M, яке й називають першим

прообразом.

Не менш важливою вимогою є стійкість до знаходження другого прообразу. Для заданого повідомлення M, другим прообразом називається повідомлення M', результат гешування якого збігається з результатом гешування заданого повідомлення, за умови, що $M \neq M'$. Стійкість до знаходження другого прообразу означає практичну складність знаходження повідомлення M', яке б задовольняло рівність $hash(M) = hash(M')$ для заданого повідомлення M [22, 52, 53]. Ця вимога є критичною для застосувань, пов'язаних із перевіркою цілісності та автентичності даних, оскільки унеможливлює підміну повідомлень без зміни результату гешування. Ключовою вимогою до криптографічних геш-функцій є стійкість до колізій, яка полягає у практичній неможливості знайти будь-яку пару різних повідомлень $(M_1; M_2)$, для яких виконується рівність $hash(M_1) = hash(M_2)$, де $M_1 \neq M_2$ [22, 53]. З урахуванням фіксованої довжини вихідних даних, наявність колізій є теоретично неминучою, однак криптографічна геш-функція повинна забезпечувати обчислювальну складність пошуку колізій на рівні, неприйнятному для здійснення практичних атак.

Окрім загальних вимог стійкості до знаходження прообразів та колізій, у сучасній криптографії виділяють низку додаткових властивостей, що визначають придатність геш-функцій для практичного застосування. Важливою характеристикою є детермінованість обчислювального процесу, яка означає, що для одного і того самого вхідного повідомлення результат гешування завжди є однаковим. Ця властивість є принциповою для використання геш-функцій у задачах контролю цілісності даних, автентифікації та побудови цифрових підписів [50].

Криптографічна геш-функція повинна забезпечувати відсутність кореляції між вхідними і вихідними бітами. Бажано, щоб будь-який вхідний біт впливав на декілька вихідних бітів, тобто функція повинна мати досить гарний лавинний ефект. Ця властивість полягає у тому, що незначна зміна вхідного повідомлення

має призводити до суттєвої та непередбачуваної зміни геш-значення. Зазвичай вважається, що зміна всього одного біта вхідних даних має викликати зміну приблизно половини бітів вихідної геш-послідовності. Наявність лавинного ефекту ускладнює встановлення статистичних залежностей між вхідними та вихідними даними, що забезпечує стійкість до методів криптоаналізу, спрямованих на виявлення закономірностей у структурі перетворень [12, 22, 54].

Властивість геш-функції ускладненої стійкість до зіткнень полягає у тому, що знаходження будь-яких двох вхідних повідомлень $(\mathrm{M}_1;\mathrm{M}_2)$ таких, що їхні геш-значення $hash(\mathrm{M}_1)$ і $hash(\mathrm{M}_2)$ відрізнялися б у невеликій кількості біт, є практично складним завданням.

Властивості часткової стійкості до обчислення прообразу та локальної однобічності, які полягають у тому, що відновлення частини вхідного повідомлення є такою же складною задачею, як і відновлення всього повідомлення. Більш того, навіть якщо відома частина вхідного повідомлення, відновлення залишку повідомлення також є трудомісткою задачею [22, 54].

Більшість геш-функцій мають ітеративні конструкції та використовують певну функцію ущільнення з фіксованими входами. Для забезпечення можливості гешування повідомлень довільної довжини вхідні дані представляються у вигляді послідовності блоків $\mathrm{M} = \{ m_1, m_2,...,m_L \}$ фіксованого розміру. Процес гешування здійснюється ітеративно шляхом послідовної обробки кожного блоку m_i функцією ущільнення з формуванням проміжних результатів гешування на кожному кроці, які позначають як h_i. Результатом виконання останньої L-ої ітерації є остаточне геш-значення повідомлення, що подається на вихід геш-функції. На практиці в багатьох геш-функціях передбачено, що це фінальне значення h_L може підлягати додатковій обробці (етапу фіналізації) з використанням спеціального перетворення $g(h_L)$, яке може включати додаткові раунди перемішування або обрізку стану до потрібної довжини геш-значення [51].

Зазначені властивості та вимоги формують базові критерії криптографічної стійкості геш-функцій і визначають можливість їх використання у практичних криптографічних застосуваннях. У контексті малоресурсних середовищ особливого значення набуває вибір конструкції геш-функції та спосіб організації ітеративного процесу гешування, оскільки саме ці фактори визначають можливість забезпечення необхідних криптографічних властивостей за умов обмежених ресурсів. На практиці для цього широко використовуються як класичні конструкції геш-функцій, зокрема ітеративні конструкції типу Меркла–Дамґарда [51, 55] на основі блокових шифрів [21, 56], так і більш сучасні підходи, такі як конструкція типу «губка» [57], які завдяки своїй універсальності та гнучкості знайшли застосування у малоресурсній криптографії.

1.3 Огляд існуючих підходів до побудови геш-функцій

Сучасні криптографічні геш-функції, незважаючи на різноманіття конкретних алгоритмічних реалізацій, у переважній більшості ґрунтуються на обмеженій кількості базових конструктивних підходів. Вибір конструкції гешування визначає принципи ітеративної обробки даних, організацію внутрішнього стану, спосіб формування геш-значення, а також істотно впливає на криптографічні властивості геш-функції.

У загальному випадку конструкція геш-функції задає математичну та структурну модель перетворення повідомлення довільної довжини у геш-значення фіксованої довжини. Вона визначає, яким чином вхідні дані розбиваються на блоки, як формуються проміжні геш-значення, яким є взаємозв’язок між послідовними ітераціями гешування та які внутрішні перетворення застосовуються до даних. Незалежно від конкретної реалізації, саме конструкція є тим рівнем абстракції, на якому закладаються як криптографічні, так і технічні властивості геш-функції.

Аналіз наукових публікацій [4-10, 16-19, 51-74] показує, що в контексті криптографічного гешування, зокрема орієнтованого на використання в умовах обмежених обчислювальних ресурсів, домінують дві базові конструкції:

ітеративна конструкція Меркла–Дамґарда та конструкція типу «губка» (sponge construction). Зазначені конструкції покладено в основу більшості класичних і сучасних геш-функцій, у тому числі призначених для застосування в середовищах із жорсткими обмеженнями на обсяг пам’яті, апаратні витрати та енергоспоживання.

1.3.1 Конструкція Меркла-Дамґарда

Конструкція Меркла–Дамґарда є однією з базових та найбільш поширених ітеративних схем побудови криптографічних геш-функцій. Теоретичні основи цієї конструкції були закладені у працях Р. Меркла та І. Дамґарда, у яких показано, що за умови використання коректної схеми доповнення повідомлення та функції ущільнення, стійкої до колізій, побудована на її основі геш-функція також зберігає властивість стійкості до колізій [51, 53]. Саме цей результат став фундаментом для подальшого розвитку теорії криптографічних геш-функцій.

Відповідно до цього підходу вхідне повідомлення m довільної довжини попередньо доповнюється за визначеним правилом з метою приведення його довжини до кратної розміру блоку. Після цього доповнене повідомлення розбивається на послідовність блоків фіксованої довжини $(m_1, m_2, .., m_n)$, де кожен блок m_i має однакову бітову розрядність. Процес доповнення є невід’ємною складовою конструкції, оскільки він запобігає виникненню окремих класів атак, пов’язаних із неоднозначністю подання вхідних даних.

Основним елементом конструкції Меркла–Дамґарда є функція ущільнення f, що приймає на вхід поточний блок повідомлення та проміжне геш-значення (ланцюгове значення) і формує нове проміжне геш-значення. Обчислення геш-значення здійснюється ітеративно за формулою (1.2):

$$h_i = f(h_{i-1}, m_i), \qquad i = 1,\ 2, ..., n, \tag{1.2}$$

де h_0 – фіксоване початкове значення (вектор ініціалізації геш-функції, IV), а h_n – остаточне геш-значення повідомлення. Кожне значення h_i є проміжним

результатом гешування та використовується як вхідний параметр для наступної ітерації. Загальну схему ітеративного процесу обчислення геш-функції згідно з конструкцією Меркла–Дамґарда представлено на рисунку 1.1.

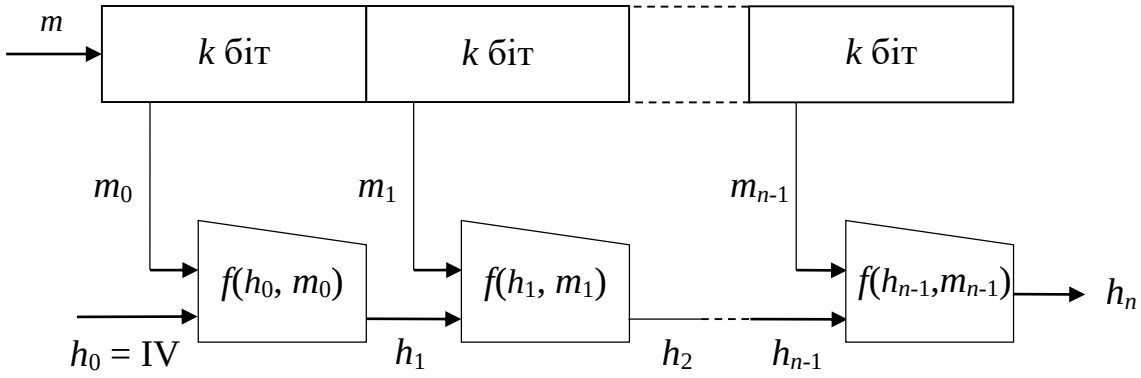

Рисунок 1.1 – Конструкція Меркла–Дамґарда [51]

Ітеративний характер конструкції забезпечує можливість послідовної обробки повідомлень довільної довжини без необхідності зберігання всього повідомлення в пам’яті одночасно. Завдяки цьому конструкція Меркла-Дамґарда широко застосовувалася при розробці класичних криптографічних геш-функцій, зокрема MD4, MD5, SHA-1 та SHA-2 [50], для яких функція ущільнення реалізується з використанням різних внутрішніх перетворень.

Важливою особливістю конструкції (1.2) є її гнучкість щодо вибору функції ущільнення. За умови збереження базових криптографічних вимог до цієї функції можливе створення різних модифікацій геш-функцій, у тому числі орієнтованих на зменшення складності внутрішніх перетворень[53, 58]. Саме ця властивість зумовила широке використання конструкції Меркла–Дамґарда та її модифікацій у малоресурсній криптографії [8, 19, 59, 60].

На жаль, конструкція Меркла–Дамґарда має низку властивостей, що негативно впливають на її безпеку. Зокрема, у цій схемі можлива атака довжинного доповнення, а для повідомлень значної довжини складність атаки на другий прообраз є суттєво нижчою за повний перебір. Крім того, побудова множинних колізій (мультиколізій) потребує лише незначного додаткового обчислювального ресурсу [25, 61, 62]. Існує також так звана атака доповнення,

коли, маючи геш-значення для невідомого повідомлення N, можна обчислювати геш-значення повідомлень, пов’язаних з N, хоча саме повідомлення N залишається невідомим [53, 58]. Через ці структурні слабкості в практичних реалізаціях застосовують різні модифікації класичної побудови. Зокрема, ширококанальна адаптація (wide-pipe construction) Стефана Лакса [58] передбачає збільшення внутрішнього розміру стану геш-функції, що підвищує стійкість до атак на другий прообраз і мультиколізій. Інші підходи передбачають використання додаткових схем доповнення та модифікацій функції ущільнення, спрямованих на зменшення ефективності атак доповнення і підвищення криптографічної стійкості алгоритму.

Поширеним підходом щодо реалізації функції ущільнення в конструкції Меркла–Дамґарда є використання блокових шифрів із застосуванням схем Девіса–Меєра, Матіаса–Меєра–Осеаса [55] та Міягучі–Пренеля [56], що представлені на рисунку 1.2.

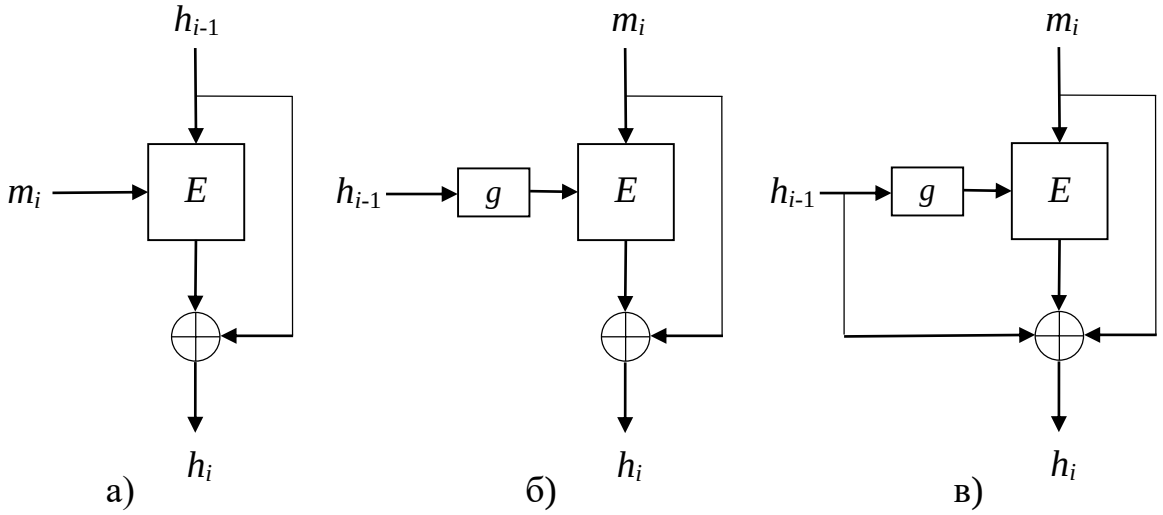

Рисунок 1.2 – Схеми побудови геш-функцій з використанням блокових шифрів:

а) Девіса–Меєра; б) Матіаса–Меєра–Осеас; в) Міягучі–Пренеля

Вказані схеми визначають, як комбінувати вихід шифру та блок вхідних даних для отримання нового стану геш-функції. У схемі Девіса–Меєра функція ущільнення реалізується шляхом зашифрування попереднього геш-значення h_{i-1} заданим блоковим шифром E_i, що як ключ використовує блок повідомлення m_i,

та подальшого виконання операції додавання за модулем два між отриманим результатом і значенням h_{i-1} для отримання наступного геш-значення h_i:

$$h_i = E_{m_i}(h_{i-1}) \oplus h_{i-1}. \quad (1.3)$$

Недоліком схеми (1.3) є те, що незважаючи на безпечний блоковий шифр, є можливість появи так званих «нерухомих точок», коли для деякого блоку повідомлення m існує значення $h = E_m^{-1}(0)$, що задовольняє умову $E_m(h) \oplus h = h$, тобто функція ущільнення повертає те саме геш-значення [56].

У схемі Матіаса–Меєра–Осеаса враховано недолік конструкції Девіса–Мейєра. Тут блок повідомлення m_i використовується як вхідні дані для функції шифрування, а попереднє геш-значення h_{i-1} після додаткового перетворення g застосовується як ключ для блокового шифру E. Наступне геш-значення обчислюється шляхом поєднання результату зашифрування з самим блоком повідомлення операцією додавання за модулем два (1.4):

$$h_i = E_{g(h_{i-1})}(m_i) \oplus m_i, \quad (1.4)$$

де $g(h_{i-1})$ – функція перетворення, що змінюється залежно від реалізації. Блоковий шифр E може використовувати ключ, довжина якого відрізняється від дожини блока, тобто не збігається з розміром геш-значення. Цю проблему усуває функція g, що перетворює h_{i-1} у ключ відповідного розміру.

Схема Міягучі–Пренеля є подальшим розвитком конструкції Матіаса–Меєра–Осеаса. Відмінність від попередньої схеми полягає в тому, що до зашифрованого блоку повідомлення додається не лише блок повідомлення, а й попереднє геш-значення (1.5):

$$h_i = E_{g(h_{i-1})}(m_i) \oplus h_{i-1} \oplus m_i. \quad (1.5)$$

Поєднання (1.5) гарантує, що кожне наступне геш-значення залежить одночасно від всіх трьох компонентів – ключа, блоку повідомлення та

попереднього геш-значення, що підвищує дифузію та стійкість функції ущільнення до криптографічних атак, зокрема атак на «нерухомі точки» та селекційні колізії [63].

1.3.2 Конструкція «губка»

Конструкція «губка» (sponge-конструкція) є сучасною схемою для побудови криптографічних геш-функцій і широко використовується в засобах, орієнтованих на застосування в умовах обмежених обчислювальних ресурсів. На відміну від класичної ітеративної конструкції Меркла–Дамґарда, sponge-конструкція дозволяє формувати вихідні дані довільної довжини та забезпечує більш гнучке керування рівнем криптографічної стійкості за рахунок параметрів внутрішнього стану [57].

Конструкція «губка» є однією з найбільш розповсюджених сучасних схем для побудови геш-функцій. Існує дві класичні sponge-конструкції:

– P-губка, де в ролі базової функції використовується відома перестановка;
– T-губка, що ґрунтується на випадковій функції й переважно застосовується як теоретична модель для аналізу безпеки.

На практиці, в багатьох криптографічних алгоритмах, зокрема у малоресурсних геш-функціях, застосовується саме P-губка, оскільки вона дозволяє реалізувати компактне внутрішнє перетворення без необхідності використання складних функцій ущільнення [64].

«Губка» є ітеративною конструкцією, яка оперує внутрішнім станом фіксованого розміру біт $b = r + c$, для якого r – бітова швидкість, що визначає обсяг даних, які обробляються за одну ітерацію, а c – ємність, яка визначає рівень криптографічної стійкості конструкції [11, 57]. Узагальнену структуру конструкції «губка» наведено на рисунку 1.3.

Робота «губки» відбувається у дві фази: «вбирання» (absorb) та «вичавлювання» (squeeze). Фаза «вбирання» призначена для поступового інтегрування вхідного повідомлення у внутрішній стан «губки», а фаза

«вичавлювання» формує вихідну послідовність довільної довжини на основі цього стану. На етапі «вбирання» вхідне повідомлення M за необхідності доповнюється відповідно до визначеної схеми доповнення (padding), після чого розбивається на блоки довжиною по *r* біт. Внутрішній стан «губки» на початку ініціалізується нульовими бітами. Геш-значення обчислюється шляхом послідовного «вбирання» доповненого повідомлення та «вичавлювання» внутрішнього стану.

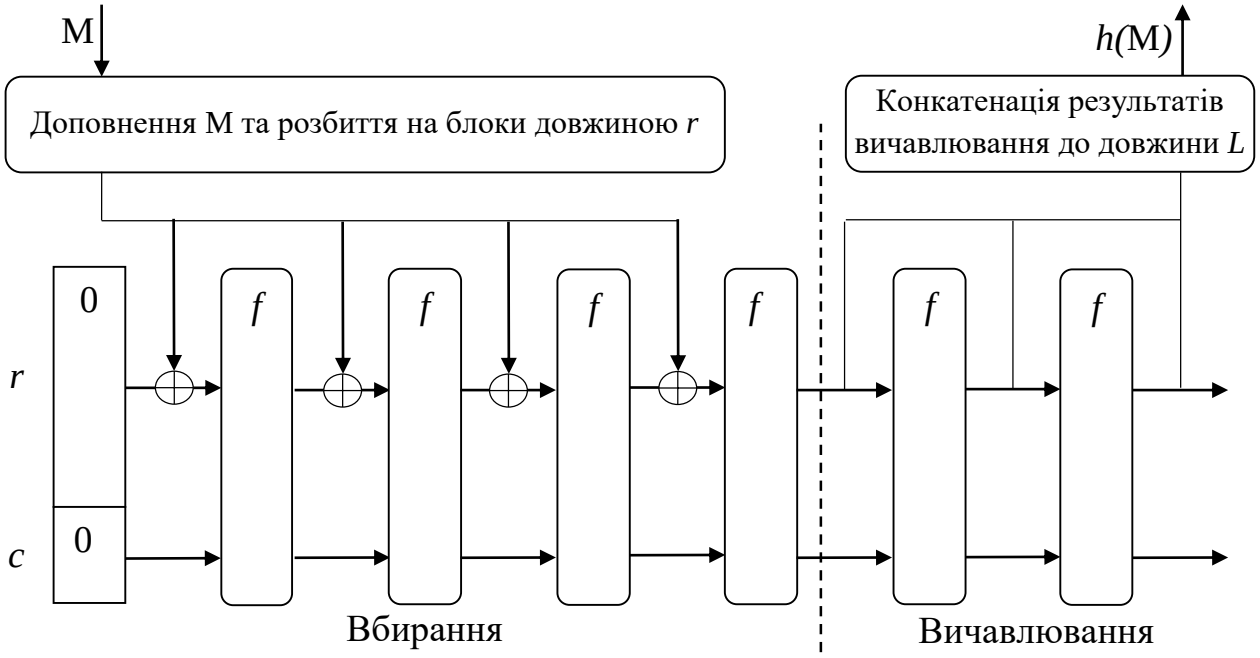

Рисунок 1.3 – Конструкція «губка» [57, 65]

Фаза «вбирання» включає кроки:

1) перші *r* біт внутрішнього стану комбінуються з черговим блоком доповненого повідомлення за допомогою операції додавання за модулем два;
2) до повного внутрішнього стану застосовується базове перетворення *f*.

Ці операції повторюються для всіх блоків повідомлення M, забезпечуючи поступову інтеграцію вхідних даних у внутрішній стан.

Після завершення фази «вбирання» виконується фаза «вичавлювання», у ході якої до внутрішнього стану багаторазово застосовується перетворення *f*, а перші *r* біт стану на кожній ітерації подаються на вихід. Процес повторюється

доти, доки не буде сформовано геш-значення необхідної довжини L, яке утворюється шляхом конкатенації отриманих r-бітових блоків.

Важливою особливістю sponge-конструкції є те, що останні c біт внутрішнього стану безпосередньо не надходять на вихід і залежать від вхідних даних лише опосередковано, що підвищує криптографічну стійкість конструкції. Зокрема, складність знаходження колізій у конструкції «губка» не менша ніж $2^{c/2}$, тоді як атаки на перший та другий прообраз вимагають не менше 2^c операцій [57, 65]. Крім того, ймовірність відрізнити випадкову «губку» від ідеального випадкового оракула обмежується величиною, пропорційною $\frac{N(N+1)}{2^{c+1}}$, де N – кількість застосувань перетворення f [15]. Це означає, що при практично доцільному виборі параметрів, коли $c \geq 2L$ для вихідного геш-значення довжиною L, конструкція «губки» забезпечує рівень криптографічної стійкості, еквівалентний ідеальній моделі випадкового оракула [64].

З точки зору малоресурсної криптографії sponge-конструкція має низку суттєвих переваг. Вона не потребує окремої функції ущільнення, має просту та регулярну структуру, дозволяє гнучко змінювати співвідношення між швидкістю та рівнем стійкості, а також забезпечує ітеративну обробку повідомлень без збереження всього вхідного масиву в пам’яті. Це робить конструкцію «губка» привабливою для використання в середовищах з обмеженими обчислювальними та енергетичними ресурсами.

Водночас sponge-конструкція має й певні недоліки. Основним із них є потреба у значних апаратних витратах на збереження внутрішнього стану. Оскільки для досягнення високого рівня криптографічної стійкості параметр ємності c повинен мати достатньо велике значення, загальний розмір проміжного стану $b = r + c$ суттєво перевищує довжину підсумкового геш-значення. Це вимагає використання більшої кількості пам’яті або регістрів для збереження внутрішніх даних під час обчислень, що може бути критичним для найбільш обмежених малоресурсних пристроїв. Крім того, велике значення параметра c при фіксованому розмірі стану b призводить до зменшення швидкості обробки

даних, а загальна ефективність реалізації значною мірою залежить від вибору базової перестановки *f*, яка повинна забезпечувати належний рівень дифузії та нелінійності при мінімальних обчислювальних витратах.

Підсумовуючи огляд підходів до побудови геш-функцій, слід зазначити, що попри їхню універсальність, пряма адаптація класичних схем до умов малоресурсного гешування супроводжується низкою компромісів. Конструкція Меркла–Дамґарда зазвичай вимагає реалізації достатньо складної функції ущільнення, особливо якщо в якості функції ущільнення використовується блоковий шифр за однією з схем (див. рис. 1.2), що призводить до зростання апаратних витрат. У свою чергу конструкція «губка», незважаючи на гнучкість та можливість формування геш-значень довільної довжини, передбачає використання внутрішнього стану, розмір якого, як правило, перевищує довжину вихідного геш-значення. Необхідність збереження та обробки такого розширеного внутрішнього стану, особливо при реалізації геш-функцій із високим рівнем криптографічної стійкості та великою довжиною геш-значення, також призводить до збільшення апаратних і енергетичних витрат.

Разом із тим, у практичних реалізаціях геш-функцій, орієнтованих на обмежені ресурси, базові конструктивні підходи зберігаються, проте зазнають суттєвих модифікацій. Зменшення складності досягається шляхом спрощення внутрішніх перетворень, скорочення розміру стану, використання елементарних логічних операцій або регістрів зсуву, а також за рахунок спеціалізованих схем обробки даних. Саме такі рішення дозволяють адаптувати зазначені конструкції до вимог малоресурсних середовищ.

1.4 Геш-функції малоресурсної криптографії

Геш-функції малоресурсної криптографії орієнтовані на використання в умовах жорстких обмежень апаратних ресурсів, енергоспоживання та обсягу внутрішньої пам’яті, зберігаючи при цьому визначений рівень криптографічної стійкості. У переважній більшості випадків такі алгоритми базуються на класичних конструкціях гешування, насамперед на ітеративній конструкції

Меркла–Дамґарда та конструкції «губка». Зменшення апаратних витрат у цих геш-функціях досягається шляхом спрощення функції ущільнення, використання простих внутрішніх перетворень або альтернативних схем обробки стану. При цьому архітектура геш-функції зберігає загальні принципи відповідної конструкції, а компроміс між продуктивністю, рівнем безпеки та апаратною складністю визначається параметрами конкретної реалізації.

У дослідженні [59] запропоновано алгоритм ARMADILLO як універсальну криптографічну конструкцію загального призначення, орієнтовану на застосування в умовах жорстких апаратних обмежень, зокрема у RFID-мітках та бездротових сенсорних мережах. Алгоритм може використовуватися для гешування даних, побудови механізмів автентифікації, генерації псевдовипадкових послідовностей, а також як основа для потокового шифрування. ARMADILLO базується на конструкції Меркла–Дамґарда зі спеціалізованою функцією ущільнення на основі бітових перестановок, залежних від даних, та підтримує довжини внутрішнього стану від 80 до 256 біт. Згідно з аналізом [66], перша версія ARMADILLO1 містить фатальну вразливість, що дозволяє пасивному нападнику відновити секретний ключ у поліноміальному часі. Модифікація ARMADILLO2 була запропонована з метою усунення цих недоліків шляхом спрощення внутрішньої структури та використання компактнішої параметризованої перестановки, що дозволило зменшити апаратну складність реалізації. Проте у [67] продемонстровано, що складність атак на другий прообраз є нижчою за заявлений рівень безпеки ARMADILLO2, причому вразливість має структурний характер і не залежить від конкретного вибору внутрішніх перестановок.

Представниками геш-функцій, побудованих з використанням блокових шифрів є DM-PRESENT, H-PRESENT, C-PRESENT та Lesamnta-LW.

У роботі [8] запропоновано сімейство малоресурсних геш-функцій DM-PRESENT, H-PRESENT та C-PRESENT, побудованих на основі малоресурсного блокового шифру PRESENT [10], орієнтованих на використання в умовах жорстких апаратних обмежень, зокрема в RFID-мітках і сенсорних мережах.

PRESENT є SP-мережею з довжиною блоку 64 біти, підтримкою ключів довжиною 80 або 128 біт та 31 раундом зашифрування, що забезпечує дифузію при мінімальних апаратних витратах. Геш-функція DM-PRESENT реалізована на основі схеми Девіса–Маєра, де 64-бітний внутрішній стан оновлюється одним циклом зашифрування PRESENT з подальшим додаванням за модулем два з попереднім станом.

У роботах [68, 69] показано, як складність знаходження колізій істотно знижується при зменшенні кількості раундів функції ущільнення і складає 2^{29} для DM-PRESENT-80, при використанні функції ущільнення з 12 раундами. Дослідники також показали вразливість такої функції ущільнення до диференційного аналізу, а також зменшення складності атак на другий прообраз, що робить можливим її застосування переважно із незначними вимогами до безпеки.

З метою підвищення криптографічної стійкості в [8] запропоновано розширені варіанти H-PRESENT-128 та C-PRESENT-192 на основі схеми Hirose [20], які формують геш-значення більшої довжини шляхом паралельного оновлення кількох ланцюгових змінних. Разом із тим дослідження [8, 60] вказують, що збільшення кількості викликів базового шифру та розміру внутрішнього стану супроводжується значним зростанням апаратних витрат і не усуває структурних обмежень, пов’язаних із використанням 64-бітного блокового шифру як криптографічної основи.

Геш-функція Lesamnta-LW орієнтована на застосування у малоресурсних середовищах, зокрема на восьмибітних мікроконтролерах, RFID-пристроях та вбудованих системах з обмеженою оперативною пам’яттю [19]. Lesamnta-LW використовує модифіковану конструкцію Меркла-Дамґарда з wide-pipe підходом, а функція ущільнення побудована на спеціалізованому блоковому шифрі з довжиною блоку 256 біт і ключем 128 біт, реалізованому у вигляді 4-гілкової узагальненої мережі Фейстеля (GFN) з AES-подібними перетвореннями SubBytes і MixColumns. Заявлений рівень криптографічної стійкості становить приблизно 2^{120} операцій для колізій та атак на другий прообраз. Проте апаратна

реалізація потребує близько 8240 логічних елементів GE, що свідчить про умовну належність Lesamnta-LW до класу малоресурсних геш-функцій. Відносно складна внутрішня структура та використання AES-подібних компонентів обмежують її застосування у сценаріях із жорсткими апаратними обмеженнями та роблять її більш доцільною для середньоресурсних вбудованих систем.

Переважна більшість сучасних малоресурсних геш-функцій в основі використовують конструкцію «губка». Геш-функція SPONGENT використовує конструкцію «губка» у поєднанні з компактною побітовою SP-мережею, що включає 4-бітні S-блоки, прості перестановки та операції XOR . Як і більшість геш-функцій на основі конструкції «губка» у [9] пропонують декілька варіантів реалізації SPONGENT залежно від трійки параметрів (n, c, r), де n – довжина геш-значення, де c – ємність конструкції, r – бітова швидкість конструкції. Різні варіанти SPONGENT-$n/c/r$ охоплюють довжини геш-значень 88, 128, 160, 224 та 256 біт. Стійкість до колізій визначається ємністю c і становить $2^{c/2}$, тоді як стійкість до атак на прообрази залежить як від c, так і від r. У роботі [9] також показано, що навіть при зменшеній кількості раундів внутрішньої перестановки кількість активних S-блоків швидко зростає, що робить практично неможливими диференційні та лінійні атаки на повну кількість раундів. Апаратна реалізація SPONGENT потребує лише 738–2228 GE залежно від параметрів, що досягається за рахунок високосеріалізованої архітектури з вузьким трактом даних та багаторазовим використанням функціональних блоків. Такий підхід істотно зменшує апаратні витрати, однак призводить до зниження пропускної здатності гешування.

У роботі [6] запропоновано сімейство малоресурсних геш-функцій PHOTON, побудованих на основі розширеної sponge-конструкції з безключовою AES-подібною внутрішньою перестановкою. Подібно до SPONGENT геш-функція представлена декількома варантами PHOTON-$n/r/r$`: PHOTON-80/20/16, PHOTON-128/16/16, PHOTON-160/36/36, PHOTON-224/32/32 та PHOTON-

256/32/32, які відрізняються довжиною геш-значення n, а також параметрами бітової швидкості r (швидкість процесу «вбирання») та $r`$ (швидкість процесу «вичавлювання»). Внутрішня перестановка PHOTON містить регулярну SP-мережу та виконується протягом 12 раундів, кожен з яких включає чотири перетворення: AddConstants, SubCells, ShiftRows та MixColumnsSerial [13, 45]. Стан подається у вигляді матриці комірок, а в шарі SubCells використовується компактний S-блок, що забезпечує нелінійність при мінімальних апаратних витратах. Ключовим для малоресурсності є шар MixColumnsSerial, де лінійне перемішування реалізовано з багаторазовим використанням обмеженої комбінаційної логіки.

Рівень криптографічної стійкості геш-функції PHOTON визначається вибором параметрів внутрішнього стану та відповідає загальним особливостям sponge-конструкцій. Для рекомендованих параметрів сімейство PHOTON забезпечує стійкість до колізій і атак на знаходження першого та другого прообразів на рівні, узгодженому з довжиною геш-значення [6]. У дослідженні [70] показано, що при використанні *cube attack* до PHOTON зі зменшеною до трьох раундів внутрішньою перестановкою можливе часткове відновлення вхідного повідомлення (близько половини бітів), що підтверджує чутливість криптографічних властивостей алгоритму до кількості раундів, але не знижує заявлений рівень стійкості повної версії геш-функції.

Завдяки поєднанню компактної внутрішньої перестановки та гнучкої sponge-конструкції PHOTON, разом із геш-функціями SPONGENT та Lesamnta-LW, включена до міжнародного стандарту ISO/IEC 29192-5 [30], який визначає вимоги та рекомендації щодо геш-функцій для малоресурсних криптографічних застосувань. Для різних варіантів PHOTON наведено реалізації з апаратною складністю від приблизно 865 до 4362 GE, залежно від обраних параметрів і типу реалізації [6]. Найбільший варіант PHOTON потребує близько 4362 GE у паралельній реалізації, що забезпечує максимальну пропускну здатність, при цьому внутрішня перестановка дозволяє економити близько 8 % апаратних ресурсів порівняно з AES. Послідовна реалізація для обчислення 256-бітного

геш-значення потребує лише 2177 GE, проте характеризується значно нижчою пропускною здатністю.

На відміну від геш-функцій SPONGENT та PHOTON, у яких внутрішня перестановка побудована на основі S-блокових перетворень і лінійних дифузійних шарів, сімейство малоресурсних геш-функцій QUARK [4, 5] використовує альтернативний підхід, заснований на регістрах зсуву. QUARK реалізовано у вигляді sponge-конструкції, базовим перетворенням якої є перестановка P, що працює над внутрішнім станом ширини $b = r + c$. Перестановка P інтерпретує внутрішній стан геш-функції у вигляді двох регістри зсуву з нелінійним зворотним зв'язком (NFSR X та Y) однакової довжини $b/2$, а також містить допоміжний регістр зсуву з лінійним зворотнім зв'язком (LFSR L) малої довжини. Всі три регістри поєднані булевими функціями f, g та h.

У межах одного такту спочатку обчислюється комбінуюча функція h (з урахуванням вибірок бітів зі станів X, Y та керуючого біта від L), після чого для X та Y виконується зсув після чого регістри отримують новий вхідний біт як суму за модулем два від виходу відповідної функції зворотного зв'язку (f або g) та значення h. Допоміжний LFSR оновлюється незалежно і використовується як додатковий керуючий елемент у функції h для усунення симетрій між тактами та запобігання атакам типу *slide resynchronization*, що використовують зсув (ресинхронізацію) внутрішнього стану в регістрових конструкціях [5].

Розробниками запропоновано три варіанти QUARK: U-QUARK, D-QUARK та S-QUARK, які відрізняються параметрами sponge-конструкції (r, c, b) та цільовим рівнем безпеки. Для U-QUARK задано $r = 8$, $c = 128$, $b = 136$, для D-QUARK – $r = 16$, $c = 160$, $b = 176$, для S-QUARK – $r = 32$, $c = 224$, $b = 256$; при цьому кількість тактів внутрішньої перестановки P у кожному випадку визначена як $4b$, що відповідає 544, 704 та 1024 тактам відповідно. Згідно з оцінками розробників, найбільш оптимізовані за апаратними витратами реалізації U-/D-/S-QUARK потребують близько 1379/1702/2296 GE [5].

У їх роботі [5] також наведено попередній криптоаналіз перестановки P, де показано можливість застосування методів *cube testers*/*cube attack*-подібних

підходів та диференційного аналізу лише для варіантів QUARK з суттєво меншою кількістю тактів, причому найкращі результати охоплюють близько п’ятої частини від повної кількості тактів (близько ~21% від 4*b*). У дослідженні [71] показано, що використання *conditional differential attack* (CDA), дозволяє підвищити межу успішного аналізу подібних спрощених варіантів QUARK до 157/171/292 тактів для U/D/S-QUARK відповідно. Водночас у [71] підкреслено, що отримані результати стосуються насамперед внутрішньої перестановки *P* в моделі випадкового початкового стану та не переходять безпосередньо у практичні атаки на повну sponge-геш-функцію QUARK для рекомендованих параметрів.

Геш-функція GLUON, запропонована в [72], побудована на використанні регістрів зсуву з переносом (FCSR), які раніше застосовувалися у потокових шифрах F-FCSR-v3 та X-FCSR-v2. Запропоновано три варіанти з довжиною геш-значення 128, 160 і 224 біти; для рекомендованих параметрів заявлена стійкість до колізій і атак на знаходження другого прообразу узгоджується з цими довжинами, а стійкість до знаходження першого прообразу відповідає повній довжині геш-значення [72]. Ключовою особливістю реалізації є паралельне завантаження даних у FCSR, що підвищує пропускну здатність гешування, але вимагає більших апаратних витра: 2071 GE для 128-бітної реалізації, 2799 GE – для 160-бітної та 4724 GE – для 224-бітної реалізації.

У роботі [7] запропоновано геш-функцію SPN-Hash, що ґрунтується на поєднанні SPN-перестановки та JH-режиму роботи, який є модифікацією конструкції «губка». У цьому режимі для кожного блока повідомлення виконується XOR з внутрішнім станом як перед, так і після застосування перестановки, що усуває прості залежності між проміжними станами та підвищує стійкість до атак типу *meet-in-the-middle* на другий прообраз. Внутрішня перестановка SPN-Hash має AES-подібну структуру і складається з 10 раундів; у кожному раунді послідовно застосовуються байтові S-блоки, лінійний шар перемішування з високими дифузійними властивостями та додавання раундових констант до стану для запобігання симетріям і фіксованим

точкам. У [7] також наведено аналіз стійкості до атак на перший та другий прообрази, а також *rebound*-подібних атак, що є характерними для геш-функцій, що використовують AES-подібні перетворення.

SPN-Hash підтримує довжини геш-значень 128, 256 та 512 біт і розроблена з урахуванням ефективної програмної реалізації та помірних апаратних витрат. Зокрема, для 256-бітного варіанта автори наводять оцінку апаратної складності на рівні близько 4625 GE. Разом із тим, використання байтових S-блоків і багато раундової SP-мережі як внутрішньої перестановки обумовлює вищі апаратні витрати порівняно з найбільш компактними малоресурсними геш-функціями, що визначає доцільність застосування SPN-Hash у середовищах із помірними, а не мінімальними апаратними обмеженнями.

У дослідженні [18] запропоновано сімейство малоресурсних геш-функцій HVH, побудованих за конструкцією «губка», де внутрішню перестановку реалізовано на основі «легкого» блокового шифру VH. Блоковий шифр VH [73] є SP-мережею з 64-бітним розміром блока, у якій застосовано подвійне псевдовипадкове перетворення, та передбачено підтримку довжин ключа в діапазоні 64–128 біт. HVH підтримує п’ять варіантів довжини геш-значення (88…256 біт), що дає змогу обирати компроміс між апаратними витратами та цільовим рівнем стійкості. Автори [18] позиціонують HVH як криптографічну геш-функцію для RFID та бездротових сенсорних мереж (WSN), де визначальними є обмеження на енергоспоживання, площу апаратної реалізації та продуктивність на малопотужних платформах. Тому HVH демонструє високу програмну швидкодію на восьмибітних мікроконтролерах, де при обчисленні 88-бітних геш-значень має пропускну здатність близько 1,47 Мбіт/с. Апаратна складність HVH при довжині геш-значення 256 біт становить 2009 GE. За твердженням розробників, геш-функції HVH забезпечують достатній запас стійкості до лінійного та диференційного криптоаналізу, а також до стандартних атак на геш-функції (колізії, перший та другий прообраз) у межах прийнятої моделі.

Геш-функція THF (Tiny Hash Function) [16] побудована за конструкцією «губка» з фіксованим внутрішнім станом довжиною 64 біт. На відміну від класичної sponge-конструкції, у THF перед фазою «вбирання» виконується додаткова обробка кожного блока повідомлення із застосовунням операцій зсуву та XOR зі згенерованими раундовими константами RC. Під час «вбирання» змішування також реалізується переважно через зсуви вліво та XOR. Обробка 64-бітних блоків організована у дві стадії: спочатку блок ділиться на 12 секцій і для них виконується 8 раундів перетворень, далі блок переформовується та ділиться на 6 секцій, після чого виконується ще 2 раунди. Надалі автори пропонують шляхом «вичавлювання» формувати геш-значення довжиною 64, 128 та 256 біт з використанням реалізації з фіксованим внутрішнім станом 64 біти, при цьому оцінена апаратна складність реалізації становить близько 1296 GE. Використання постійного малорозрядного внутрішнього стану спрощує реалізацію та зменшує апаратні витрати, однак може обмежувати досяжний рівень криптографічної стійкості, тому потребує окремого підтвердження через криптоаналіз. У дослідженні [16] підкреслюють застосування внутрішніх констант, що ускладнює аналіз і зворотне відновлення внутрішнього стану, а також наводять експериментальну оцінку дифузії (лавинного ефекту) за відстанню Геммінга із середнім значенням близько 74%, що свідчить про суттєву зміну виходу при малих змінах вхідних даних, тоді як стійкість до колізій, атак на перший та другий прообраз та інших атак на геш-функції не досліджувалася.

У статті [74] запропоновано геш-функцію Hash-One, що реалізує sponge-конструкцію на основі двох нелінійних регістрів зсуву зі зворотним зв’язком (NFSR) P та Q, які разом утворюють 161-бітний внутрішній стан з параметрами «губки» (n=160, c=160 та r=1). Перестановка геш-функції виконується послідовним зсувом регістрів із формуванням нових бітів стану на основі трьох булевих функцій P_f, Q_f та L_f, причому їх апаратну реалізацію оптимізовано шляхом використання лише NAND-вентилів.

Перестановка в Hash-One виконується ітеративно для кожного вхідного біта: для першого та останнього біта повідомлення передбачено 324 цикли

оновлення внутрішнього стану, тоді як для всіх проміжних бітів – 162 цикли, що спрямовано на посилення дифузії під час обробки повідомлення. Після завершення обчислень геш-функція формує вихідне геш-значення довжиною 80 або 160 біт.

За заявами авторів, параметр ємності c=160 відповідає орієнтовному рівню стійкості 2^{80} до колізій та 2^{160} до атак на перший прообраз. Дослідження [74] підтверджують криптографічну стійкість Hash-One результатами аналізу диференційної та алгебраїчної атак, а також оцінюванням можливості здійснення *cube attack* шляхом тестування внутрішньої перестановки на наявність статистичних відхилень, які потенційно могли б бути використані для побудови атаки. Малоресурсність апаратної реалізації Hash-One забезпечується простою внутрішньою перестановкою на основі двох регістрів NFSR та булевих функцій P_f, Q_f та L_f, апаратну реалізацію яких оптимізовано шляхом використання лише NAND-вентилів, завдяки чому досягається дуже низька апаратна складність геш-функції, що складає близько 1006 GE.

У свою чергу, геш-функція DeeR-Hash [17], подібно до Hash-One, використовує два нелінійні регістри зсуву зі зворотним зв’язком (NFSR) та додатково вводить лінійний регістр зсуву (LFSR) для покращення дифузії. На відміну від Hash-One, у DeeR-Hash відсутнє багаторазове ітерування внутрішнього перетворення для кожного біта повідомлення, а обробка здійснюється з бітовою швидкістю r = 2, тобто парами бітів, що зменшує енергоспоживання та обчислювальні витрати. Важливою особливістю перестановки DeeR-Hash є динамічна схема вибору зворотних зв’язків (tap positions) для оновлення стану регістрів, що реалізується через систему мультиплексорів і конфігураційних масивів та дозволяє формувати оновлення стану з мінімальним набором логічних елементів. DeeR-Hash підтримує формування геш-значень довжиною 80 та 160 біт, а заявлена апаратна складність становить близько 483 GE для 80-бітного варіанта та близько 984 GE для 160-бітного.

Для порівняння рівня криптографічної стійкості розглянутих малоресурсних геш-функцій у таблицях 1.1 та 1.2 наведено їх основні криптографічні характеристики, зокрема довжину геш-значення, оцінки стійкості до колізій, атак на перший і другий прообрази, а також інші характеристики отримані в роботах [32, 36]. Ці дані дозволяють простежити баланс між складністю апаратної реалізації та забезпеченням мінімальних вимог до безпеки, які відповідно до стандартів повинні складати не менше ніж $2^{L/2}$ для стійкості до колізій і 2^{L} для стійкості до атак на перший прообраз, де L – довжина геш-значення. Довжина геш-значення суттєво впливає як на криптографічну стійкість, так і на апаратну складність. Алгоритми з довжиною геш-значення 128 біт в більшості випадків демонструють апаратні витрати на рівні від 1000 до 2000 GE, тоді як для довших геш-значень (160-256 біт) складність часто зростає до 2000-4000 GE.

Таблиця 1.1 – Криптографічні характеристики геш-функцій основі конструкції Меркла–Дамґарда

Геш-функція	Довжина геш-значення (біт)	Структура	Мінімальна апаратна складність (GE)	Оцінка стійкості до		
				колізій	атак на перший прообраз	атак на другий прообраз
ARMADILLO [59]	80	Бітові перетворення на основі даних	2923	2^{40}	2^{80}	2^{80}
	128		4353	2^{64}	2^{128}	2^{128}
	160		5406	2^{80}	2^{160}	2^{160}
	192		6554	2^{96}	2^{192}	2^{192}
	256		8653	2^{128}	2^{256}	2^{256}
DM-PRESENT	64	Девіса-Мейєра	1600	2^{40}	2^{64}	–
H-PRESENT [8]	128		2330	2^{64}	2^{128}	–
C-PRESENT [8]	192		4600	2^{96}	2^{192}	–
Lesamnta-LW [60]	256	Блоковий шифр	8240	2^{120}	2^{256}	2^{256}

Таблиця 1.2 – Криптографічні характеристики геш-функцій основі конструкції «губка»

Геш-функція	Довжина геш-значення (біт)	Структура	Мінімальна апаратна складність (GE)	Оцінка стійкості до		
				колізій	атак на перший прообраз	атак на другий прообраз
U-Quark [4]	136	P-губка	1379	2^{64}	2^{128}	2^{64}
D-Quark [4]	160		1702	2^{80}	2^{160}	2^{80}
S-Quark [4]	224		2296	2^{112}	2^{224}	2^{112}
SPONGENT [9]	88	P-губка	738	2^{40}	2^{80}	2^{40}
	128		1060	2^{64}	2^{120}	2^{64}
	160		1329	2^{80}	2^{144}	2^{80}
	224		1728	2^{112}	2^{208}	2^{112}
	256		1950	2^{128}	2^{240}	2^{128}
PHOTON [6]	80	P-губка	865	2^{40}	2^{64}	2^{40}
	128		1122	2^{64}	2^{112}	2^{60}
	160		1396	2^{80}	2^{124}	2^{80}
	224		1736	2^{112}	2^{192}	2^{112}
	256		2177	2^{128}	2^{224}	2^{128}
GLUON [72]	128	T-губка	2071	2^{64}	2^{128}	2^{64}
	160		2800	2^{80}	2^{160}	2^{80}
	224		4724	2^{112}	2^{224}	2^{112}
SPN-Hash [7]	128	JH-губка	2777	2^{64}	2^{128}	–
	256		4625	2^{64}	2^{128}	–
HVH [18]	88	P-губка	857	2^{40}	2^{72}	2^{40}
	128		1145	2^{64}	2^{120}	2^{64}
	160		1385	2^{80}	2^{144}	2^{80}
	224		1769	2^{112}	2^{208}	2^{112}
	256		2009	2^{128}	2^{224}	2^{128}
THF [16]	128/256	P-губка	1296	–	–	–
Hash-One [74]	160	P-губка	1006	2^{80}	2^{160}	2^{80}
DeeR-Hash [17]	160	P-губка	984	2^{79}	2^{158}	2^{79}

Оптимальним є забезпечення високого рівня криптографічної стійкості при мінімальній складності апаратної реалізації (GE). У цьому контексті помітною є перевага конструкції «губка», над ітеративними структурами. Наприклад, для 128-бітних геш-функцій SPONGENT-128 демонструє стійкість до колізій на рівні 2^{64} при апаратній складності всього 1060 GE, що значно ефективніше, порівняно з GLUON-128 (2071 GE) або H-PRESENT (2330 GE), які забезпечують такий самий рівень безпеки, але вимагають для своєї реалізації майже вдвічі більше апаратних витрат.

Алгоритми DeeR-Hash та Hash-One у категорії обчислення геш-значень довжиною 160–192 біт демонструють найкращий компроміс між безпекою та апаратною складністю. Вони забезпечують стійкість до колізій на рівні 2^{79} та 2^{80} відповідно при апаратній складності 984 GE і 1006 GE. Найвищий рівень криптографічної стійкості у геш-функцій на основі блокових шифрів Lesamnta-LW та ARMADILLO, однак, якщо враховувати апаратні витрати, то SPONGENT та HVH забезпечують високу стійкість для 256-бітних гешів з мінімальними апаратними витратами, що складають 1950 GE та 2009 GE відповідно.

1.5 Особливості апаратної реалізації засобів гешування

Під час переходу від теоретичного аналізу конструкцій геш-функцій до їх практичного застосування вирішальним постає питання способу реалізації. У системах загального призначення гешування зазвичай виконують програмно, і основними критеріями ефективності є час виконання обчислень та потреби в пам’яті. Однак для IoT-вузлів, сенсорних мереж, RFID-систем і вбудованих контролерів, де ресурсні обмеження є визначальними, програмна реалізація часто виявляється або надто повільною, або енергетично невигідною, або навіть неможливою. Саме тому в малоресурсній криптографії апаратна реалізація геш-функцій розглядається як базовий практичний підхід, що забезпечує підвищення швидкодії обчислень, зменшення енергоспоживання пристрою та обмеження апаратної складності реалізації, що відповідає вимогам міжнародного стандарту малоресурсної криптографії ISO/IEC 29192-1 [29].

Усі процесори, що використовуються для реалізації криптографічних перетворень, за функціональним призначенням поділяються на універсальні та спеціалізовані. Універсальні процесори призначені для виконання широкого спектра задач і застосовуються в персональних комп’ютерах та загальних обчислювальних системах. Їх система команд орієнтована на підтримку різнорідних обчислень, зокрема операцій із цілими та дійсними числами, обробку масивів даних, керування пам’яттю та мультипрограмний режим роботи. Частина елементарних операцій, що використовуються в криптографічних алгоритмах (логічні операції, зсуви, додавання), виконується достатньо ефективно, однак операції, що забезпечують високий рівень дифузії та перемішування даних, як правило, реалізуються з істотними часовими та енергетичними витратами.

1.5.1 Універсальні та спеціалізовані засоби для виконання криптографічних перетворень

Виконання криптографічних перетворень здійснюється процесором, який за призначенням поділяють на два типи: універсальний та спеціалізований процесори.

Універсальні процесори призначені для широкого спектра завдань [75]. Вони використовуються у персональних комп’ютерах та інших компонентах комп’ютерних систем і тому мають систему команд та архітектуру, орієнтовану на типові операції керування й обробки даних, щоб за їх допомогою можна було реалізувати програми різного роду. Водночас значна частина криптографічних перетворень (особливо ті, що забезпечують інтенсивне перемішування і нелінійність) при виконанні на універсальній архітектурі можуть виконуватись дуже повільно, вимагати більшого енергоспоживання, що є критичним для багатьох пристроїв Інтернету речей [2].

Спеціалізовані процесори, навпаки, призначені для вирішення вузького класу задач та мають спрощену систему команд [75]. Їхня архітектура формується з урахуванням конкретного набору операцій, характерних для

криптографічних перетворень, що дає змогу оптимізувати як алгоритм виконання, так і апаратну реалізацію в цілому [76]. На практиці це забезпечує істотне скорочення кількості тактів на виконання операцій, зменшення накладних витрат керування та прогнозованість часових характеристик, що особливо важливо для пристроїв з обмеженою потужністю живлення й ресурсами пам’яті. Для геш-функцій реалізація у вигляді спеціалізованого процесора дозволяє винести основні криптографічні перетворення в окремий апаратний модуль із фіксованою структурою обробки даних. Такий підхід дає змогу оптимізувати виконання операцій перемішування та обробки внутрішнього стану геш-функції, а також обмежити апаратну складність реалізації відповідно до заданого класу малоресурсних пристроїв. Спеціалізовані процесори гешування є доцільними для використання в IoT-вузлах і вбудованих системах, де необхідно забезпечити контроль цілісності та автентичності даних за наявності жорстких обмежень на енергоспоживання й обчислювальні ресурси.

1.5.2 Технології апаратної реалізації засобів гешування

Для апаратної реалізації засобів гешування в малоресурсних системах застосовують два основні типи мікросхем:

- спеціалізовані інтегральні схеми (ASIC);
- програмовані логічні інтегральні схеми (ПЛІС), зокрема FPGA.

Реалізація на основі ASIC передбачає проєктування спеціалізованої інтегральної схеми, архітектура якої оптимізується під конкретну геш-функцію на етапі розробки і після виготовлення залишається незмінною. Такий підхід дозволяє досягти мінімальної апаратної складності реалізації та найкращих показників енергоефективності. Саме тому у фахових публікаціях та документах апаратні характеристики геш-функцій, зокрема показник апаратної складності, найчастіше наводять у вигляді результатів ASIC-синтезу та виражають у вентильних еквівалентах (GE) [12]. Водночас реалізація засобів гешування у вигляді ASIC пов’язана зі значними витратами на етапах проєктування, виготовлення та відлагодження, а також не допускає внесення змін до алгоритму

після виготовлення мікросхеми. Це обмежує застосування ASIC у випадках, коли необхідна гнучкість або можливість оновлення криптографічних механізмів у процесі експлуатації пристрою.

Реалізація засобів гешування на основі програмованих логічних інтегральних схем (ПЛІС), зокрема FPGA є альтернативною ASIC і широко застосовується в дослідженнях та експериментальних розробках малоресурсних криптографічних алгоритмів [24]. Архітектура ПЛІС ґрунтується на використанні конфігурованих логічних блоків, програмованих міжз'єднань та вбудованих допоміжних блоків, що дозволяє реалізовувати різні апаратні структури без фізичної зміни мікросхеми.

Основною перевагою ПЛІС є можливість багаторазового перепрограмування, що значно спрощує процес розробки, тестування та оптимізації апаратних реалізацій геш-функцій. Це дозволяє досліджувати різні архітектурні підходи, зокрема послідовні, частково паралельні та інші реалізації, а також оцінювати компроміси між апаратною складністю, швидкодією та енергоспоживанням. Разом із тим, наявність конфігураційної інфраструктури та універсальних логічних блоків у ПЛІС зумовлює додаткові апаратні й енергетичні витрати порівняно з ASIC. Унаслідок цього реалізації геш-функцій на ПЛІС, як правило, характеризуються більшою площею кристала та вищим енергоспоживанням за однакового рівня продуктивності.

Важливою складовою апаратного проєктування є формальний опис цифрової логіки та її подальша перевірка. Для цього застосовують мови опису апаратури (HDL), які забезпечують структурний і поведінковий опис вузлів, можливість ієрархічної декомпозиції, моделювання часових характеристик та подальший синтез у схемотехнічну реалізацію [77, 78]. Найпоширенішими HDL для проєктування криптографічних примітивів є VHDL (стандарт IEEE 1076) та Verilog/SystemVerilog (стандарт IEEE 1800) [79, 80], що використовуються як у FPGA-проєктах, так і під час підготовки до ASIC-синтезу.

Проєктування апаратних реалізацій геш-функцій зазвичай організовують як послідовність взаємопов'язаних етапів:

- розроблення структурних компонентів (операційних блоків і керування) та їх інтеграція у спеціалізований модуль гешування;
- функціональне моделювання і перевірка логічної структури на рівні елементарних вузлів;
- HDL-опис (VHDL/Verilog) і симуляція для підтвердження коректності на рівні сигналів і тактів;
- синтез під цільову технологію (FPGA або ASIC) з отриманням апаратних метрик (ресурси, частота, часові обмеження; для ASIC – оцінка апаратної складності GE за відповідною бібліотекою стандартних елементів).

У межах такого підходу середовища моделювання виконують різні, але взаємодоповнювальні функції. Для початкової перевірки структури та взаємодії блоків доцільно застосовувати засоби схемного моделювання, зокрема Logisim-evolution [81], які дозволяють зібрати структуру з найпростіших елементів, проаналізувати роботу окремих вузлів і їх взаємодію в складі цілого модуля. Окрім цього, Logisim-evolution підтримує використання HDL-описів у складі проєкту: зокрема можливо імпортувати/експортувати опис VHDL-компонента та виконувати його перевірку із залученням зовнішнього HDL-симулятора [81].

Подальшу перевірку коректності на рівні сигналів і тактів виконують у HDL-симуляторах, зокрема ModelSim або Questa [6, 82], які підтримують симуляцію VHDL та Verilog і широко використовуються у FPGA-проєктуванні. Таким чином, HDL-опис забезпечує можливість моделювання роботи пристрою та подальшого синтезу його схемотехнічної реалізації для ПЛІС або ASIC.

1.5.3 Технічні характеристики апаратних реалізацій геш-функцій

Оцінювання ефективності апаратної реалізації малоресурсних геш-функцій потребує використання уніфікованих показників, які дозволяють порівнювати різні алгоритми та архітектурні рішення між собою. На відміну від програмних реалізацій, для яких основними критеріями є час обчислення геш-значення та обсяг використаної пам’яті, в апаратних реалізаціях ключову роль

відіграють показники, що безпосередньо характеризують складність мікросхеми, її продуктивність та енергетичні витрати. У межах даного дисертаційного дослідження основну увагу приділено таким технічним характеристикам.

Апаратна складність є універсальною мірою розміра апаратної реалізації криптографічного примітиву, що визначається площею схеми у вентильних еквівалентах GE [12, 31]. Один вентильний еквівалент відповідає площі двовходового вентиля NAND у заданій технології. Менші значення GE відповідають компактнішим і дешевшим реалізаціям; для реалізації на пристроях з дуже низькими ресурсами (наприклад, 4-бітових мікроконтролерах) вважається прийнятним забезпечення апаратної складності до 1000 GE, а для більш загальних IoT-платформ допустимими є реалізації до 2000-3000 GE [12, 83].

Пропускна здатність [12, 83, 84] визначає кількість даних, які можуть бути оброблені за одиницю часу (у бітах або байтах за секунду) та обчислюється як:

$$T = \frac{B \cdot F}{N}, \tag{1.6}$$

де B – розмір блока (у бітах), F – тактова частота (у герцах), N – кількість тактів, що необхідна для обробки блока даних [83]. Цей показник є особливо важливим для пристроїв, що працюють з потоками даних або потребують швидкого формування геш-значень.

Енергоспоживання є критичним параметром для автономних та найбільш малоресурсних IoT-пристроїв. Обчислюється за формулою (1.7) [27, 83, 84].

$$P = \frac{B \cdot E_{per\ bit}}{Lat}, \tag{1.7}$$

де B – розмір блока (у бітах), Lat – час за який мікросхема видає вихідні дані після отримання вхідного повідомлення, $E_{per\ bit}$ – енергія необхідна для зберігання біта даних.

Для кількісної оцінки компромісу між продуктивністю та апаратною

складністю використовується коефіцієнт апаратної ефективності FoM запропонований у [14, 59, 83]. FoM є узагальненим показником апаратної продуктивності, який кількісно відображає компроміс між швидкодією обчислень і вартістю (складністю) апаратної реалізації. Формально він обчислюється як пропускна здатність, масштабована до нанобіт, поділена на добуток тактової частоти та квадрата апаратної складності в вентильних еквівалентах (GE). У CMOS-реалізаціях динамічне енергоспоживання є приблизно пропорційним площі мікросхеми, яку узагальнено характеризує апаратна складність у GE. Тому більше значення показника FoM відповідає більш ефективній за площею та енергоспоживанням реалізації.

$$\mathrm{FoM} = \frac{T \cdot 10^9}{F \cdot \mathrm{GE}^2}, \tag{1.8}$$

де T – пропускна здатність в бітах за секунду, F – тактова частота (у герцах), а GE – значення апаратної складності.

Окрім наведених технічних характеристик, при оцінюванні апаратних реалізацій малоресурсних геш-функцій необхідно враховувати також криптографічні характеристики, які визначають рівень інформаційної безпеки алгоритму. Такі характеристики, зокрема довжина геш-значення та стійкість до атак на колізії, перший і другий прообрази, вже були детально розглянуті під час аналізу геш-функцій малоресурсної криптографії.

Зазначені технічні характеристики в сукупності формують основу для порівняльного аналізу апаратних реалізацій геш-функцій і використовуються для оцінювання їх придатності до застосування в різних малоресурсних пристроях.

1.5.4 Апаратна реалізація малоресурсних пристроїв гешування

Для оцінювання можливості практичного використання апаратних реалізацій геш-функцій у системах Інтернету речей використовуються дві окремі характеристики: клас пристрою, на якому розгортається геш-функція, та

допустима апаратна складність самого криптографічного примітиву у вентильних еквівалентах GE.

Класифікація малоресурсних пристроїв подана у IETF RFC 7228 [85] і охоплює три класи: Class 0 – ультраобмежені пристрої (пасивні RFID-мітки, аналогові сенсори, смарткартки), Class 1 – активні RFID, вузли бездротових сенсорних мереж, BLE-мітки, Class 2 – вбудовані контролери з частковою підтримкою стеків HTTP та TLS. Більш потужні платформи, зокрема IoT-шлюзи й периферійні обчислювальні вузли, відносяться до неформального класу Class 2+ [14]. Для пристроїв Class 0 типовою є відсутність повноцінного процесорного ядра або наявність найпростіших керуючих автоматів, для Class 1 використовуються 8/16-розрядні мікроконтролери (наприклад, ATtiny, ATmega328P), для Class 2 – 32-розрядні мікроконтролери (Cortex-M0+, Cortex-M3, MSP430), для Class 2+ – продуктивніші вбудовані платформи (Cortex-M4, ESP32, Raspberry Pi Pico/Zero) [14].

У роботах [14, 83] сформовано чотирирівневу класифікацію малоресурсних пристроїв за допустимою апаратною складністю криптографічних примітивів у GE, що співвідноситься з класами за RFC 7228. Узагальнений вигляд класифікації наведено у таблиці 1.3.

Значення апаратної складності у таблиці 1.3 відповідають реалізації криптографічного примітиву, а не повним апаратним ресурсам пристрою чи блоку безпеки в цілому. Так, для пасивних UHF RFID-міток зі стандартом EPC Gen2 загальні апаратні ресурси становлять кілька тисяч GE, з яких на блок безпеки відведено не більше ніж 2000 GE [8], а безпосередньо на криптографічний примітив (зокрема геш-функцію) – у межах до 1000 GE [14]. Решта апаратних ресурсів блоку безпеки витрачається на керуючу логіку протоколу, інтерфейси та регістри.

Категорія I (до 1000 GE) охоплює пристрої з жорсткими апаратними обмеженнями, зокрема пасивні RFID-мітки, найпростіші ідентифікаційні носії та надобмежені сенсорні вузли. Для цього класу характерна наявність лише мінімального набору логічних елементів, відсутність повноцінного

процесорного ядра або використання найпростіших керуючих автоматів. Малоресурсні геш-функції, призначені для таких пристроїв, мають бути орієнтовані передусім на мінімізацію апаратної складності реалізації, навіть за рахунок збільшення кількості тактів обчислення та зниження швидкодії [14].

Таблиця 1.3 – Класифікація малоресурсних пристроїв за допустимою апаратною складністю криптографічних примітивів

Категорія	Апаратна складність примітиву, GE	Цільовий клас пристроїв	Характеристика пристроїв
I	до 1000	Class 0	Пристрої з надзвичайно обмеженими апаратними ресурсами, зокрема пасивні RFID-мітки (інтерфейс EPC Gen2, ISO/IEC 18000-63), найпростіші ідентифікаційні носії та сенсорні елементи
II	1000–2000	Class 1	Пристрої з обмеженим енергоспоживанням та помірними обчислювальними можливостями, зокрема вбудовані контролери, активні RFID-мітки, WSN-вузли, BLE-мітки та сенсорні вузли
III	2000–3000	Class 2	Пристрої з розширеними обчислювальними можливостями, зокрема сенсорні вузли підвищеної функціональності та комбіновані комунікаційні модулі
IV	3000–4000	Class 2+	Вбудовані контролери та периферійні обчислювальні вузли початкового рівня

Категорія II (1000–2000 GE) охоплює пристрої з обмеженим енергоспоживанням та помірними обчислювальними можливостями, зокрема вузли бездротових сенсорних мереж, активні RFID-мітки, BLE-мітки та прості вбудовані контролери. Для цього класу характерна наявність повноцінного мікроконтролерного ядра з обмеженою розрядністю та обсягом пам’яті, що робить програмну реалізацію криптографічних перетворень малоефективною з

погляду часу виконання та енергетичних витрат. У таких умовах апаратна реалізація геш-функцій використовується переважно для забезпечення цілісності та автентичності даних за умови мінімізації апаратної складності реалізації при збереженні прийнятного часу формування геш-значення.

Категорія III (2000–3000 GE) відповідає пристроям із розширеними обчислювальними можливостями, до яких належать сенсорні вузли підвищеної функціональності, комбіновані комунікаційні модулі та спеціалізовані вбудовані системи з інтегрованими радіоінтерфейсами. Для цього класу допустиме використання геш-функцій із більшою довжиною геш-значення та підвищеною пропускною здатністю, оскільки вимоги до апаратної складності дозволяють реалізувати складніші внутрішні перетворення. У таких системах апаратні реалізації геш-функцій можуть бути інтегровані в криптографічні модулі разом з іншими примітивами, забезпечуючи баланс між рівнем безпеки та продуктивністю.

Категорія IV (3000–4000 GE) охоплює вбудовані контролери та периферійні обчислювальні вузли початкового рівня, що мають суттєво більший апаратний ресурс порівняно з попередніми категоріями. Такі пристрої, як правило, використовуються у вузлах агрегації даних, керуючих модулях та периферійних обчислювальних системах, де поряд із енергетичними обмеженнями висуваються вимоги до підвищеної швидкодії криптографічних перетворень. Для цього класу можливе застосування малоресурсних геш-функцій із більш складною апаратною структурою та частковою паралелізацією внутрішніх операцій.

Переважна більшість малоресурсних геш-функцій з підродізу 1.4, орієнтованих на апаратну реалізацію, допускають різні варіанти організації обчислювального процесу. Зокрема, для одного й того самого алгоритму гешування можливе проєктування як послідовної, так і паралельної апаратної реалізації. Вибір конкретного варіанта безпосередньо впливає на апаратну складність, пропускну здатність та енергоспоживання геш-функції, а отже – на її придатність до використання в певному класі малоресурсних пристроїв.

Послідовна апаратна реалізація ґрунтується на часовому розподілі виконання раундових і допоміжних перетворень із повторним використанням одних і тих самих апаратних компонентів. У межах такої архітектури обробка окремих частин внутрішнього стану геш-функції здійснюється послідовно в різні такти із застосуванням одного функціонального блоку. Це дає змогу істотно знизити апаратну складність реалізації за рахунок скорочення кількості логічних елементів, що є визначальним чинником для пристроїв із жорсткими обмеженнями до апаратних витрат (категорії I–II згідно з класифікацією малоресурсних пристроїв). Водночас послідовні реалізації характеризуються меншою пропускною здатністю та більшим числом тактів, необхідних для формування геш-значення.

Паралельна апаратна реалізація ґрунтується на одночасному виконанні кількох операцій або обробці декількох частин внутрішнього стану в межах одного такту. Це досягається шляхом дублювання функціональних блоків або реалізації в апаратурі кількох раундових перетворень геш-функції з їх одночасним виконанням. Паралельна реалізація дозволяє істотно підвищити пропускну здатність геш-функції та скоротити час обробки блока повідомлення, однак супроводжується збільшенням апаратної складності та енергоспоживання. Паралельні реалізації є доцільними для пристроїв категорій III–IV, де допустиме використання більшого апаратного ресурсу з метою підвищення продуктивності.

Технічні характеристики апаратної складності, пропускної здатності та енергоспоживання засобів, що реалізують геш-функцій малоресурсної криптографії подано у таблицях 1.4 та 1.5. Апаратна складність, пропускна здатність та енергоспоживання подані двома оцінками, перша з яких для паралельної реалізації, а друга для послідовної. Для більшості геш-функцій, таких як ARMADILLO, SPONGENT, Quark, PHOTON, HVH розробники пропонують послідовну реалізацію з мінімальними апаратними витратами та паралельну реалізацію з підвищеною продуктивністю за рахунок паралельної обробки і збільшених апаратних ресурсів.

Таблиця 1.4 – Технічні характеристики засобів гешування на основі конструкції Меркла–Дамґарда

Геш-функція	Довжина геш-значення (біт)	Технологія (мкм)	Апаратна складність (GE)	Пропускна здатність (Кбіт/с при 100 кГц)	Енергоспожи-вання (мкВт)
ARMADILLO [59]	80	0.18	4030/2923	1090/272	77/44
	128		6025/4353	1000/250	118/65
	160		7492/5406	1000/250	158/83
	192		8999/6554	1000/250	183/102
	256		11914/8653	1000/250	251/137
DM-PRESENT	64	0.18	2530/1600	387.88/14.63	7.49/1.83
H-PRESENT	128		4256/2330	200/11.45	-
C-PRESENT [8]	192		8048/4600	59.26/1.9	-

Серед геш-функцій із мінімальними апаратними витратами до категорії I належать лише SPONGENT-88 (738 GE), PHOTON-80 (865 GE) та HVH-88 (857 GE) у послідовній реалізації, однак усі вони забезпечують довжину геш-значення лише 80–88 біт, чого може бути недостатньо для сучасних сценаріїв застосування. Геш-функції сімейств DM-PRESENT, SPN-Hash та GLUON є придатними переважно для пристроїв категорії III, оскільки їх апаратна складність знаходиться в діапазоні 2000–3000 GE при помірній пропускній здатності.

Для пристроїв категорій III–IV з пріоритетом швидкодії при довжині геш-значення 256 біт найбільш прийнятним є HVH-256 (177,78 кбіт/с, 2680 GE у паралельній реалізації), тоді як ARMADILLO, незважаючи на найвищу пропускну здатність серед розглянутих (1000 Кбіт/с для варіантів 128–256 біт), потребує від 6025 до 11914 GE, що виводить його за межі всіх чотирьох розглянутих категорій.

Таблиця 1.5 – Технічні характеристики засобів гешування на основі конструкції «губка»

Геш-функція	Довжина геш-значення (біт)	Технологія (мкм)	Апаратна складність (GE)	Пропускна здатність (Кбіт/с при 100 кГц)	Енергоспожи-вання (мкВт)
U-Quark [4]	128	0.18	2392/1379	11.76/1.47	4.07/2.44
D-Quark [4]	160		2819/1702	18.18/2.27	4.76/3.10
S-Quark [4]	224		4640/2296	50.0/3.13	8.39/4.35
SPONGENT [9]	88	0.13	1127/738	111.3/35.8	2.31/1.57
	128		1687/1060	11.43/0.34	3.58/2.20
	160		2190/1329	17.78/0.40	4.47/2.85
	224		2903/1728	13.33/0.22	5.97/3.73
	256		3281/1950	11.43/0.17	6.62/4.21
PHOTON [6]	80	0.18	1168/865	15.15/2.82	-
	128		1708/1122	10.26/1.61	-
	160		2117/1396	20/2.70	-
	224		2786/1736	15.69/1.86	-
	256		4362/2177	20.51/3.21	-
SPN-Hash [7]	128	0.18	4600/2777	55.7/36.1	-
	256		8500/4625	111.3/35.8	-
GLUON [72]	128	0.18	2071	12.12	-
	160		2800	32	-
	224		4724	58.18	-
HVH [18]	88	0.18	1129/857	44.44/2.02	-
	128		1537/1145	44.44/1.31	-
	160		1876/1385	88.89/2.02	-
	224		2420/1769	88.89/1.48	-
	256		2680/2009	177.78/2.54	-
THF [16]	128	0.13	1296	1255/250	4.52/3.14
	256		1296	1255/250	5.22/3.87
Hash-One [74]	160	0.18	2130/1006	46/2	-

THF-256 демонструє найвищу пропускну здатність (1255 Кбіт/с) за складності 1296 GE, однак відсутність підтверджених оцінок криптографічної стійкості унеможливлює його обґрунтоване практичне застосування. Енергоспоживання геш-функцій загалом зростає зі збільшенням апаратної складності реалізації. Наприклад, для SPONGENT-256 енергоспоживання у послідовній та паралельній реалізаціях становить 4,21 та 6,62 мкВт відповідно, що є одним із найкращих показників серед розглянутих.

Для категорії II доступні геш-функції з довжиною геш-значення 128–160 біт, проте їх пропускна здатність у послідовній реалізації є вкрай низькою – від 1,31 до 2,82 Кбіт/с при 100 кГц. Геш-функції з довжиною геш-значення 256 біт мають апаратну складність не менше 1950 GE (SPONGENT-256), що унеможливлює їх застосування в категорії I та суттєво обмежує вибір для категорії II.

1.6 Постановка задач дослідження

Забезпечення контролю цілісності даних та автентифікації пристроїв є важливими задачами безпеки в системах Інтернету речей. Для розв’язання цих задач в умовах обмежених апаратних ресурсів одним з основних криптографічних інструментів є малоресурсні геш-функції.

Порівняльний аналіз відомих малоресурсних геш-функцій показав, що існуючі рішення за сукупністю показників апаратної складності, довжини геш-значення та пропускної здатності не повною мірою відповідають потребам широкого класу IoT-пристроїв, що підтверджує актуальність розробки нових методів малоресурсного гешування, які забезпечують прийнятний баланс між зазначеними показниками та криптографічною стійкістю.

Таким чином, для усунення недоліків відомих методів та засобів малоресурсного гешування необхідно розв’язати такі задачі:

- розробити методи малоресурсного гешування даних;
- розробити моделі спеціалізованих процесорів для гешування даних;
- розробити протокол автентифікації RFID-міток з контролем цілісності

даних;

– провести експериментальні дослідження розроблених методів гешування та протоколу автентифікації.

1.7 Висновки до розділу 1

1. Проаналізовано задачі безпеки IoT-систем та обґрунтовано роль криптографічних геш-функцій як одного з основних інструментів забезпечення контролю цілісності даних та автентифікації пристроїв в умовах обмежених апаратних ресурсів.

2. Визначено основні криптографічні вимоги до геш-функцій, зокрема стійкість до колізій, атак на перший та другий прообрази, а також властивість лавинного ефекту. Встановлено, що визначальним параметром придатності геш-функції до практичного впровадження в малоресурсних пристроях є апаратна складність реалізації, що виражається в одиницях GE.

3. Проаналізовано базові конструкції геш-функцій, зокрема конструкцію Меркла–Дамґарда та конструкцію «губка», визначено принципи їх роботи, переваги та обмеження. Встановлено, що конструкція «губка» потребує збереження внутрішнього стану, розмір якого перевищує довжину геш-значення, що збільшує апаратну складність при забезпеченні високого рівня криптографічної стійкості, тоді як конструкція Меркла–Дамґарда має структурні вразливості, що потребують застосування відповідних модифікацій.

4. Виконано огляд та порівняльний аналіз криптографічних характеристик відомих малоресурсних геш-функцій з точки зору криптографічної стійкості. Показано, що геш-функції на основі конструкції Меркла-Дамґарда з використанням блокових шифрів як функції ущільнення забезпечують прийнятний рівень криптографічної стійкості, проте характеризуються значно вищою апаратною складністю порівняно з функціями на основі конструкції «губка».

5. Розглянуто особливості апаратної реалізації засобів гешування, зокрема застосування ASIC та ПЛІС, а також послідовні та паралельні архітектури

реалізацій. Встановлено, що послідовна реалізація дозволяє суттєво знизити апаратну складність, однак супроводжується значним зменшенням пропускної здатності геш-функції.

6. Виконано порівняльний аналіз технічних характеристик відомих малоресурсних геш-функцій з точки зору придатності для конкретних категорій IoT-пристроїв. Показано, що для пристроїв категорій I–II жодна з розглянутих геш-функцій не забезпечує задовільного поєднання апаратної складності, довжини геш-значення та пропускної здатності, що підтверджує актуальність розробки нових методів малоресурсного гешування.

Основні результати аналізу методів та засобів малоресурсного гешування даних опубліковано в роботах автора [32, 36].

РОЗДІЛ 2
РОЗРОБКА МЕТОДІВ ТА ЗАСОБІВ МАЛОРЕСУРСНОГО ГЕШУВАННЯ

2.1 Розробка методів малоресурсного гешування

Розробка криптографічних геш-функцій для малоресурсних систем вимагає поєднання двох суперечливих вимог: забезпечення необхідних криптографічних властивостей та мінімізації апаратної складності реалізації. Отримані в попередньому розділі результати порівняльного аналізу підтвердили перспективність підходів, орієнтованих на спрощення внутрішніх перетворень та використання простих комбінованих схем (зокрема на основі регістрів зсуву й елементарних логічних та арифметичних операцій) як напряму зниження апаратних витрат при збереженні потрібних властивостей гешування.

Разом із тим, у межах виконаного аналізу також було показано, що значна частина сучасних малоресурсних рішень із конкурентною апаратною складністю побудована на конструкції «губка». Незважаючи на це, у даному дослідженні як базову ітеративну основу обрано класичну конструкцію Меркла–Дамґарда, що добре узгоджується з побайтною моделлю обробки даних (блок 1 байт) і дозволяє формалізувати функцію ущільнення у вигляді простого апаратно-орієнтованого перетворення з мінімальною логікою керування. Додатковою перевагою побайтної моделі є відсутність операцій вирівнювання повідомлення до розміру блока, оскільки функція ущільнення застосовується безпосередньо до кожного байта, а довжина повідомлення визначається кількістю ітерацій.

Запропоновано два методи малоресурсного гешування, які відрізняються способом взаємодії між вхідними даними та псевдовипадковою керуючою послідовністю. Для обох підходів використано ітеративну схему, в якій вхідне повідомлення m подається у вигляді послідовності байтів:

$$m = \{m_1, m_2, ..., m_L\}, \tag{2.1}$$

де L – кількість байтів у повідомленні. Відповідно до конструкції Меркла–Дамґарда, оновлення проміжного геш-значення виконується послідовно для

кожного байта за допомогою функції ущільнення $f(\cdot)$:

$$h_i = f(m_i; h_{i-1}), \quad i = 1,\ 2,\ \ldots\ ,\ L, \tag{2.2}$$

а остаточний результат гешування визначається як $h = h_L$.

Проміжні геш-значення, обчислені функцією ущільнення $f(\cdot)$ задається послідовністю байтів:

$$h_i = \{h_{i,0}, h_{i,1}, \ldots, h_{i,k-1}\}, \quad h_{i,j} \in \{0,1,\ldots,255\}, \tag{2.3}$$

де кожен елемент $h_{i,j}$ є байтом, а параметр k обчислюється як $k = l/8$, де l – довжина геш-значення в бітах. Значення h_0 є початковим геш-значення, що ініціалізується послідовністю з k байтів:

$$h_0 = \{h_{0,0}, h_{0,1}, \ldots, h_{0,k-1}\}. \tag{2.4}$$

Початкове геш-значення формується як деяка детермінована послідовність відповідно до прийнятої моделі застосування, може бути заданою константою або згенерованим псевдовипадковим числом. Також h_0 може застосовуватись у вигляді криптографічної солі, що дозволяє забезпечити унікальність результату перетворення для однакових вхідних повідомлень. У разі застосування методу в якості ключової геш-функції значенням h_0 може бути секретним ключем або зберігати поєднання секретного ключа з криптографічною сіллю.

2.1.1 Розробка методу гешування типу «дані-генератор»

Метод гешування типу «дані–генератор» реалізує побайтну ітеративну обробку повідомлення відповідно до конструкції Меркла–Дамґарда (2.2). У подальшому цей метод будемо називати як метод HDG (від Hash «Data–Generator»).

Особливістю методу є те, що проміжні геш-значення формуються шляхом нагромаджувального додавання байтів вхідних даних у позиціях проміжних геш-значень, визначених псевдовипадковою бітовою послідовністю генератора.

Вхідне повідомлення подається згідно з формулою (2.1), а проміжні та початкове геш-значення визначаються відповідно до формул (2.3) та (2.4). Для обробки чергового байту повідомлення m_i генератор G формує біти керування:

$$g_{i,j} \in \{0,1\}, \quad j = 0,\ 1,..., k-1. \tag{2.5}$$

Керуючі біти визначають блоки проміжного геш-значення, у яких виконується додавання байту даних. Оновлення проміжних геш-значень здійснюється шляхом додавання байту m_i до відповідних складових $h_{i-1,j}$ попереднього проміжного геш-значення лише за умови $g_{i,j} = 1$. Процес реалізації функції ущільнення $f(\cdot)$ для *i*-го байту даних передбачає виконання таких дій:

1) для $j = 0,..., k-1$ виконувати:

2) згенерувати псевдовипадковий біт $g_{i,j}$;

3) виконати обчислення:

$$h_{i,j} = (h_{i-1,(k-1-j)} + m_i \cdot g_{i,j}) \mod 256. \tag{2.6}$$

Процес ущільнення здійснюється над попереднім проміжним геш-значенням h_{i-1}, а результатом виконання *k* обчислень за формулою (2.6) є оновлене значення h_i, що використовується на наступній ітерації згідно з формули (2.2).

Обчислення за формулою (2.6) реалізується за допомогою циклічного зсуву байтів проміжного геш-значення з наступним арифметичним оновленням. Нехай h_i^* – проміжне геш-значення для *i*-ї ітерації представлено формулою:

$$h_i^* = \{h_{i,0}^*, h_{i,1}^*, ..., h_{i,k-1}^*\}. \tag{2.7}$$

Тоді результатом циклічного зсуву блоків h_i^* буде значення h_i, що формується наступним чином:

$$\{h_{i,0}, h_{i,1}, h_{i,2}, ..., h_{i,k-2}, h_{i,k-1}\} = \{h^*_{i,k-1}, h^*_{i,0}, h^*_{i,1}, ..., h^*_{i,k-3}, h^*_{i,k-2}\}, \quad (2.8)$$

після чого виконується оновлення байту, що зберігається на нульовій позиції:

$$h_{i,0} = (h_{i,0} + m_i \cdot g_{i,j}) \mod 256. \quad (2.9)$$

Перетворення за формулами (2.8) та (2.9) виконується k разів протягом однієї ітерації функції ущільнення, тобто для всіх $j = 0, ..., k-1$. Оскільки на кожному кроці j байти циклічно зсуваються вправо, елемент, який на початку ітерації перебував на позиції j, після $k-1-j$ зсувів опиняється на позиції $k-1$, звідки надходить до суматора за модулем 256. Таким чином, біт керування $g_{i,j}$, згенерований на кроці j, визначає додавання для елемента, який після завершення ітерації функції у результуючому проміжному геш-значенні h_i займає позицію $k-1-j$, що відповідає формулі (2.6). Схему i-ї ітерації обчислення геш-значення за методом HDG наведено на рисунку 2.1.

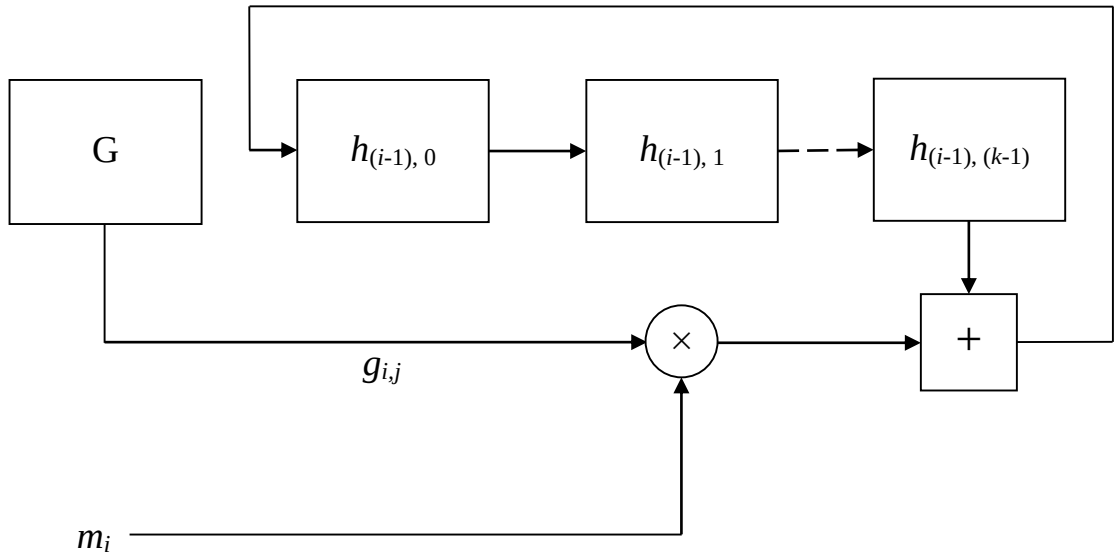

Рисунок 2.1 – Схема *i-ї* ітерації обчислення геш-значень методом HDG

Формула (2.6) демонструє ключову особливість методу, відповідно до якої байти проміжного геш-значення оновлюється шляхом додавання за модулем 256 байту вхідних даних, причому факт додавання визначається бітом керування

$g_{i,j}$, який формується генератором G. Якщо $g_{i,j}=0$, то додавання не виконується, зміни проміжного геш-значення на даному кроці зумовлюються лише циклічним переміщенням його блоків. Натомість при $g_{i,j}=1$, до відповідного блоку $h_{i-1,(k-1-j)}$ додається байт даних m_i за модулем 256, тобто здійснюється оновлення проміжного геш-значення для цього блоку. Результатом виконання останнього L-го ущільнення $f(\cdot)$ є значення h_L:

$$h_L=\{h_{L,0},h_{L,1},...,h_{L,k-1}\}, \tag{2.10}$$

що згідно з формулою (2.2) і є остаточним результатом гешування $h=h_L$ вхідного повідомлення m.

Використання додавання за модулем 256 забезпечує перемішування в межах байту. На відміну від операцій, що обмежуються лише побітовими перетвореннями, додавання формує залежність результату не тільки від однойменних розрядів доданків, а й від перенесень, які поширюють вплив молодших бітів на старші. Це підсилює взаємодію між бітами без залучення складних нелінійних компонентів. У контексті малоресурсної реалізації такий підхід є доцільним, оскільки апаратна складність восьмирозрядного суматора, як правило, істотно нижча порівняно з реалізацією табличних підстановок або багатораундових перестановок, а ефект перенесень забезпечує додатковий рівень нелінійної взаємодії бітів.

Псевдовипадкова керуюча послідовність g_i, сформована генератором G, визначає номера байтів проміжного геш-значення, у яких для поточної ітерації виконується додавання байту m_i. За рахунок цього один і той самий байт даних може впливати на різну кількість елементів проміжного геш-значення залежно від кількості одиниць у керуючій послідовності для відповідної ітерації. Зміна g_i від ітерації до ітерації приводить до того, що внесок кожного байта повідомлення накопичується в різних позиціях, а підсумкове геш-значення формується як результат послідовного нагромадження внесків усіх байтів

вхідного повідомлення в межах ітеративної схеми (2.2). Циклічне переміщення байтів проміжного геш-значення, яке супроводжує обчислення за формулою (2.6), забезпечує перерозподіл накопичених значень між позиціями в процесі ітерацій, унаслідок чого додавання, виконані для попередніх байтів повідомлення, впливають на оновлення інших байтів геш-значення на наступних кроках, що підсилює перемішування.

Важливою властивістю запропонованого методу є його апаратна орієнтованість. Реалізація функції ущільнення потребує використання k восьмирозрядних регістрів для зберігання байтів проміжних геш-значень з можливістю циклічного зсуву, одного восьмирозрядного суматора та простої логіки керування на основі бітів $g_{i,j}$. Генератор G може бути реалізовано простими регістровими структурами для формування псевдовипадкової бітової послідовності, що дає змогу отримати просту малоресурсну реалізацію з точки зору апаратних витрат.

2.1.2 Удосконалення методу гешування HDG

Запропонований метод HDG забезпечує низьку апаратну складність, однак має особливість, яка впливає на інтенсивність перемішування байтів проміжних геш-значень під час оброблення коротких повідомлень. Для криптографічних геш-функцій критичними є інтенсивне перемішування бітів геш-значення, забезпечення лавинного ефекту та відсутність небажаних статистичних закономірностей у проміжних геш-значеннях [22, 23]. У базовому перетворенні функції ущільнення (2.6) керуючий біт генератора визначає, чи виконується додавання i-го байту даних у відповідний блок проміжного геш-значення. Якщо на певному кроці для частини позицій керуючі біти дорівнюють нулю, то ці блоки не отримують арифметичного оновлення від байту повідомлення і змінюються лише за рахунок циклічного переміщення. Для довгих повідомлень цей ефект компенсується великою кількістю ітерацій, однак для коротких повідомлень кількість кроків ущільнення невелика, і нерівномірність арифметичного оновлення по позиціях проміжного геш-значення може

уповільнювати наростання дифузії на початкових ітераціях.

Для підвищення інтенсивності оновлення байтів проміжного геш-значення без суттєвого зростання апаратних витрат запропоновано удосконалення функції ущільнення методу HDG [38]. Суть удосконалення полягає у зміні ролі керуючих бітів. У базовому методі HDG вони визначають, чи виконується додавання, тоді як в удосконаленому варіанті вони визначають, яке значення додається до відповідного байту проміжного геш-значення: байт вхідного повідомлення m_i або його інверсне значення $\overline{m_i}$, що є побітовою інверсією в межах байту. Таким чином, арифметичне оновлення відбувається для всіх байтів проміжного геш-значення на кожному кроці, а псевдовипадкова послідовність генератора впливає лише на те, у якій формі (прямій чи інверсній) вхідний байт буде внесений у відповідну позицію.

Модифіковане перетворення функції ущільнення задається формулою:

$$h_{i,j} = \left(h_{i-1,(k-1-j)} + m_i \cdot g_{i,j} + \overline{m}_i \cdot (1 - g_{i,j})\right) \bmod 256, \qquad (2.11)$$

де, $j = 0,...,k-1$, а k визначає кількість байтів проміжного геш-значення відповідно до (2.3). Перетворення (2.11) безпосередньо відображає суть модифікації: якщо біт генератора $g_{i,j} = 1$, то до $h_{i-1,(k-1-j)}$ додається m_i; якщо $g_{i,j} = 0$, то додається значення $\overline{m}_i$. Отже, на кожному кроці для кожної позиції виконується одна й та сама базова операція додавання, а керування визначає лише, яке саме значення подається на вхід суматора. Циклічне переміщення байтів проміжного геш-значення зберігається таким самим, як у базовому варіанті HDG згідно з (2.8) та (2.9). Схему *i*-ї ітерації обчислення геш-значення за удосконаленим методом «дані–генератор» наведено на рисунку 2.2. В подальшому удосконалений метод типу «дані–генератор» позначається як HDGm.

Як і для геш-функції HDG, згідно з (2.2), отримане після обробки всіх *L* байтів значення h_L є остаточним геш-значенням для вхідного повідомлення.

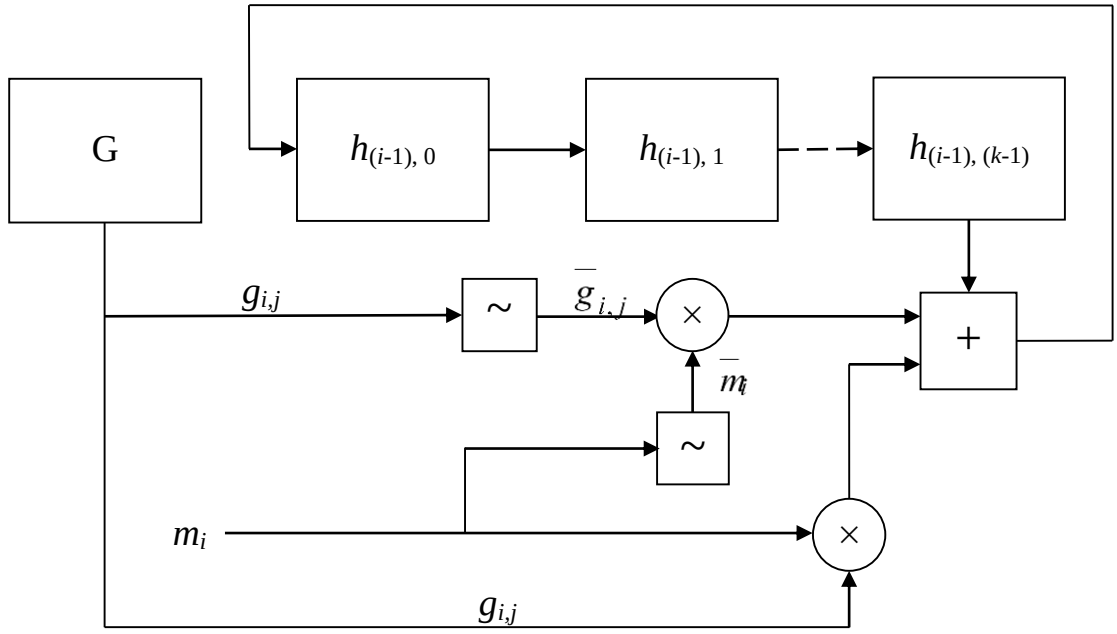

Рисунок 2.2 – Схема обчислення геш-значень методом HDGm

Запропонована модифікація усуває ситуацію, коли на окремій ітерації частина байтів проміжного геш-значення не залежить від поточного байту повідомлення, що сприяє більш швидкому наростанню дифузії на ранніх ітераціях. При цьому апаратна орієнтованість зберігається, оскільки реалізація (2.11) потребує тих самих компонентів, що й базовий метод HDG, а додаткова операція інверсії $\bar{m}_i$ реалізується лише восьмирозрядним логічним інвертором.

2.1.3 Розробка методу гешування типу «генератор-дані»

Малоресурсне побайтове гешування може бути реалізоване не лише з використанням підходу, відповідно до якого, байти вхідного повідомлення безпосередньо нагромаджуються у проміжному геш-значенні, а й за протилежним принципом. У такому випадку генератор псевдовипадкових чисел використовується як джерело даних для оновлення проміжних геш-значень, а вхідне повідомлення виконує функцію керування процесом внесення цих даних. Подібний принцип покладено в основу методу гешування типу «генератор-дані», який в подальшому будемо називати як метод HGD (від Hash «Generator–Data»).

Кожний байт повідомлення m_i розглядається як сукупність бітів:

$$m_{i,t} \in \{0,1\}, \quad t = 0,\ 1,...,\ 7, \tag{2.12}$$

На відміну від методу HDG, де функція ущільнення обробляє байт повідомлення m_i як єдине ціле протягом k кроків обчислення, метод HGD передбачає побітну обробку кожного байта вхідного повідомлення. Для кожного біта $m_{i,t}$ виконується окремий цикл оновлення всіх k байтів проміжного геш-значення. Таким чином, обробка одного байту повідомлення m_i передбачає виконання $8k$ операцій обчислення байтів проміжного геш-значення, що забезпечує багаторазове оновлення байтів проміжного геш-значення під час оброблення одного байта повідомлення.

Кожний байт $h_{i,j}$ проміжного геш-значення може оновлюватися до восьми разів під час виконання однієї ітерації функції ущільнення. У зв'язку з цим для позначення проміжних результатів i-го ущільнення введено $h_{i,j}^{(t)}$, де t – номеру обробленого біта повідомлення, а j – номер байта проміжного геш-значення, причому $j = 0,...,k-1$.

Початкові проміжні геш-значення для кожного i-го виконання функції ущільнення визначаються як:

$$h_{i,j}^{(0)} = h_{i,j}, \quad j = 0,\ 1,...,k-1, \tag{2.13}$$

Під час оброблення кожного біта $m_{i,t}$ генератор G формує k псевдовипадкових байтів:

$$g_{i,j} \in \{0,\ 1,...,\ 255\}, \quad j = 0,\ 1,...,k-1. \tag{2.14}$$

Процес реалізації функції ущільнення $f(\ )$ для i-го байту даних передбачає виконання таких дій:

1) для $t = 0,...,7$ виконувати:

2) для $j = 0,..., k-1$ виконувати:

3) згенерувати псевдовипадковий байт $g_{i,j}$;

4) виконати обчислення:

$$h_{i,j}^{(t+1)} = (h_{i-1,(k-1-j)}^{(t)} + m_{i,t} \cdot g_{i,j}) \mod 256. \quad (2.15)$$

Схему *i*-ї ітерації обчислення геш-значення за методом HGD представлено на рисунку 2.3.

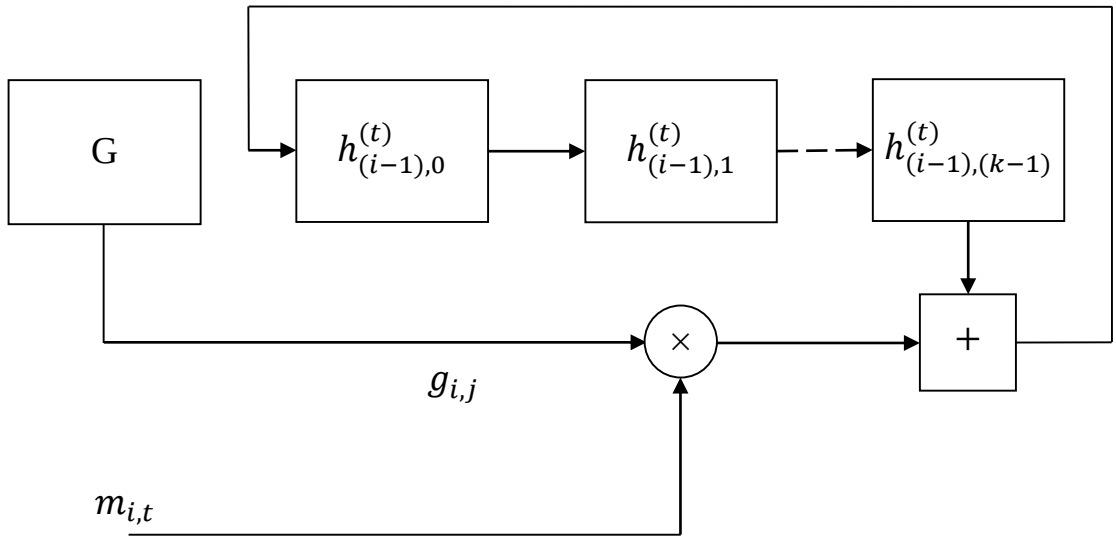

Рисунок 2.3 – Схема *i*-ї ітерації обчислення геш-значень за методом HGD

Відповідно до формул (2.14) та (2.15) біт повідомлення $m_{i,t}$ визначає, чи вносяться сформовані генератором псевдовипадкові байти $g_{i,j}$ до блоків проміжного геш-значення. Якщо $m_{i,t} = 1$, то то сформовані генератором псевдовипадкові байти $g_{i,j}$ додаються за модулем 256 до відповідних блоків проміжного геш-значення. Якщо $m_{i,t} = 0$, то додавання не виконується. Завдяки цьому реалізується принцип «дані керують внесенням псевдовипадкових значень», при якому значення одного біта повідомлення керує оновленням усіх *k* байтів проміжного геш-значення в межах відповідного циклу оброблення.

Після послідовної обробки всіх бітів $m_{i,t}$ проміжне геш-значення для *i*-го ущільнення визначається за формулою:

$$h_{i,j} = h_{i,j}^{(8)}, \quad j = 0,\ 1,..., k-1. \tag{2.16}$$

Отримане проміжне геш-значення h_i використовується під час наступного виконання функції ущільнення, де його байти подаються за формулою (2.13). Якщо $i=L$, то h_L є остаточним результатом гешування вхідного повідомлення *m*.

Апаратна реалізація запропонованого методу, може бути побудована на основі регістрів для зберігання байтів проміжного геш-значення, восьмирозрядного суматора за модулем 256 та генератора псевдовипадкових чисел на основі LFSR.

2.2 Розробка моделей спеціалізованого процесора для гешування

2.2.1 Функціональна модель спеціалізованого процесора

Апаратна реалізація розроблених методів малоресурсного гешування передбачає побудову спеціалізованого процесора (СП), який виконує обчислення геш-значення безпосередньо на апаратному рівні. Такий процесор інтегрується до складу обчислювальної системи шляхом підключення до зовнішнього контролера – центрального процесора комп'ютерної системи, мікроконтролера RFID-мітки, процесора бездротового сенсорного вузла або іншого керувального пристрою. Незалежно від типу зовнішнього контролера, взаємодія з СП описується єдиною функціональною моделлю.

Зовнішній контролер забезпечує формування вхідних даних, керування режимами роботи СП та отримання результату. Спеціалізований процесор виконує обчислювальні перетворення та зберігає проміжні результати у власних внутрішніх регістрах. Розподіл функцій між зовнішнім контролером і СП наведено у таблиці 2.1.

Взаємодія спеціалізованого процесора з зовнішнім контролером відбувається через спільну шину даних та набір сигналів керування, що задають

режими роботи СП. Схему оброблення потоку даних у СП наведено на рисунку 2.4 у вигляді функціональної моделі.

Таблиця 2.1 – Розподіл функцій між зовнішнім контролером та спеціалізованим процесором

Процес	Функції зовнішнього контролера	Функції спеціалізованого процесора
Ініціалізація	Формування початкового стану генератора та початкового геш-значення. Подання сигналу ініціалізації.	Встановлення початкового стану генератора та початкового геш-значення.
Гешування	Формування повідомлення; побайтове подання повідомлення на вхід СП.	Обчислення геш-значення.
Виведення результату	Подання сигналу завершення гешування; зчитування геш-значення з виходу СП.	Побайтове виведення кінцевого геш-значення на вихідну шину.

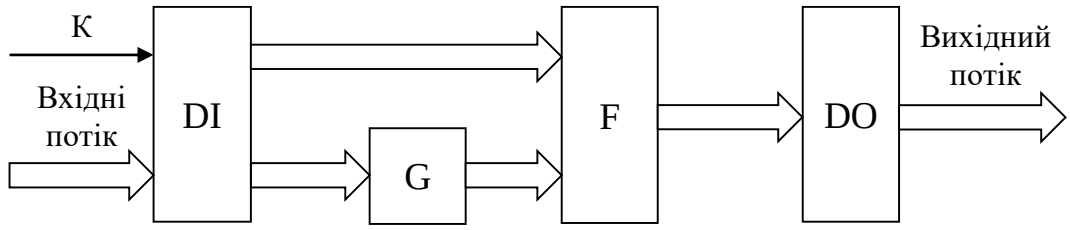

Рисунок 2.4 – Функціональна модель спеціалізованого процесора

Надходження керуючого сигналу К та вхідного потоку даних від зовнішнього контролера ініціює процес «ВХІДНИЙ ДИСПЕТЧЕР» (DI), який визначає поточний режим роботи та спрямовує потік даних до відповідних процесів. У режимі ініціалізації DI спрямовує початкові стани до процесу «ГЕНЕРАЦІЯ» (G) та до процесу «УЩІЛЬНЕННЯ» (F). У режимі гешування DI спрямовує байти повідомлення до процесу F. Процес G формує псевдовипадкову послідовність, яка надходить до процесу F. Процес F виконує обчислення функції ущільнення та зберігає проміжні геш-значення. Результат гешування

надходить на процес «ВИХІДНИЙ ДИСПЕТЧЕР» (DO), який формує вихідний потік даних у вигляді побайтового виведення кінцевого геш-значення до зовнішнього контролера.

Така організація дозволяє розвантажити зовнішній контролер від виконання трудомістких обчислень геш-значення, зосередивши на ньому лише функції подання вхідних даних та зчитування результату.

2.2.2 Структурна модель спеціалізованого процесора

Розроблені у підрозділі 2.1 методи малоресурсного гешування HDG, HDGm та HGD мають спільні принципи побудови, що полягають в ітеративному оновленні проміжного геш-значення за функцією ущільнення з використанням додавання за модулем 256 та генератора псевдовипадкової послідовності. Це дає змогу реалізувати кожен з методів на основі єдиної структури спеціалізованого процесора, причому відмінності між окремими реалізаціями визначаються організацією взаємодії його функціональних блоків і логікою керування. Структурну модель спеціалізованого процесора для малоресурсного гешування наведено на рисунку 2.5.

Структура спеціалізованого процесора містить чотири функціональні блоки, які реалізують процеси, визначені функціональною моделлю (див. рис. 2.4). Процеси вхідного диспетчера (DI) та вихідного диспетчера (DO) реалізуються блоком керування (CB) спільно з комутаційними елементами блоку регістрів (BRG): блок керування визначає поточний режим роботи та формує сигнали, що спрямовують потоки даних, а комутаційні елементи блоку регістрів забезпечують перемикання між прийманням вхідних даних та виведенням результату. Процес генерації псевдовипадкової послідовності (G) реалізується генератором псевдовипадкової послідовності (LFSR). Процес обчислення функції ущільнення (F) реалізується блоком додавання за модулем 256 (ADDB) спільно з блоком регістрів (BRG), де ADDB виконує арифметичне оновлення, а BRG забезпечує зберігання та циклічний зсув проміжних геш-значень.

Реалізація геш-функції у вигляді спеціалізованого процесора може бути

виконана в кількох варіантах, що відрізняються довжиною геш-значення в бітах $l \in \{128,\ 192,\ 256\}$. Обране значення l визначає розрядність базових регістрів та характеристики окремих вузлів процесора через величину $k = l/8$, тоді як загальна організація спеціалізованого процесора залишається незмінною.

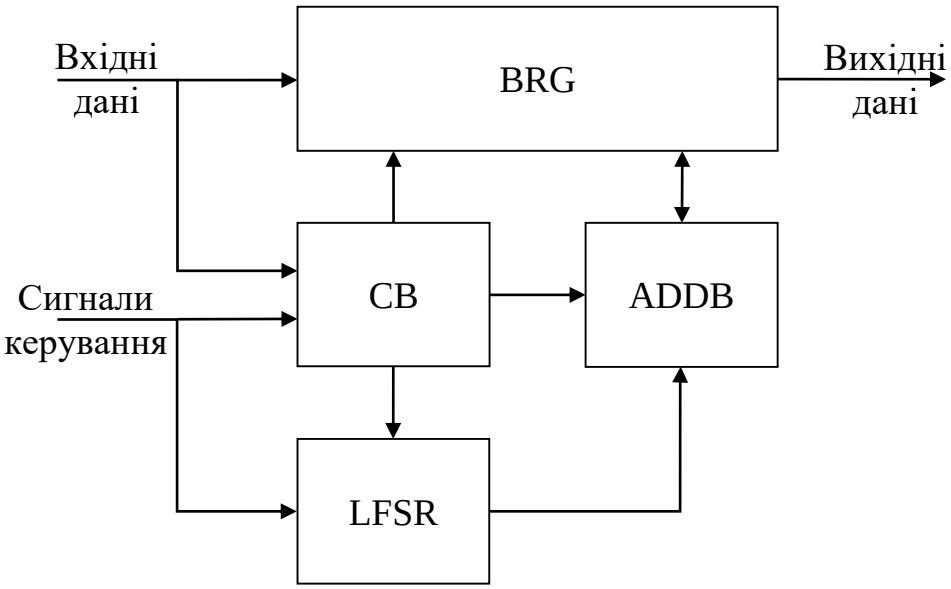

Рисунок 2.5 – Структурна модель спеціалізованого процесора для малоресурсного гешування даних

Блок BRG призначений для зберігання проміжних результатів та містить *k* восьмирозрядних регістрів, кожен із яких відповідає одному блоку проміжного геш-значення. На кожному такті функції ущільнення блок виконує циклічний зсув вмісту регістрів та приймає оновлене значення від блоку додавання. У режимі ініціалізації блок побайтно приймає початкове геш-значення через вхідну шину, а у режимі виведення результату – побайтно подає вміст регістрів на вихідну шину.

Генерацію псевдовипадкових значень у спеціалізованому процесорі виконує блок LFSR. В основі блоку LFSR регістр зсуву з лінійним зворотним зв’язком максимального періоду, побудований за конструкцією Ґалуа [86, 87], яка забезпечує компактну реалізацію зворотного зв’язку, що забезпечує зменшення апаратних витрат. Розрядність LFSR визначається вимогами до

конкретного застосування та може розглядатися як параметр реалізації спеціалізованого процесора.

Для застосувань, у яких гешуються короткі повідомлення, а самі геш-значення використовуються протягом обмеженого проміжку часу, доцільним є використання *k*-розрядного LFSR пропорційно до довжини геш-значення. Такий підхід дозволяє зменшити апаратні витрати на реалізацію генератора псевдовипадкової послідовності. Якщо ж геш-функція використовується для обробки довгих повідомлень або для формування геш-значень, що зберігаються протягом тривалого часу, доцільно використовувати 32-розрядний LFSR незалежно від довжини геш-значення. Зі збільшенням довжини вхідних даних зростає кількість звернень до генератора, а більша розрядність LFSR забезпечує більший період псевдовипадкової послідовності та зменшує вплив повторного виникнення його станів на процес формування геш-значення. Максимальна довжина періоду 2^{32}-1 досягається вибором примітивного полінома над полем *GF*(2), який визначає номери розрядів регістра, для яких формується зворотний зв'язок за допомогою елементів XOR. Розрядність числа, що генерує блок LFSR визначається реалізованим методом гешування.

Блок ADDB виконує операцію додавання за модулем 256 двох восьмирозрядних чисел з додатковим керування, що визначає, чи виконується арифметичне оновлення на поточному такті. Блок ADDB є комбінаційним вузлом і не потребує тактових сигналів. Блок керування (СВ) забезпечує узгоджену роботу усіх компонентів спеціалізованого процесора та реалізує послідовне виконання процесів ініціалізації, гешування і виведення результату. СВ приймає зовнішні сигнали керування та байт поточного повідомлення, формує внутрішній тактовий сигнал, сигнали керування ініціалізацією, дозволу запису у блок регістрів та сигнал керування блоком ADDB. Склад внутрішніх компонентів та організація блоку керування визначаються реалізованим методом гешування.

Процес роботи спеціалізованого процесора організовано таким чином. За сигналом запуску блок керування ініціює паралельну ініціалізацію генератора та

блоку регістрів. Генератор ініціалізується побітно протягом фіксованого інтервалу в 32 такти, а блок регістрів – побайтно протягом *k* тактів. Ініціалізація BRG завершується не пізніше ніж ініціалізація LFSR, оскільки $k \le 32$. Загальна тривалість фази ініціалізації складає 32 такти для будь-якої довжини геш-значення. Після завершення ініціалізації спеціалізований процесор переходить у режим гешування, в якому на кожному такті LFSR формує нове значення псевдовипадкової послідовності, BRG виконує циклічний зсув, а ADDB оновлює значення відповідного блоку проміжного геш-значення. Після надходження сигналу завершення на блок CB спеціалізований процесор припиняє обчислення і побайтно виводить кінцеве геш-значення на вихідну шину.

Запропонована структурна модель дозволяє реалізувати розроблені методи з використанням спільного набору функціональних блоків, що забезпечує узгодженість апаратних реалізацій та спрощує порівняльний аналіз їхніх технічних характеристик.

2.3 Автоматні моделі спеціалізованих процесорів для гешування

Для формалізованого опису розроблених спеціалізованих процесорів та подальшого аналізу їхніх технічних характеристик у роботі використано апарат теорії цифрових автоматів [76]. Відповідно до цього підходу, операційний пристрій задається п’ятіркою множин {D, R, S, Y, X}, де D – множина вхідних сигналів, що надходять до пристрою; R – множина вихідних сигналів, що є результатом його роботи; S – множина внутрішніх станів, що визначаються станами регістрів, тригерів та лічильників; Y – множина мікрооперацій та керувальних сигналів, що реалізують перетворення над внутрішніми станами; X– множина логічних умов, які визначають порядок виконання мікрооперацій. Послідовність виконання дій визначається окремо у вигляді алгоритму роботи керуючого автомата.

2.3.1 Автоматна модель HDG-процесора

Запропонований у підрозділі 2.1.1 метод малоресурсного гешування типу

«дані–генератор» реалізовано у вигляді спеціалізованого процесора (HDG-процесор), структура якого відповідає структурній моделі (див. рис. 2.5). Моделювання HDG-процесора виконано в середовищі Logisim-evolution у вигляді структурної цифрової моделі для довжини геш-значення l = 256 біт, що відповідає кількості регістрів $k = l/8 = 32$. Модель HDG-процесора наведено на рисунку 2.6.

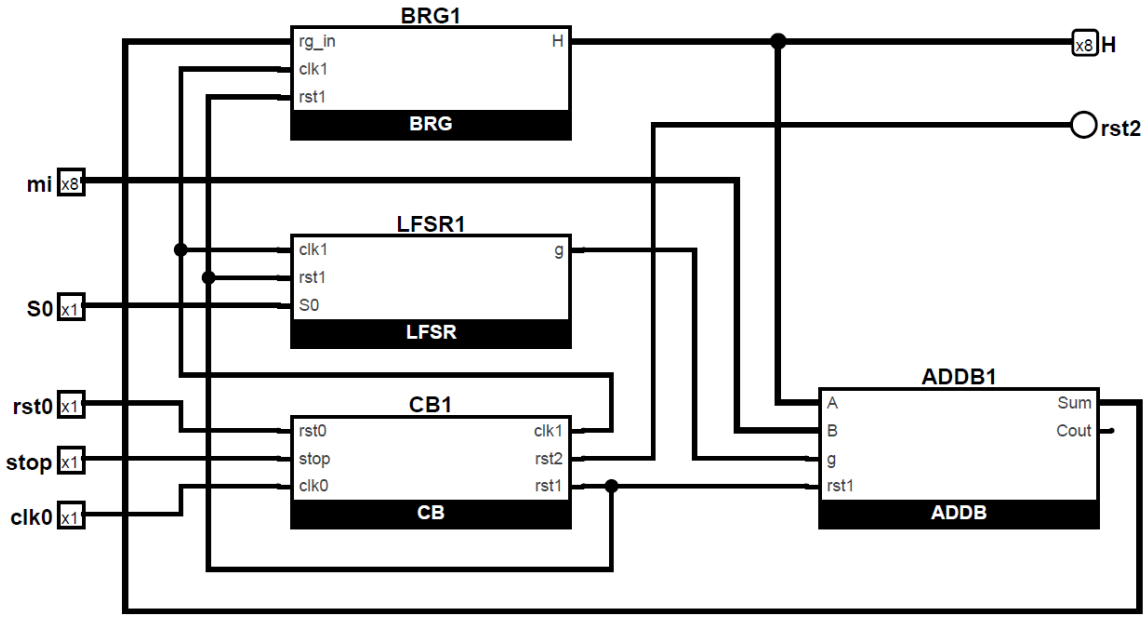

Рисунок 2.6 – Модель HDG-процесора

HDG-процесор подано у вигляді моделі цифрового пристрою:

$$\text{HDG} = \{\text{D, R, S, Y, X}\}. \qquad (2.17)$$

До множини вхідних сигналів D належать: байт даних m_i, що підлягає гешуванню; біт S_0, який є елементом послідовності, що складає початковий стан регістра LFSR. До множини вихідних сигналів R належить результат гешування H. Серед внутрішніх станів S пристрою: S_{BRG} – стан регістрів блоку BRG; S_{LFSR} – стан генератора LFSR; S_{CB} – стан елементів блоку CB. До множини Y належать мікрооперації: синхронізації (тактовий сигнал clk_0); ініціалізація початкового стану HDG-процесора (сигнал rst_0); нагромаджувальне додавання *ADD*. Таким чином модель HDG-процесора описується такою сукупністю множин:

1. Множина вхідних сигналів D = {m_i, S_0}.

2. Множина вихідних сигналів R = {H}.

3. Множина внутрішніх станів S = {S_{BRG}, S_{LFSR}, S_{CB}}.

4. Множина мікрооперацій Y = {clk_0, rst_0, ADD}.

5. Множина логічних умов X = {x_0, x_1}, де: x_0 – умова, що $rst_2 = 1$; x_1 – умова, що $stop = 1$.

Блок LFSR є 32-розрядним регістром зсуву з лінійним зворотним зв'язком у конструкції Ґалуа. Він складається з 32 D-тригерів D_{32} – D_1, де кожен тригер зберігає один біт стану. До входу тригера D_{32} підключено мультиплексор 2-в-1, а до виходу D_1 підключено логічний елемент AND, до якого через інший вхід надходить інверсне значення сигналу rst_1. При значенні сигналу $rst_1 = 1$ мультиплексор передає на вхід тригера D_{32} сигнали S_0 для ініціалізації початкового стану LFSR, а елемент AND розриває зворотний зв'язок з виходу тригера D_1. Якщо сигнал $rst_1 = 0$, мультиплексор відновлює зв'язок з виходу елементу AND до тригера D_{32}, а елемент AND відновлює зворотній зв'язок з XOR-елементами, що підключені до входів D-тригерів, номери яких визначаються примітивним поліномом:

$$P(x) = x^{32} + x^{30} + x^{26} + x^{25} + 1. \qquad (2.18)$$

Представлена конфігурація забезпечує мінімальні апаратні витрати на реалізацію регістра зсуву з лінійним зворотним зв'язком та максимальний період псевдовипадковій послідовності 2^{32}–1 [87]. Фрагмент моделі блоку LFSR наведено на рисунку 2.7, а алгоритм його роботи – на рисунку 2.8.

Модель блоку LFSR описується такою сукупністю множин:

1. Множина вхідних сигналів D={S_0}.

2. Множина вихідних сигналів R = {g}, де g – однобітовий псевдовипадковий сигнал (молодший біт стану).

3. Множина внутрішніх станів S = {S_{D1},..., S_{D32}}.

4. Множина операцій Y = {clk_1, rst_1, y_1, y_2}, де y_1 – запис до регістру зі

зворотним зв’язком; y_2 – запис без зворотного зв’язку (режим ініціалізації).

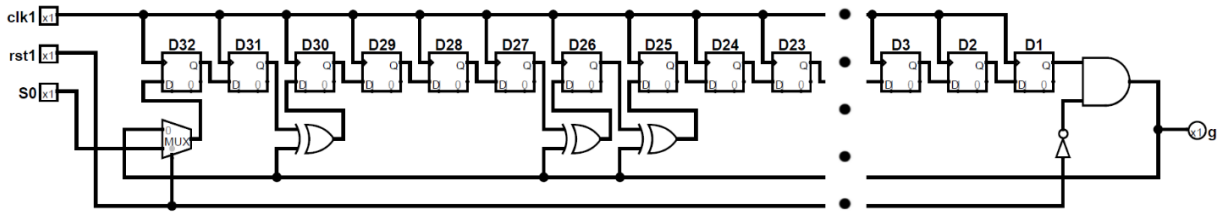

Рисунок 2.7 – Фрагмент моделі блоку LFSR для HDG-процесора

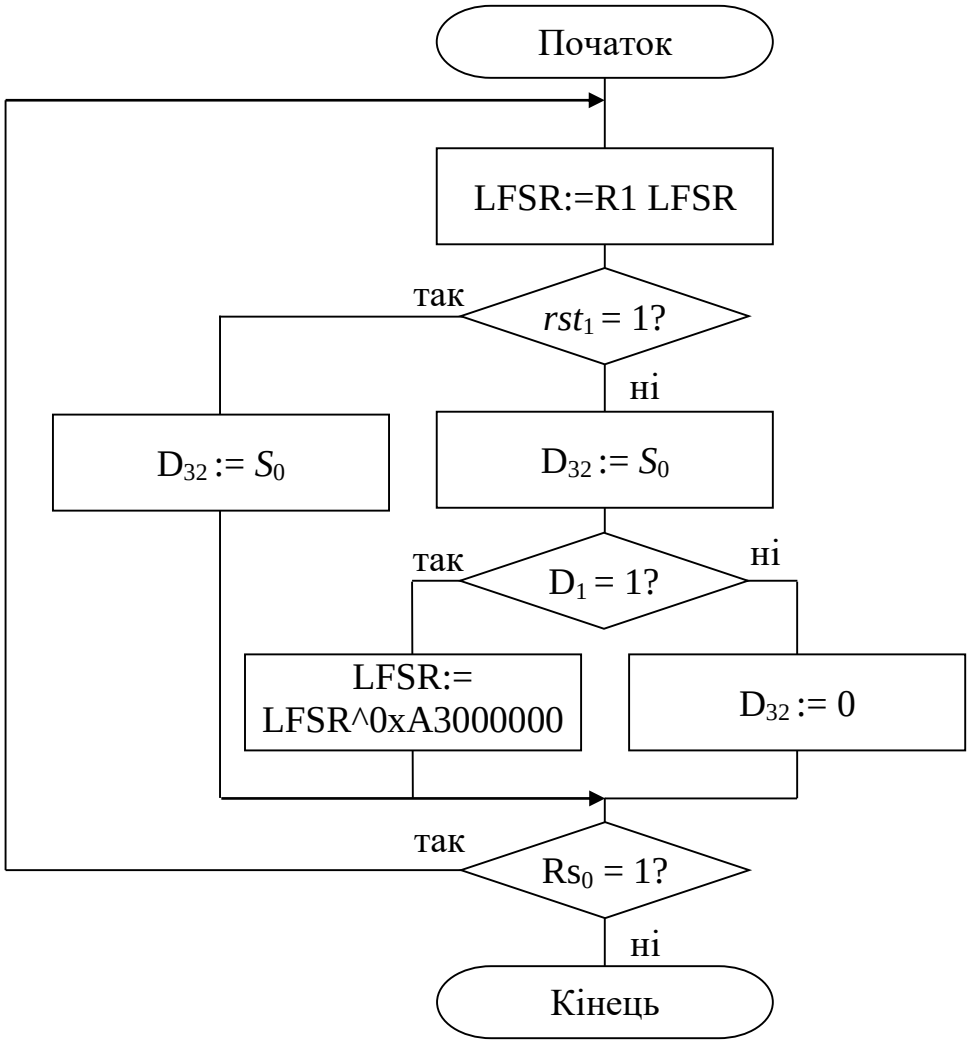

Рисунок 2.8 – Алгоритм роботи блоку LFSR

Блок BRG складається 32 послідовно з’єднаних восьмирозрядних регістрів Rg_1–Rg_{32}, де кожен регістр зберігає один байт проміжного геш-значення. До виходу регістра Rg_{32} підключено восьмирозрядний логічний елемент AND, до

іншого входу якого надходить інверсне значення сигналу rst_1. При встановленні $rst_1 = 1$ елемент AND забезпечує вихідний сигнал блоку BRG $H = 0$, що необхідно для коректної ініціалізації. Початкове геш-значення h_0 надходить до HDG-процесора через вхідний сигнал m_i та блок ADDB, де додається з нульовими байтами (забезпечено вихідним сигналом $H = 0$ блоку BRG) та записуються до блоку BRG (сигнал *rg_in*). Якщо $rst_1 = 0$, елемент AND забезпечує передачу сигналу $H = Rg_{32}$. Алгоритм роботи блоку BRG наведено на рисунку 2.9, а фрагмент моделі – на рисунку 2.10.

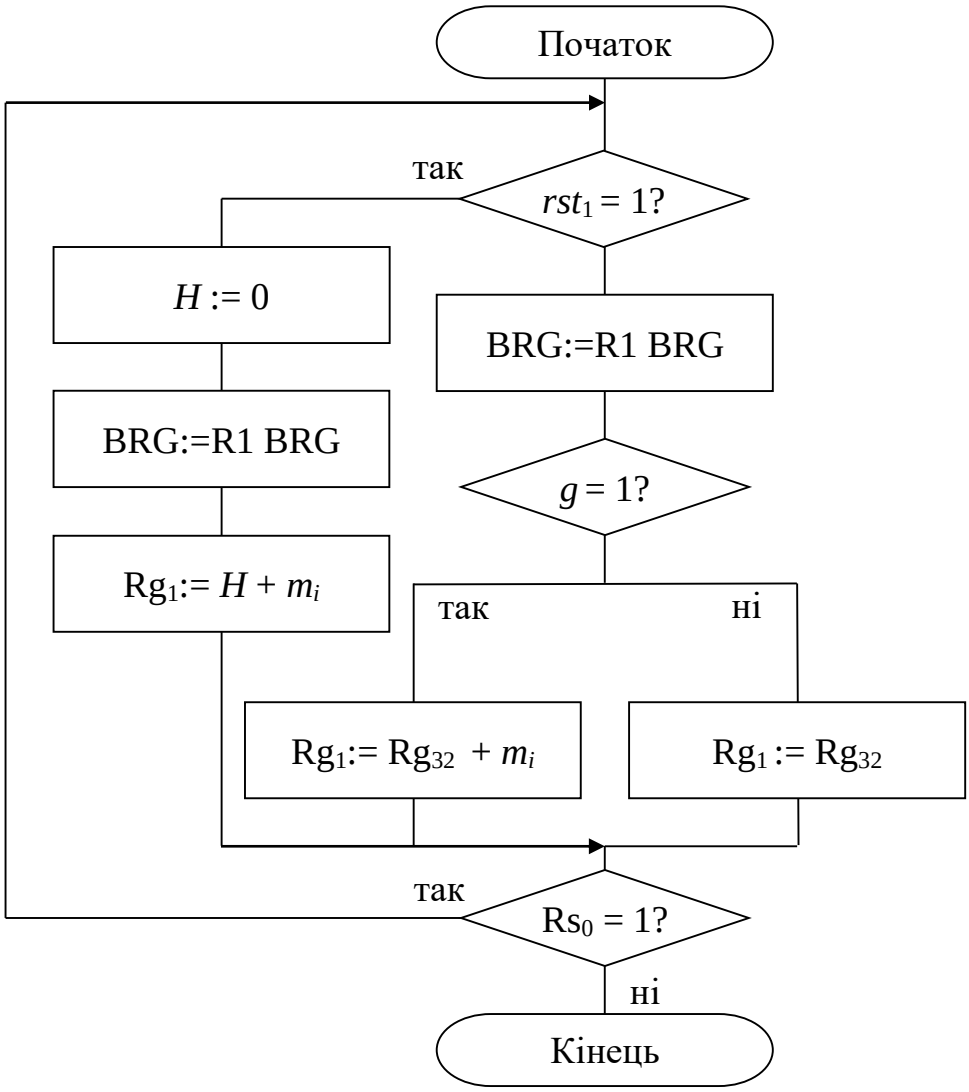

Рисунок 2.9 –Алгоритм роботи блоку BRG для HDG-процесора

Модель блоку BRG описується такою сукупністю множин:

1. Множина вхідних сигналів D = {*rg_in*}.

2. Множина вихідних сигналів R = {*H*}.

3. Множина внутрішніх станів S = {Rg_1… Rg_{32}}.

4. Множина операцій Y = {clk_1, rst_1}.

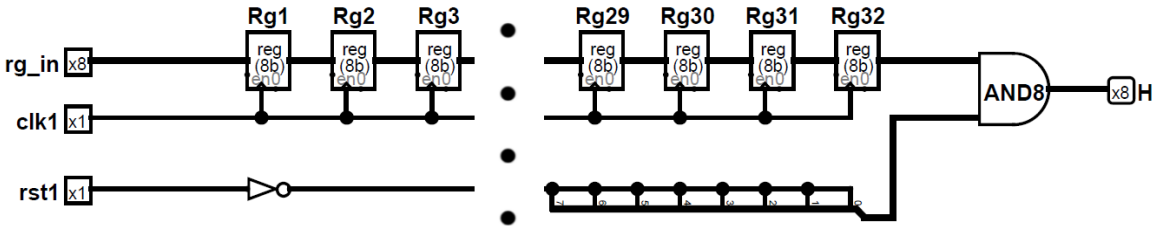

Рисунок 2.10 – Фрагмент моделі блоку BRG для HDG-процесора

Блок ADDB є восьмирозрядним комбінаційним суматором за модулем 256 з додатковим керуванням операндом *B*. Він складається з восьми однорозрядних суматорів та восьми логічних елементів AND, що здійснюють керування внесенням операнда *B*. Модель блоку ADDB наведено на рисунку 2.11.

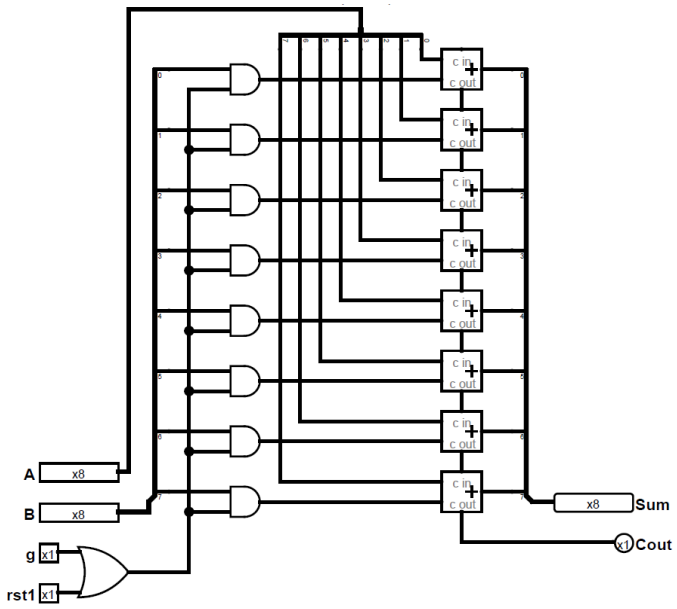

Рисунок 2.11 – Модель блоку ADDB для HDG-процесора

Операнд *A* є вихідним байтом з регістра Rg_{32} блоку BRG. Операнд *B* є поточним байтом вхідного повідомлення m_i. Сигнал керування *g* надходить від блоку LFSR. Якщо *g* = 1, то операнд *B* надходить до суматора і

$Sum = (A + B) \bmod 256$. Якщо g = 0, замість B надходить нульовий код, а $Sum = A$, що відповідає обчисленню (2.6). Додатково на блок ADDB надходить сигнал rst_1 через елемент OR, що забезпечує проходження операнда B під час ініціалізації BRG.

Блок CB забезпечує керування процесом гешування та взаємодію із зовнішнім контролером. Серед компонентів блоку CB: п'ятирозрядний лічильник Ct_1 та два RS-тригери Rs_0, Rs_1 кожен з яких складається з двох елементів NAND. Модель блоку CB наведено на рисунку 2.12, а алгоритм його роботи – на рисунку 2.13.

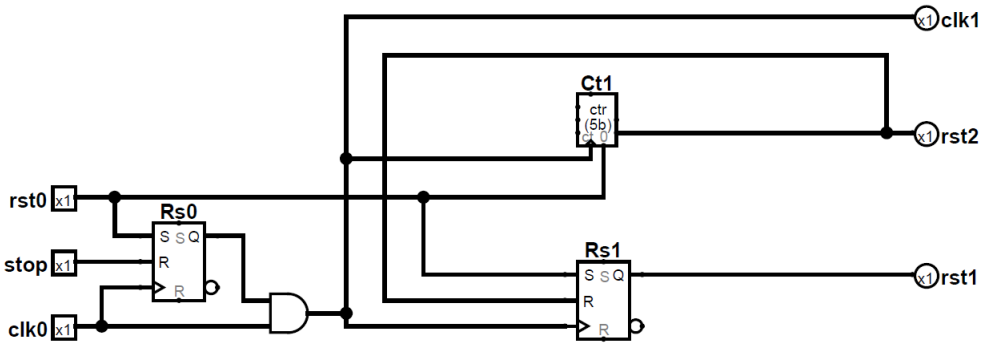

Рисунок 2.12 – Модель блоку CB для HDG-процесора

Ініціалізація блоку CB розпочинається сигналом rst_0, який встановлює тригер Rs_0 в одиничний стан, завдяки якому до блоку починають надходити тактові сигнали clk_1. Крім того в одиничний стан встановлюється тригер Rs_1, а лічильник Ct_1 встановлюється в початковий нульовий стан та розпочинає лічбу. Завдяки тригеру Rs_1 сигнал rst_1 набуває одиничного значення протягом 32 тактів, що забезпечує паралельну ініціалізацію блоків BRG та LFSR. Протягом усіх 32 тактів виконується побітний запис початкового стану S_0 до регістра LFSR. Зовнішній контролер подає байти початкового геш-значення h_0 протягом останніх k тактів фази ініціалізації, що дозволяє уніфікувати блок керування для всіх довжин геш-значення без додаткової логіки. Коли стан лічильника Ct_1 досягає значення 31, формується сигнал $rst_2 = 1$, який встановлює Rs_1 в нульовий стан, що завершує режим ініціалізації.

Після завершення ініціалізації HDG-процесор переходить у режим обчислення геш-значень, у якому на кожному такті LFSR генерує один псевдовипадковий біт, BRG виконує циклічний зсув регістрів, а ADDB оновлює відповідний байт проміжного геш-значення. Коли завершується гешування всього повідомлення, зовнішній контролер надсилає сигнал *stop*=1, який встановлює тригер Rs_0 в нульовий стан, що припиняє формування тактових сигналів clk_1 та зупиняє роботу спеціалізованого процесора.

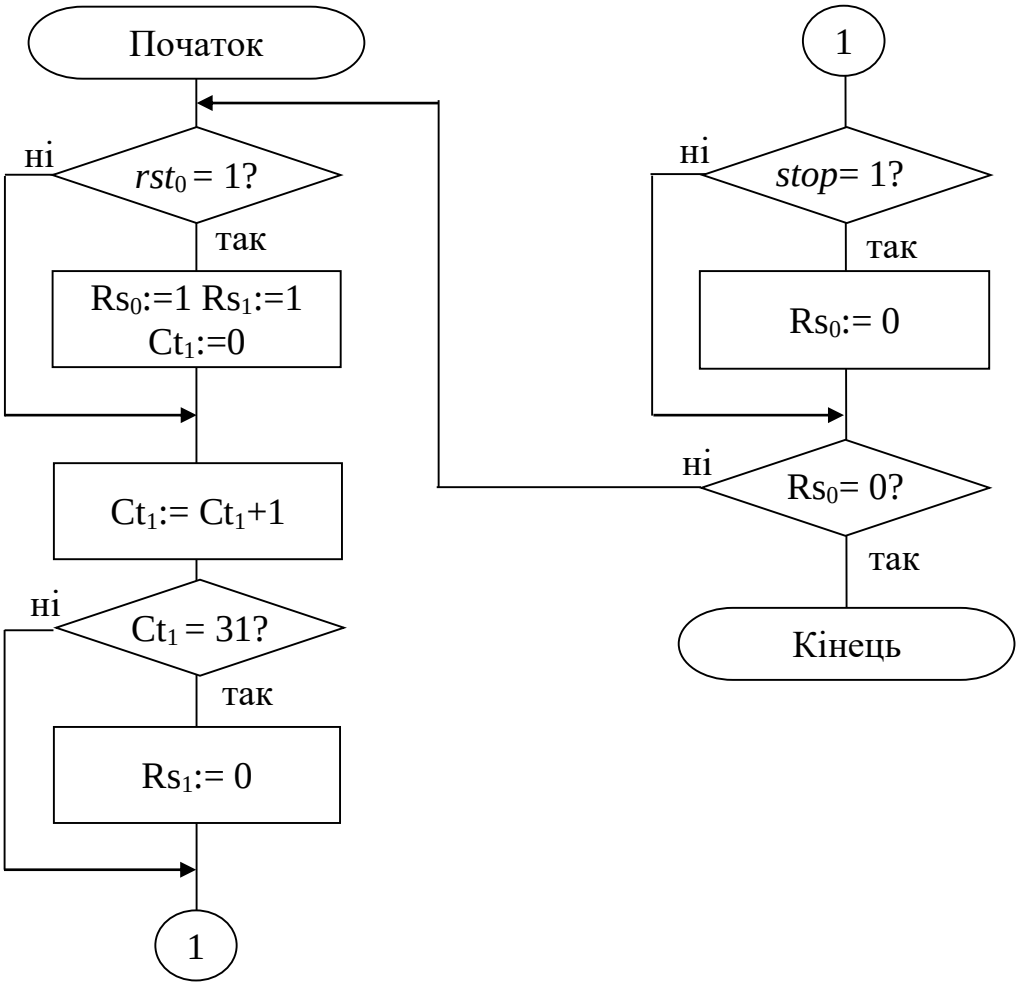

Рисунок 2.13 – Алгоритм роботи блоку СВ для HDG-процесора

Модель блоку СВ описується такою сукупністю множин:

1. Множина вхідних сигналів D = {clk_0, rst_0, *stop*}.
2. Множина вихідних сигналів R = {clk_1, rst_1, rst_2}.
3. Множина внутрішніх станів S = {S_{Ct1}, S_{Rs0}, S_{Rs1}}.

4. Множина операцій Y = {SET_{Rs0}, SET_{Rs1}, $RESET_{Rs0}$, $RESET_{Rs1}$, $RESET_{Ct1}$, *INC*}.

Побудована у середовищі Logisim-evolution структурна модель HDG-процесора для обчислення геш-значень довжиною 256 біт відтворює взаємодію блоків BRG, LFSR, ADDB і CB у режимах ініціалізації та гешування. У процесі моделювання підтверджено, що HDG-процесор коректно виконує послідовність операцій гешування та формує очікуване геш-значення для тестових вхідних даних, що свідчить про функціональну узгодженість реалізованої апаратної логіки з алгоритмічною специфікацією методу. Отримана модель використовується як основа для подальшого аналізу апаратних характеристик розробленого засобу.

2.3.2 Автоматна модель HGD-процесора

Запропонований у підрозділі 2.1.3 метод малоресурсного гешування типу «генератор–дані» реалізовано у вигляді спеціалізованого HGD-процесора, відповідно до структурної моделі (див. рис. 2.5). На відміну від HDG-процесора, де байти повідомлення нагромаджуються у проміжному геш-значенні, а генератор формує однобітову керуючу послідовність, у HGD-процесорі генератор формує псевдовипадкові байти які додаються до блоків проміжного геш-значення, а біти повідомлення визначають, чи виконується додавання на кожному кроці. Це зумовлює відмінності в організації блоків LFSR, BRG та CB порівняно з HDG-процесором.

Моделювання HGD-процесора виконано у середовищі Logisim-evolution для довжини геш-значення l = 128 біт, що відповідає кількості регістрів k = 16. Вибір довжини геш-значення l = 128 біт зумовлений тим, що для найбільш обмежених класів малоресурсних пристроїв інтернету речей, зокрема пасивних RFID-міток та простих сенсорних вузлів, не завжди є необхідність у геш-значеннях довжиною 256 біт. Водночас для моделі блоку LFSR обрано розрядність 32 біти. Такий вибір зумовлений тим, що в методі HGD генератор формує псевдовипадковий байт, який безпосередньо використовується при

обчисленні проміжних геш-значень, тому для моделі доцільно використати LFSR з більшим періодом послідовності та більш інтенсивною зміною молодших біт стану. Крім того, використання 32-розрядного LFSR дозволяє дослідити роботу HGD-процесора для випадку, коли розрядність генератора перевищує значення k, що відповідає загальній ідеї параметризації структури спеціалізованого процесора. Модель HGD-процесора наведено на рисунку 2.14.

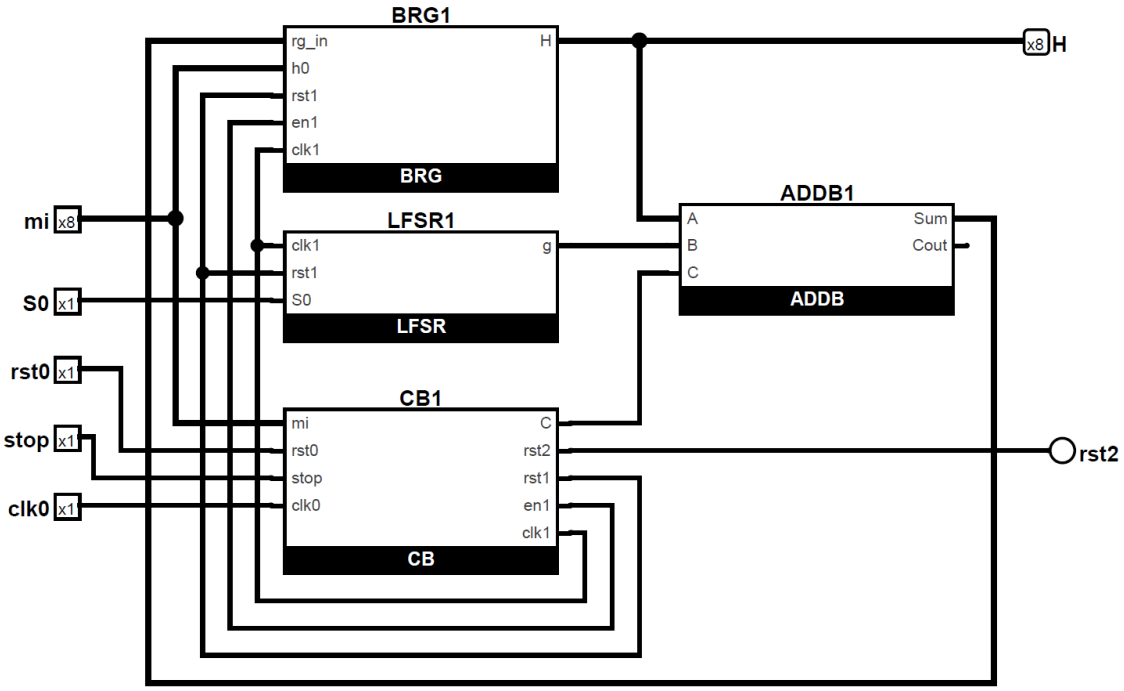

Рисунок 2.14 – Модель HGD-процесора

HGD-процесор подано у вигляді моделі операційного автомата:

$$\text{HGD} = \{\text{D, R, S, Y, X}\}. \qquad (2.19)$$

Модель HGD-процесора на верхньому рівні збігається з моделлю HDG-процесора (2.19) і описується такою самою сукупністю множин:

1. Множина вхідних сигналів D = $\{m_i, S_0\}$.

2. Множина вихідних сигналів R = $\{H\}$.

3. Множина внутрішніх станів S = $\{S_{BRG}, S_{LFSR}, S_{CB}\}$.

4. Множина мікрооперацій Y = $\{clk_0, rst_0, ADD\}$.

5. Множина логічних умов X = $\{x_0, x_1\}$, де: x_0 – умова, що $rst_2 = 1$; x_1 – умова, що $stop = 1$.

Відмінності між реалізаціями зосереджені на рівні окремих блоків і розглядаються далі.

У розглянутій моделі HGD-процесора блок LFSR реалізовано на основі 32-розрядного регістра зсуву з лінійним зворотним зв'язком у конструкції Ґалуа. Принциповою відмінністю від моделі HDG-процесора є розрядність вихідного сигналу генератора. Якщо для HDG на вихід подається один біт псевдовипадкової послідовності, то для HGD вихідним сигналом блоку LFSR є вісім молодших біт його стану, які формують псевдовипадковий байт g. На кожному такті LFSR виконує один зсув, після чого з його стану зчитується новий восьмирозрядний вихідний сигнал.

Вибір 32-розрядного LFSR для моделі HGD-процесора зумовлений необхідністю забезпечення більшого періоду псевдовипадкової послідовності порівняно з *k*-розрядним варіантом, що є важливим у випадку гешування довших повідомлень або використання геш-значень протягом тривалішого часу. Крім того, оскільки у HGD для обчислення проміжних геш-значень використовується псевдовипадковий байт, суттєвим є забезпечення інтенсивної зміни молодших біт стану генератора. Саме тому для моделі обрано 32-розрядний LFSR і примітивний поліном над полем GF(2) [77, 88, 89], у якому розряди зворотного зв'язку частково зосереджені в межах молодшого байта стану:

$$P(x) = x^{32} + x^{31} + x^{28} + x^{23} + x^{21} + x^{17} + x^{10} + x^5 + x^3 + x + 1, \qquad (2.20)$$

який забезпечує максимальний період послідовності $2^{32}-1$ [88]. Поліном визначає 9 розрядів регістра, для яких формується зворотний зв'язок за допомогою елементів XOR, з яких три (що відповідають елементам x^5, x^3 та x) розташовані в межах молодшого байту стану. Фрагмент моделі блоку LFSR наведено на рисунку 2.15.

Модель блоку LFSR описується такою сукупністю множин:

1. Множина вхідних сигналів D = $\{S_0\}$.

2. Множина вихідних сигналів $R = \{g\}$, де g – сигнал що є псевдовипадковим байтом (8 молодших біт стану LFSR).

3. Множина внутрішніх станів $S = \{S_{D1},..., S_{D32}\}$.

4. Множина операцій $Y = \{clk_1, rst_1, y_1, y_2\}$, де y_1 – запис до регістру зі зворотним зв'язком; y_2 – запис без зворотного зв'язку (режим ініціалізації).

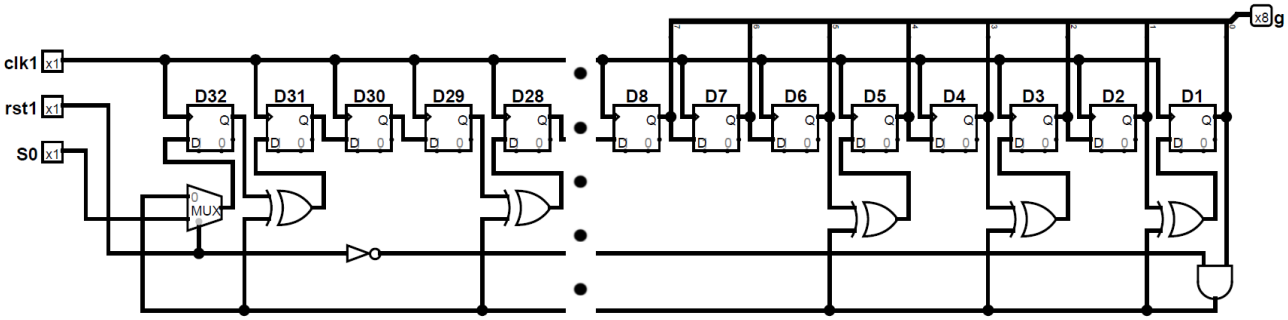

Рисунок 2.15 – Модель блоку LFSR для HGD-процесора

Блок BRG у HGD-процесорі складається з 16 послідовно з'єднаних восьмирозрядних регістрів Rg_1–Rg_{16}. На відміну від HDG-процесора, де перемикання між режимами ініціалізації та гешування забезпечується логічним елементом AND на виході останнього регістра, у HGD-процесорі до входу першого регістра Rg_1 під'єднано мультиплексор 8MUX2-в-1, що перемикає джерело вхідних даних. Керування мультиплексором здійснюється сигналом rst_1 від блоку CB. При rst_1=1 (режим ініціалізації) мультиплексор пропускає на вхід Rg_1 байт повідомлення m_i з вхідної шини, що забезпечує побайтний запис початкового геш-значення h_0. При rst_1=0 (режим гешування) мультиплексор пропускають результат обчислення Sum від блоку ADDB.

Запис у регістри здійснюється лише за умови en_1=1, що формується блоком CB. Циклічний зсув вмісту регістрів виконується на кожному такті запису. Модель блоку BRG наведено на рисунку 2.16, а алгоритм його роботи – на рисунку 2.17.

Модель блоку BRG описується такою сукупністю множин:

1. Множина вхідних сигналів D = {*rg_in*, h_0}.

2. Множина вихідних сигналів R = {*H*}.

3. Множина внутрішніх станів S = {Rg_1… Rg_{16}}.

4. Множина операцій Y = {clk_1, rst_1, en_1}.

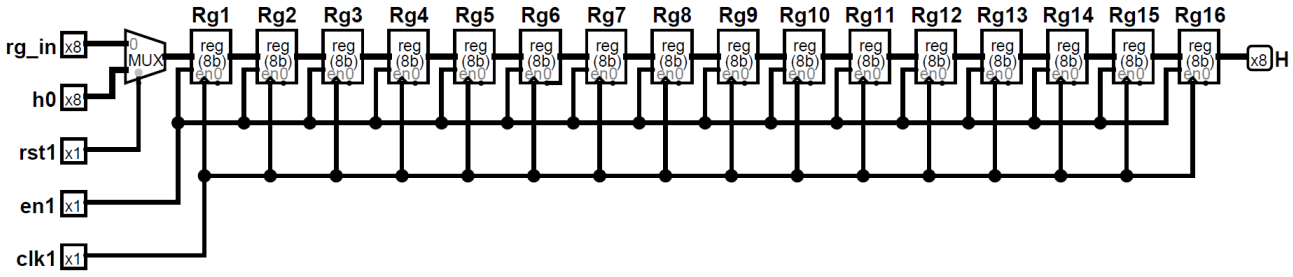

Рисунок 2.16 – Модель блоку BRG для HGD-процесора

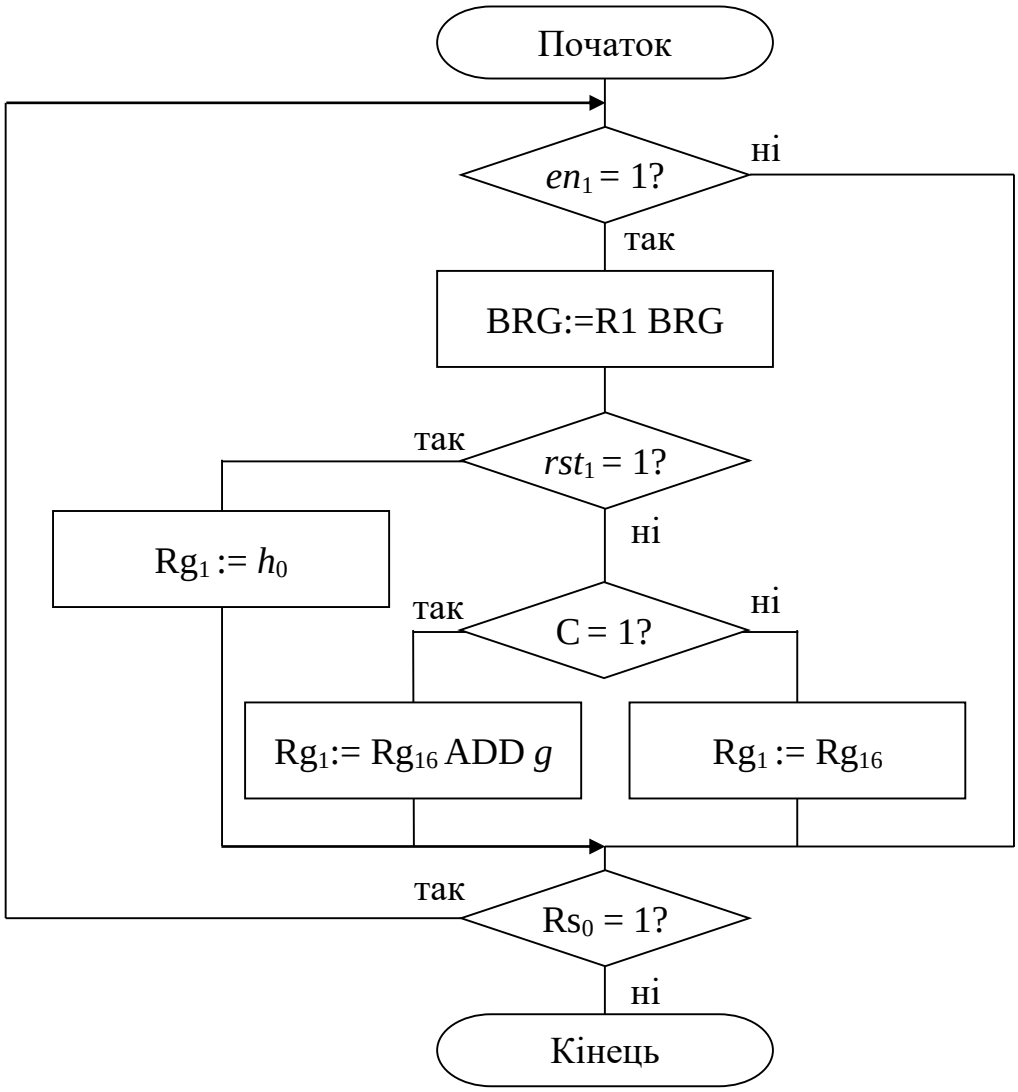

Рисунок 2.17 – Алгоритм роботи блоку BRG для HGD-процесора

Блок ADDB у HGD-процесорі, як і в HDG, є восьмирозрядним комбінаційним суматором за модулем 256 з керуванням операндом *B*. На вхід A надходить байт проміжного геш-значення з виходу регістра Rg_{16} блоку BRG, на вхід *B* – псевдовипадковий байт g від LFSR, а на вхід *C* – керуючий сигнал від

блоку СВ. Значення сигналу C є бітом вхідного байту $m_{i,t}$, що вибирається мультиплексором у блоці СВ. При C =1 операнд B надходить до суматора і $Sum = (A + B) \bmod 256$. При C =0 замість B надходить нульовий код, і $Sum = A$, що відповідає обчисленню (2.15). Модель блоку ADDB наведено на рисунку 2.18.

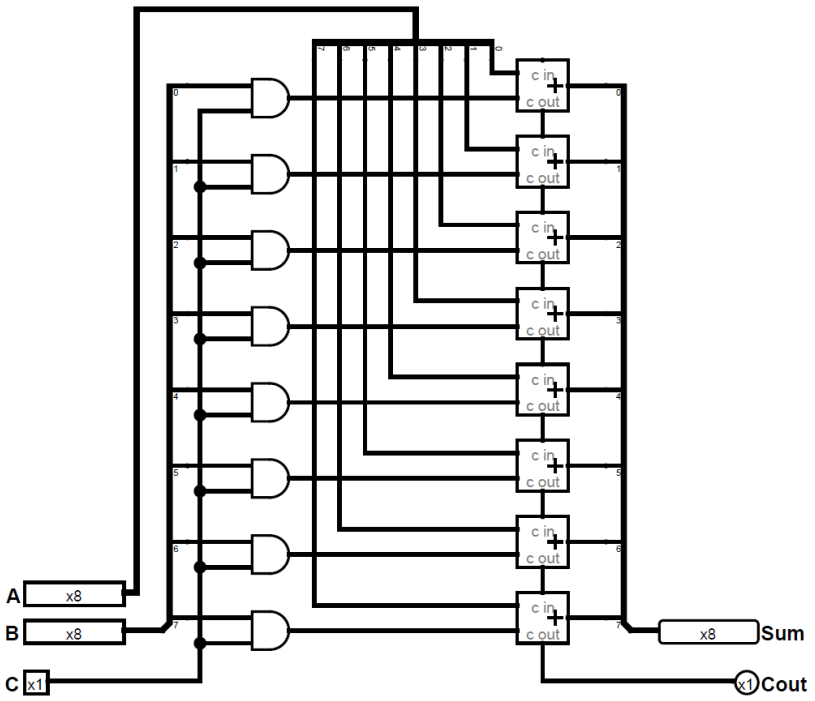

Рисунок 2.18 – Модель блоку ADDB для HGD-процесора

Блок керування СВ у HGD-процесорі є складнішим порівняно з відповідним блоком HDG-процесора, що зумовлено необхідністю побітної обробки кожного байту повідомлення. До складу блоку СВ входять два RS-тригери Rs_0 та Rs_1, п’ятирозрядний лічильник Ct_1 , трирозрядний лічильник Ct_2, мультиплексор 8-в-1 та комбінаційна логіка формування керуючих сигналів. Модель блоку СВ наведено на рисунку 2.19, а алгоритм його роботи представлено на рисунку 2.20.

Тригер Rs_0 зберігає загальний стан HGD-процесора (увімкнено/вимкнено). Один такт сигналу rst_0=1 встановлює Rs_0 в одиничний стан, після чого починають формуватись внутрішні тактові сигнали clk_1. Сигнал *stop* встановлює Rs_0 в нульовий стан, що припиняє роботу HGD-процесора. Тригер Rs_1 визначає

режим ініціалізації: при Rs_1=1 HGD-процесор перебуває на етапі ініціалізації, а при Rs_1 =0 – у режимі гешування.

Ініціалізація HGD-процесора виконується паралельно для блоків LFSR та BRG протягом 32 тактів. Лічильник Ct_1 рахує від 0 до 31, при цьому протягом перших k тактів (для l = 128 маємо k = 16) формується сигнал en_1=1, що забезпечує побайтний запис початкового геш-значення h_0 до регістрів BRG. Для значень Ct_1 від k до 31 сигнал en_1=0, а BRG перебуває у стані очікування, тоді як LFSR продовжує побітний запис початкового стану з S_0. Коли Ct_1 досягає значення 31, формується сигнал rst_2=1, який скидає Rs_1 в нульовий стан і завершує фазу ініціалізації.

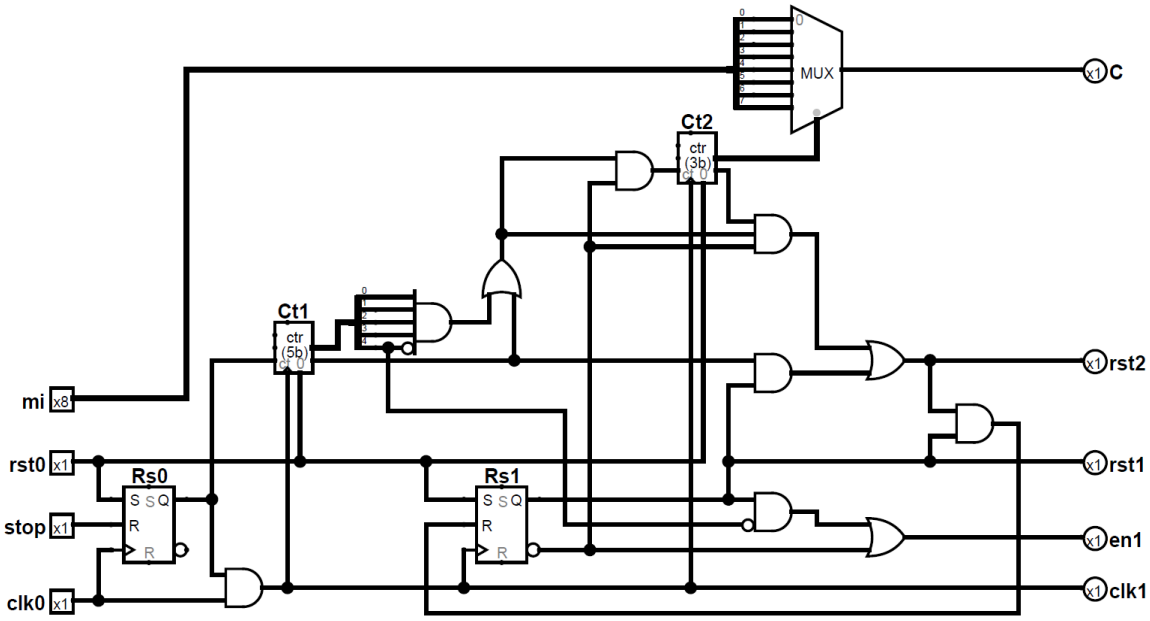

Рисунок 2.19 – Модель блоку CB для HGD-процесора

Після завершення ініціалізації процесор переходить в режим гешування. Обробка одного байту повідомлення m_i відбувається протягом 8k тактів. Лічильник Ct_1 рахує від 0 до k-1, визначаючи позицію оновлюваного байту у блоці BRG. Лічильник Ct_2 рахує від 0 до 7, визначаючи номер біту $m_{i,t}$ поточного байту повідомлення. Мультиплексор 8-в-1 вибирає біт $m_{i,t}$ за номером t =Ct_2 і подає його як керувальний сигнал C на вхід блоку ADDB.

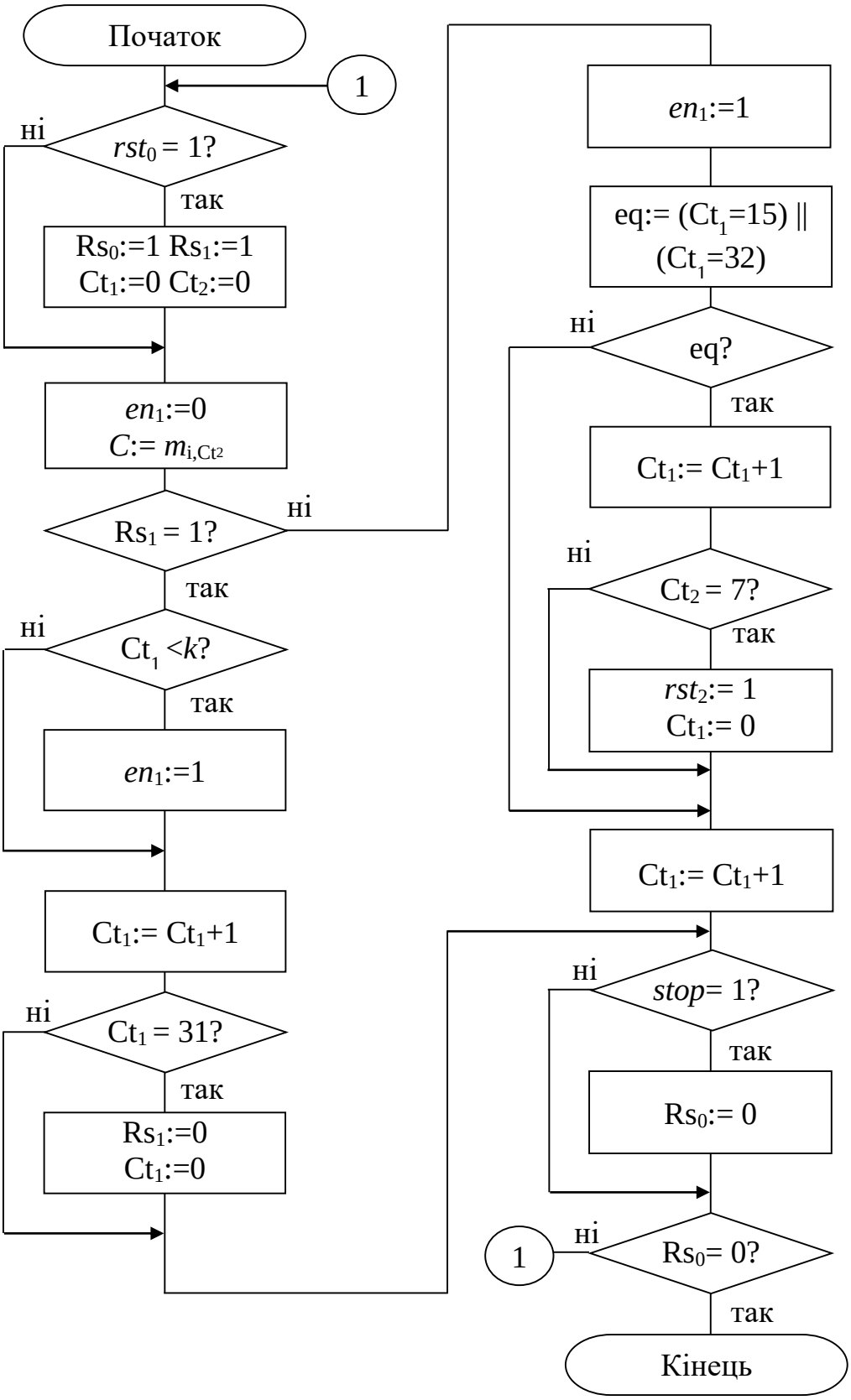

Рисунок 2.20 – Алгоритм роботи блоку СВ для HGD-процесора

При досягненні Ct_1=k-1 (значення 15) та Ct_1=2k-1 (значення 31)

завершується цикл оновлення k байтів проміжного геш-значення для t-го біту $m_{i,t}$ і лічильник Ct_2 інкрементується, що перемикає мультиплексор на наступний біт повідомлення. Таким чином за один повний цикл лічильника Ct_1 (32 такти) обробляються два біти повідомлення, а повна обробка одного байту m_i виконується за чотири таких цикли. Коли Ct_2 досягає значення 7 одночасно з виконанням умови $Ct_1=k-1$ або $Ct_1=2k-1$, формується сигнал $rst_2=1$, що сигналізує зовнішньому контролеру про завершення обробки поточного байту повідомлення. Сигнал $rst_2=1$ скидає лічильник Ct_1 в нульовий стан.

Для отримання обчисленого геш-значення зовнішній контролер подає короткий сигнал $rst_0=1$ після чого HGD-процесор переходить в режим ініціалізації, під час якого побайтно виводить обчислене геш-значення з регістрів BRG. Зупинка роботи процесора здійснюється встановленням сигналу *stop*.

Модель блоку СВ описується такою сукупністю множин:

1. Множина вхідних сигналів $\mathrm{D}=\{clk_0, rst_0, m_i, stop\}$.

2. Множина вихідних сигналів $\mathrm{R}=\{clk_1, rst_1, en_1, rst_2, C\}$.

3. Множина внутрішніх станів $\mathrm{S}=\{S_{Ct1}, S_{Ct2}, S_{\mathrm{Rs0}}, S_{\mathrm{Rs1}}\}$.

4. Множина мікрооперацій $\mathrm{Y} = \{SET_{\mathrm{Rs0}},\ SET_{\mathrm{Rs1}},\ RESET_{\mathrm{Rs0}},\ RESET_{\mathrm{Rs1}},\ RESET_{\mathrm{Ct1}},\ RESET_{\mathrm{Ct2}},\ INC\}$.

5. Множина логічних умов $\mathrm{X} = \{x_0, x_1, x_2, x_3\}$, де: x_0 – умова, що $Ct_1 = 31$ (завершення ініціалізації); x_1 – умова, що $Ct_1 \bmod 16 = 15$ (завершення циклу зсуву всіх регістрів BRG); x_2 – умова, що $Ct_2 = 7$ та $x_1 = 1$ (завершення обробки байту); x_3 – умова, що $stop = 1$.

У процесі моделювання підтверджено, що HGD-процесор коректно виконує послідовність операцій гешування та формує очікуване геш-значення для тестових вхідних даних, що збігається з результатами програмної реалізації методу HGD. HDG-процесор вимагає значно простішого керування у порівнянні з HGD, що демонструють відповідні моделі блоків СВ. Водночас HGD-процесор забезпечує інтенсивніше перемішування байтів проміжного геш-значення за рахунок внесення псевдовипадкових байтів від генератора, що підтверджується

результатами статистичного тестування, наведеними у розділі 4.

2.4 Порівняльні оцінки апаратної складності та продуктивності розроблених спеціалізованих процесорів гешування

Порівняння апаратної ефективності розроблених спеціалізованих процесорів HDG та HGD виконано за трьома показниками: апаратною складністю, продуктивністю (пропускною здатністю) та інтегральним коефіцієнтом апаратної ефективності FoM.

Оцінювання апаратної складності проводиться в умовних одиницях GE для технології 0.18 мкм із використанням бібліотеки стандартних комірок UMCL18G212T3 [60, 90]. Розрахунок здійснюється шляхом декомпозиції кожного функціонального блоку спеціалізованого процесора на базові логічні елементи бібліотеки та підсумовування їхніх еквівалентів GE. Для узгодженості оцінок використано такі значення еквівалентів UMCL18G212T3:

$$\begin{gathered} D = 5.33 \text{ GE};\ NOT = 0.67 \text{ GE};\ NAND = 1 \text{ GE};\ AND = 1.33 \text{ GE}; \\ OR = 1.33 \text{ GE};\ MUX = 2.33 \text{ GE};\ XOR = 2.67 \text{ GE}. \end{gathered} \tag{2.21}$$

Апаратна складність спеціалізованих процесорів залежить від довжини геш-значення l через параметр $k = l/8$, який визначає кількість восьмирозрядних регістрів у блоці BRG. Загальна складність спеціалізованих процесорів обчислюється як сума складностей функціональних блоків:

$$S(k, w) = S_{\text{BRG}}(k) + S_{\text{LFSR}}(w) + S_{\text{ADDB}} + S_{\text{CB}}(k, w). \tag{2.22}$$

Складність блоку LFSR визначається розрядністю регістра w та кількістю розрядів зворотного зв'язку n_{xor} примітивного полінома. Як зазначено у підрозділі 2.2, розрядність LFSR може становити $w = 32$ біти (фіксована для всіх довжин геш-значення) або $w = k$ біт (зменшена відповідно до довжини геш-значення). Складність блоку LFSR обчислюється як:

$$S_{\text{LFSR}}(w) = w \cdot D + n_{xor} \cdot XOR + MUX + NOT + AND. \tag{2.23}$$

Складність блоку ADDB є однаковою для HDG та HGD і складає:

$$S_{\text{ADDB}} = 16XOR + 24AND + 8OR. \quad (2.24)$$

Для модифікації HDGm вісім елементів AND замінюються на 8 мультиплексорів 2-в-1 та 8 елементів NOT:

$$S_{\text{ADDB}}^{HDGm} = 16XOR + 16AND + 8OR + 8MUX + 8NOT. \quad (2.25)$$

Блок BRG містить k восьмирозрядних регістрів та комутаційні елементи, склад яких визначається реалізованим методом. Для HDG-процесора використовується 8 елементів NAND для керування виходом, для HGD-процесора – 8 мультиплексорів 2-в-1 для перемикання входу:

$$S_{\text{BRG}}^{\text{HDG}}(k) = k \cdot 8D + 8NAND;\ S_{\text{BRG}}^{\text{HGD}}(k) = 8k \cdot D + 8MUX. \quad (2.26)$$

Складність блоку CB залежить від реалізованого методу та конфігурації лічильника Ct_1. Для HDG-процесора блок CB містить основний лічильник Ct_1 та два RS-тригери. Для HGD-процесора CB додатково містить 3-розрядний лічильник та мультиплексор 8-в-1, що реалізується каскадом із 7 мультиплексорів 2-в-1. Розрядність Ct_1 визначається максимальною кількістю тактів, необхідною для ініціалізації LFSR і становить 5 біт при $w = 32$ або $[\log_2(k)\,]$, біт при $w = k$. Для визначення моменту завершення цикл (Ct_1=k−1) використовується або вихід переповнення лічильника, або додатковий компаратор складністю $S_{cmp} \approx 4AND + NOT$. Компаратор не потрібен при $w = k$, де $k = 16$ або $k = 32$. В решті випадків компаратор перемикає забезпечує керування та встановлення лічильника Ct_1 в нульовий стан. Тоді складність блоку CB для HDG та HGD-процесорів обчислюється за формулами:

$$S_{\text{CB}}^{\text{HDG}}(k,w) = n_{ct} \cdot D + 4NAND + OR + S_{cmp}(k,w) + S_{\text{logic}}^{HDG}, \quad (2.27)$$

$$S_{\text{CB}}^{\text{HGD}}(k,w) = (n_{ct} + 3) \cdot D + 7MUX + 4NAND + S_{cmp}(k,w) + S_{\text{logic}}^{HGD},$$

де $S_{cmp}(k,w)=0$, якщо $w=k$ при $k\neq 24$; $n_{ct}=[\log_2(\max(k,w))]$ – розрядність Ct_1; S_{logic} – складність решти логіки формування сигналів керування.

Продуктивність спеціалізованих процесорів для гешування оцінюється пропускною здатністю, тобто кількістю біт вхідного повідомлення, що обробляються за одиницю часу. Нехай вхідне повідомлення має довжину L байтів, тобто $8L$ біт, а внутрішній тактовий сигнал спеціалізованого процесора clk_1 надходить з частотою F_{clk_1}. Для коректної оцінки враховується, що загальний час обчислення включає ініціалізацію спеціалізованого процесора, оброблення L байтів повідомлення та зчитування результату гешування через вихідний інтерфейс. Пропускна здатність визначається відповідно до (1.6):

$$T(L)=\frac{8L\cdot F_{clk_1}}{N(L)}, \tag{2.28}$$

де $N(L)$ – загальна кількість тактів clk_1, необхідна для ініціалізації, оброблення L байтів та виведення результату.

У HDG-процесорі обробка одного байту повідомлення потребує k тактів, а ініціалізація та виведення результату – по 32 та k тактів відповідно. Тоді:

$$N_{\text{HDG}}(L)=32+k\cdot L+k, \quad . \tag{2.29}$$

У HGD-процесорі обробка одного байту повідомлення потребує $8k$ тактів, оскільки для кожного з 8 бітів повідомлення виконується k оновлень регістрів блоку BRG. Тоді загальна кількість тактів для обробки L байт HGD-процесором:

$$N_{\text{HGD}}(L)=32+8k\cdot L+k=64+8kL+k. \tag{2.30}$$

Для достатньо великих L, витрати часу на ініціалізацію та зчитування стають малими. Тоді узагальнені оцінки пропускної здатності процесорів:

$$T_{\text{HGD}}\approx\frac{F_{clk_1}}{k}=\frac{8F_{clk_1}}{l};\ T_{\text{HDG}}\approx\frac{8F_{clk_1}}{k}=\frac{64F_{clk_1}}{l}, \tag{2.31}$$

тобто зі збільшенням довжини геш-значення, пропускна здатність зменшується обернено пропорційно *l*.

Пропускна здатність модифікації HDGm залишається на рівні HDG, оскільки модифікація не впливає на кількість тактів обробки. Числові оцінки апаратної складності та пропускної здатності для різних значень довжини геш-значення за умови використання типової для малоресурсних вузлів тактової частоти $F_{clk_1} = 100$ кГц наведено у таблиці 2.2.

Таблиця 2.2 – Апаратна складність та пропускна здатність СП

Довжина геш-значення *l*, біт	*k*	*w* (LFSR)	Апаратна складність (GE)			Пропускна здатність (Кбіт/с при 100 кГц)	
			HDG	HDGm	HGD	HDG/ HDGm	HGD
128	16	16	915	929	972	50	6.25
128	16	32	998	1012	1069	50	6.25
192	24	24	1300	1314	1360	33.33	4.17
192	24	32	1355	1368	1410	33.33	4.17
256	32	32	1680	1693	1752	25	3.13

Результати таблиці 2.2 підтверджують, що апаратна складність розроблених спеціалізованих процесорів не перевищує 1752 GE для довжини геш-значення 256 біт, з 32-розрядним LFSR, що свідчить про можливість застосування подібних апаратних засобів для малоресурсних IoT-пристроїв, що належать до категорії II та вище. Використання LFSR зменшеної розрядності $w = k$ дозволяє додатково скоротити апаратну складність, що є доцільним для застосувань з короткими повідомленнями. HDG-процесор забезпечує вищу пропускну здатність порівняно з HGD за нижчої апаратної складності.

2.5 Порівняльні оцінки спеціалізованих апаратних засобів гешування

Для узагальненого оцінювання апаратної ефективності реалізацій геш-функцій, у роботі використовується коефіцієнт FoM, визначений у розділі 1

(формула 1.8). Значення FoM для розроблених процесорів наведено в таблиці 2.3.

Таблиця 2.3 – Значення коефіцієнта FoM для розроблених методів

Довжина геш-значення *l*, біт	*k*	*w* (LFSR)	FoM		
			HDG	HDGm	HGD
128	16	16	597	580	66
128	16	32	502	489	55
192	24	24	197	193	22
192	24	32	182	178	21
256	32	32	89	87	10

Порівняння виконано з відомими малоресурсними геш-функціями ARMADILLO, PHOTON, SPN-Hash, Quark, GLUON, PRESENT та HVH, основні технічні характеристики яких розглянуто у розділі 1, а значення коефіцієнтів FoM отримано за результатами роботи [14]. Порівняльні оцінки апаратної складності, пропускної здатності та коефіцієнтів FoM розроблених і відомих малоресурсних геш-функцій наведено на рисунках 2.21–2.23 відповідно.

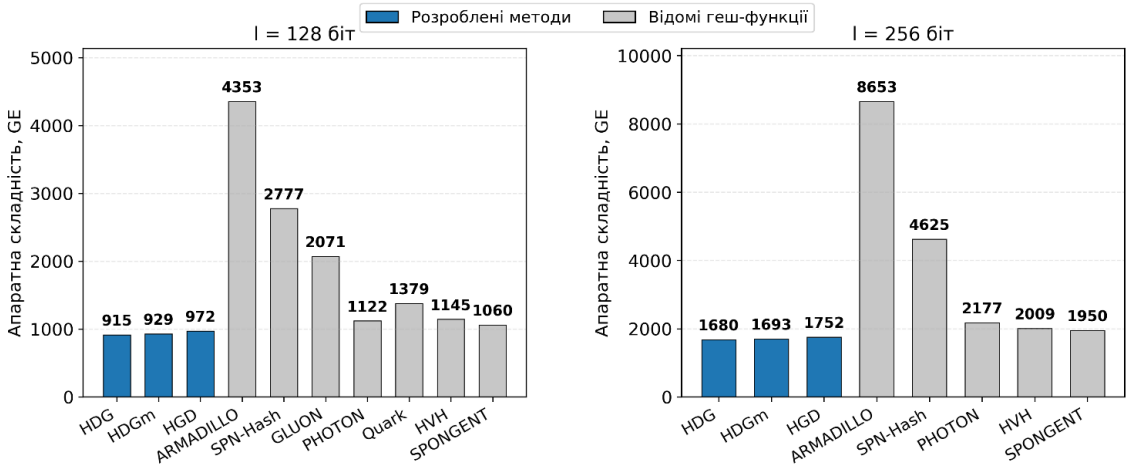

Рисунок 2.21 – Порівняльні оцінки апаратної складності малоресурсних геш-функцій

Аналіз результатів порівняння апаратної складності та пропускної здатності підтверджує переваги розроблених спеціалізованих процесорів, що

забезпечують найменшу апаратну складність серед усіх розглянутих малоресурсних геш-функцій. При цьому зберігається відносно висока пропускна здатність гешування, особливо для HDG та HDGm.

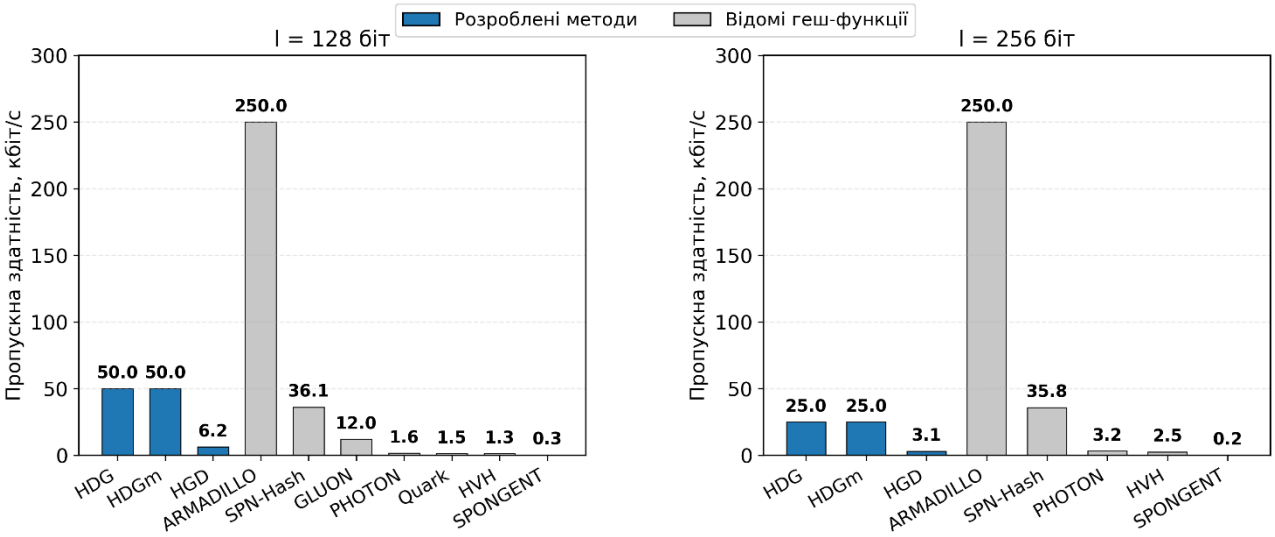

Рисунок 2.22 – Порівняльні оцінки пропускної здатності малоресурсних геш-функцій

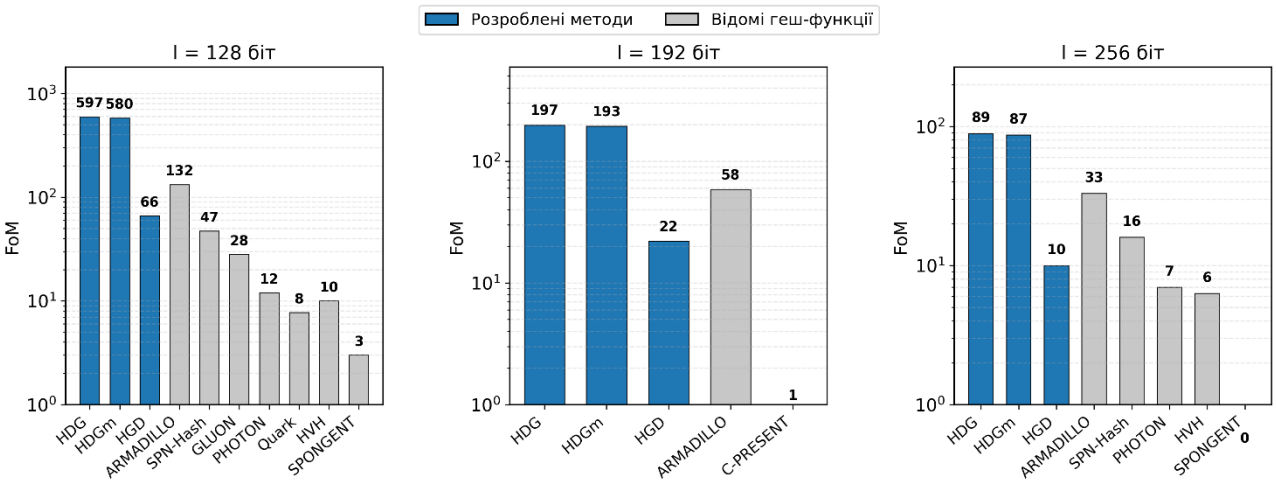

Рисунок 2.23 – Порівняльні оцінки коефіцієнтів FoM малоресурсних геш-функцій

Порівняльні оцінки FoM підтверджують, що розроблені HDG та HDGm забезпечують найвищі значення FoM серед розглянутих для усіх довжин геш-значень 128–256 біт. Для l = 128 біт значення FoM для HDG та HDGm перевищують ARMADILLO приблизно у 4.5 рази та SPN-Hash приблизно в 12

разів. Для $l = 192$ та $l = 256$ біт реалізація HDG також демонструє перевагу за FoM порівняно з відомими малоресурсними геш-функціями, що свідчить про більш вигідне поєднання пропускної здатності та апаратних витрат. Геш-функція HGD характеризується помірними значеннями FoM, що можна пояснити її меншою пропускною здатністю через багатотактову обробку одного байта повідомлення. Незважаючи на це HGD-процесор демонструє кращі показники апаратної ефективності ніж геш-функції, побудовані за конструкцією типу «губка», як Quark, HVH, PHOTON та GLUON.

2.6 Висновки до розділу 2

1. Запропоновано методи малоресурсного гешування на основі ітеративної конструкції Меркла–Дамґарда, у яких функція ущільнення передбачає використання простих операцій додавання за модулем 256, циклічного зсуву складових проміжного геш-значення, а також генератора псевдовипадкової послідовності, що дозволяє зменшити апаратні витрати на реалізацію геш-функцій.

2. Метод HDG передбачає внесення байтів повідомлення до проміжного геш-значення у позиціях, які визначаються псевдовипадковою послідовністю, що забезпечує відносно високу швидкості гешування при мінімальних витратах. Метод має особливість, яка впливає на інтенсивність перемішування байтів проміжних геш-значень під час оброблення коротких повідомлень. Тому запропоновано удосконалений метод HDGm, що використовує розширене правило вибору даних, які додаються до проміжних геш-значень для усунення цього недоліку.

3. Метод HGD відрізняється тим, що проміжне геш-значення оновлюється шляхом внесення псевдовипадкових байтів, які формуються генератором і керуються байтами даних вхідного повідомлення. Це вимагає більшої кількості кроків для обчислення геш-значення, але забезпечує інтенсивніше перемішування байтів геш-значення порівняно з HDG та HDGm.

4. З метою реалізації кожного з розроблених методів запропоновано

структурну модель спеціалізованого процесора, яка містить нагромаджувальний суматор, генератор псевдовипадкової послідовності, регістри для збереження проміжних геш-значень та блок керування. Розроблені у середовищі Logisim-evolution автоматні моделі спеціалізованих процесорів HDG та HGD підтвердили коректність організації їх роботи та відповідність запропонованим функціям ущільнення.

5. Отримані оцінки апаратної складності показали, що розроблені спеціалізовані процесори забезпечують обчислення геш-значень довжиною 128–256 біт при складності 915–1680 GE для HDG та 972–1752 GE для HGD. Отже, навіть для 256-бітного геш-значення апаратні витрати залишаються меншими за 2000 GE, що підтверджує придатність розроблених засобів для малоресурсних пристроїв Інтернету речей.

6. Порівняння з відомими малоресурсними засобами для гешування показало зменшення апаратних витрат на 14–24 % для HDG та на 8–20 % для HGD. За інтегральним показником FoM реалізації HDG та HDGm забезпечують найвищу апаратну ефективність серед усіх розглянутих малоресурсних геш-функцій.

Основні наукові результати, представлені в цьому розділі, опубліковано у працях автора [32, 33, 34, 38, 39, 40].

РОЗДІЛ 3
ОРГАНІЗАЦІЯ КОНТРОЛЮ ЦІЛІСНОСТІ ДАНИХ ТА АВТЕНТИФІКАЦІЇ ПРИСТРОЇВ ІОТ

3.1 Контроль цілісності даних у малоресурсних IoT-системах

3.1.1 Контроль цілісності повідомлень телеметрії сенсорних вузлів

Однією з найбільш поширених задач у IoT-системах є збирання та передавання даних телеметрії від сенсорних вузлів до шлюзу або серверної частини системи [83]. Типовим прикладом є IoT-система моніторингу параметрів мікроклімату, в якій сенсорний вузол періодично вимірює температуру та відносну вологість повітря і передає отримані дані бездротовим каналом зв’язку до шлюзу системи [49]. Повідомлення з даними телеметрії від такого вузла, як правило, має невеликий обсяг в декілька десятків байт і містить виміряні значення фізичних величин разом із службовою інформацією (мітка часу, ідентифікатор вузла). Конфіденційність таких даних здебільшого не є критичною, однак їх модифікація під час передавання може призвести до хибних рішень системи керування, некоректного обліку або порушення роботи автоматизованих процесів. Контроль цілісності окремого такого повідомлення з даними телеметрії можна забезпечити шляхом обчислення геш-значення, що формується із використанням спільного секретного ключа K, відомого лише сенсорному вузлу та шлюзу:

$$h_i = H(K||M_i), \tag{3.1}$$

де M_i – і-те повідомлення, K – секретний ключ, $H(\cdot)$ – геш-функція.

Сенсорний вузол формує повідомлення у вигляді пари (M_i, h_i) та надсилає його до шлюзу. Шлюз обчислює очікуване геш-значення $h_i^* = H(K||M_i^*)$ для отриманого повідомлення M_i^* і порівнює його з отриманим h_i. Якщо $h_i^* = h_i$, повідомлення вважається цілісним, інакше фіксується факт його модифікації. Схему контролю цілісності даних телеметрії наведено на рисунку 3.1.

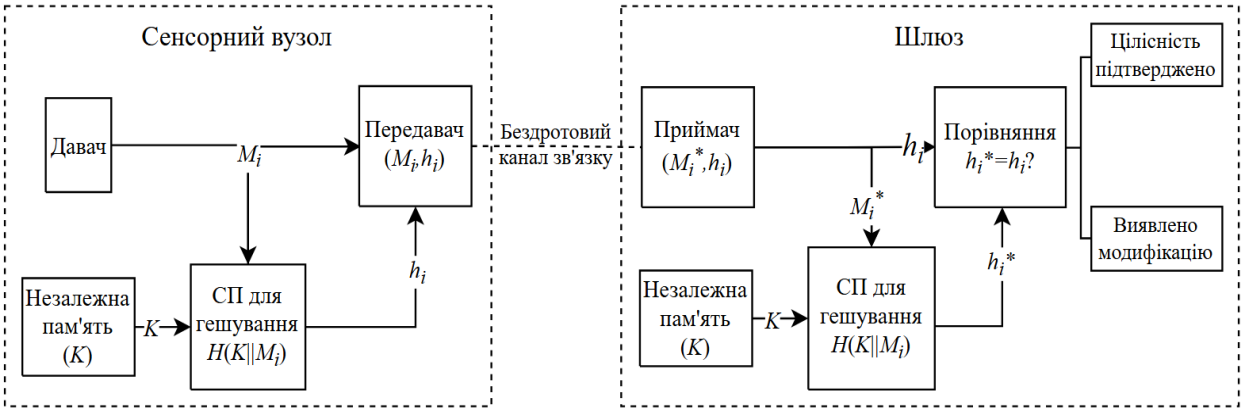

Рисунок 3.1 – Схема контролю цілісності телеметричного повідомлення

Для застосувань, у яких важливим є контроль цілісність не лише окремого повідомлення, а й послідовності вимірювань, може бути використаний механізм зв’язування геш-значень у ланцюжок:

$$h_i = H(K||M_i||h_{i-1}), \tag{3.2}$$

де h_{i-1} – геш-значення попереднього повідомлення з даними від сенсорного вузла. Такий підхід пов’язує кожне повідомлення з попереднім, що забезпечує виявлення не лише модифікації окремого повідомлення, а також його вилучення або додавання нового. Однак механізм представлений формулою (3.2) є чутливим до втрати пакетів і тому доцільний переважно для надійних каналів зв’язку або для формування журналів вимірювань, що зберігаються локально.

Для розглянутої схеми характерними є невеликий обсяг повідомлень, відсутність довгострокового зберігання геш-значень та необхідність мінімізації апаратних витрат сенсорного вузла. Згідно з результатами, наведеними у підрозділі 2.5, для таких умов доцільним є використання геш-функції HDG з довжиною геш-значення l = 128 біт. Апаратна складність HDG-128 становить 915-998 GE при пропускній здатності 50 кбіт/с (за тактової частоти 100 кГц), що є найкращим показником серед розглянутих малоресурсних геш-функцій за коефіцієнтом апаратної ефективності FoM (див. рис. 2.23).

Для коротких повідомлень сенсорних вузлів у системах із жорсткими обмеженнями на апаратні ресурси додатково може бути розглянуто використання LFSR зменшеної розрядності ($k = l/8 = 16$ біт замість фіксованих 32 біт), як описано у підрозділі 2.2. Період такого генератора є достатнім для повідомлень обсягом до кількох кілобайт, а зменшення розрядності LFSR забезпечує додаткове скорочення апаратної складності спеціалізованого процесора.

3.1.2 Контроль цілісності програмного забезпечення при завантаженні IoT-пристрою

Іншим практичним сценарієм контролю цілісності є перевірка програмного забезпечення (прошивки) IoT-пристрою при його завантаженні. Для цього використовується механізм безпечного завантаження Secure Boot, що передбачає перевірку цілісності образу прошивки перед його виконанням. Перевірка цілісності дозволяє виявити несанкціоновані зміни в прошивці, внесені внаслідок атаки або збою під час зберігання чи оновлення. Класичні схеми безпечного завантаження [91], зокрема ті, що описані у рекомендаціях NIST SP 800-193 [92], передбачають використання цифрових підписів для автентифікації образу прошивки, що забезпечує як контроль цілісності, так і підтвердження походження коду. Однак перевірка цифрових підписів потребує значних обчислювальних ресурсів та реалізації асиметричної криптографії, що є неприйнятним для найбільш обмежених класів IoT-пристроїв. У таких випадках контроль цілісності прошивки може бути реалізований за спрощеною схемою з використанням малоресурсної криптографічної геш-функції.

Загальна схема такої перевірки цілісності полягає в тому, що на етапі виробництва або оновлення обчислюється еталонне геш-значення образу прошивки, яке зберігається у захищеній ділянці незалежної пам’яті пристрою. При кожному запуску системи завантажувач обчислює геш-значення поточного образу прошивки та порівнює його з еталонним. У разі збігу значень пристрій продовжує завантаження, у разі розбіжності – блокує виконання або активує

процедуру відновлення. Подібну схему контролю цілісності прошивки при завантаженні наведено на рисунку 3.2.

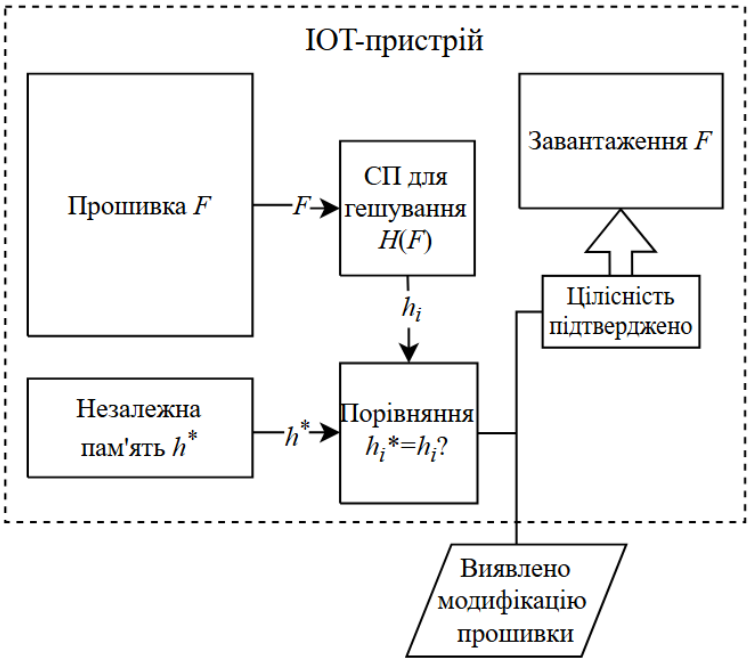

Рисунок 3.2 – Схема контролю цілісності програмного забезпечення при завантаженні IoT-пристрою

Аналогічний підхід застосовується при оновленні прошивки бездротовим каналом (Over-The-Air, OTA), коли пристрій верифікує отриманий образ перед його встановленням. У цьому випадку геш-значення може обчислюватися потоково під час зчитування байтів нового образу без потреби повного збереження прошивки в оперативній пам’яті до завершення перевірки, що особливо важливо для мікроконтролерів із малим обсягом RAM.

У роботі [93] запропоновано один із варіантів подібного підходу для IoT-пристроїв, у якому верифікація прошивки здійснюється із застосуванням малоресурсних геш-функцій SPONGENT та PHOTON без використання цифрових підписів. Особливістю методу є поділ образу прошивки на сегменти з окремою оцінкою критичності кожного сегмента та адаптивний вибір глибини перевірки залежно від режиму роботи пристрою.

Використання геш-функцій HDG та HGD для контролю цілісності

прошивки при завантаженні системи дозволить скоротити час верифікації порівняно з використанням SPONGENT та PHOTON завдяки вищій пропускній здатності спеціалізованих процесорів для гешування, що підтверджується результатами порівняльного аналізу, наведеними у підрозділі 2.4. При цьому, на відміну від сценарію контролю цілісності повідомлень сенсорних вузлів, верифікація прошивки характеризується більшим обсягом оброблюваних даних, оскільки розмір образу прошивки може складати декілька кілобайтів, та довгостроковим зберіганням еталонного геш-значення між завантаженнями або оновленнями. Враховуючи ці особливості доцільним є використання геш-функції HDG-256 з 32-розрядним LFSR, апаратна складність якої становить 1680 GE (див. табл. 2.2).

3.1.3 Контроль цілісності даних RFID-мітки

Іншим прикладним сценарієм контролю цілісності даних у малоресурсних IoT-системах є перевірка незмінності інформації, що зберігається в RFID-мітці та передається під час її зчитування. На відміну від розглянутих вище прикладів організації контролю цілісності, де дані телеметрії формуються динамічно, а образ прошивки передається одноразово, дані RFID-мітки, як правило, записуються під час її ініціалізації і залишаються незмінними протягом усього терміну використання. RFID-мітки можуть містити інформацію про товар, об’єкт обліку або параметри пристрою і, хоча здебільшого подібні дані не є конфіденційними, вони потребують захисту від несанкціонованої модифікації.

Базовий механізм контролю цілісності для RFID-мітки може бути організований аналогічно до (3.1): під час її ініціалізації обчислюється геш-значення $H(M)$ для збережених даних M, яке записується разом із ними у пам’ять RFID-мітки. Під час зчитуванні RFID-мітки система обчислює геш-значення від отриманих даних і порівнює його зі збереженим. Однак такий підхід сам по собі не забезпечує автентифікації сторін взаємодії та не захищає від атак повторного відтворення, оскільки зловмисник може перехопити пару $(M, H(M))$ і використати її для імітації легітимної RFID-мітки.

Тому для систем, у яких необхідно поєднати контроль цілісності даних RFID-мітки зі взаємною автентифікацією сторін в умовах жорстких обмежень на апаратні ресурси, доцільним є використання спеціалізованого протоколу, що інтегрує обидві функції в межах єдиної схеми обміну даними, що пропонується у підрозділі 3.2.

3.2 Протокол автентифікації RFID-міток з контролем цілісності даних

Технологія радіочастотної ідентифікації (RFID) є однією з найбільш поширених технологій автоматичної ідентифікації в IoT-інфраструктурі [31]. Технологія RFID ґрунтується на бездротовій взаємодії між RFID-міткою, розміщеною на об’єкті, та зчитувачем, який ініціює сеанс зв’язку і передає отримані дані до серверної частини системи. Загальну схему взаємодії учасників RFID-системи наведено на рисунку 3.3.

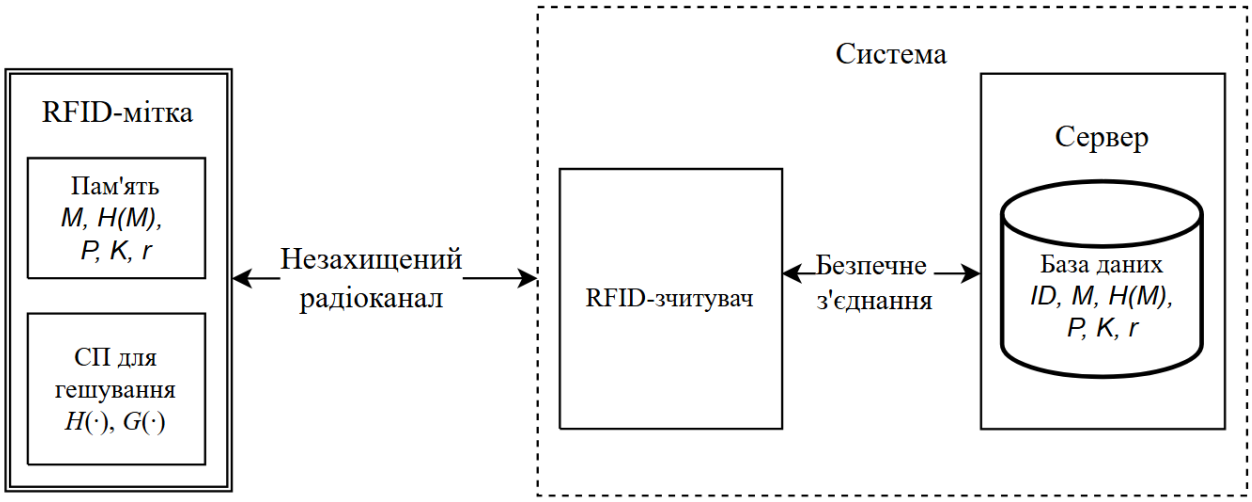

Рисунок 3.3 – Схема взаємодії між RFID-міткою, зчитувачем та сервером

RFID-система складається з трьох основних компонентів. RFID-мітка розміщується на об’єкті ідентифікації та містить пам’ять для зберігання даних M, геш-значення $H(M)$ та одноразових параметрів протоколу (P, K, r), а також спеціалізований процесор для обчислення геш-функції $H(\cdot)$ та функції оновлення псевдовипадкового числа $G(\cdot)$. Зчитувач ініціює сеанс зв’язку з RFID-міткою через незахищений радіоканал та виконує роль посередника між міткою

та сервером. Сервер містить базу даних де зберігаються записи для кожної зареєстрованої RFID-мітки, що включають ідентифікатор ID, дані M, геш-значення $H(M)$ та одноразові параметри (P, K, r). Обмін даними між зчитувачем та сервером здійснюється через захищений канал зв'язку, тому в межах протоколу вони розглядаються як єдине ціле, що далі позначається терміном «система».

Обмін даними між RFID-міткою та зчитувачем здійснюється через відкритий радіоканал, що створює загрози інформаційній безпеці, оскільки повідомлення можуть бути перехоплені, модифіковані або повторно використані зловмисником [49, 93, 95]. Пасивні RFID-мітки, що отримали найбільш масове поширення завдяки низькій вартості виробництва, не мають власного джерела живлення і живляться виключно від електромагнітного поля зчитувача [95]. Відповідно до специфікації EPC Gen2 [95, 96] обсяг логічних вентилів, що відводиться на криптографічну складову пасивної мітки, як правило, не перевищує 2000 GE, що обмежує можливість застосування стандартних криптографічних примітивів.

Переважна більшість існуючих протоколів автентифікації для RFID-міток [94-103] забезпечує взаємну автентифікацію учасників взаємодії та стійкість до типових мережевих атак. Водночас серед розглянутих підходів здебільшого відсутні рішення, що також забезпечують цілісність інформації, яка зберігається та передається RFID-міткою. Як показано у підрозділі 3.1.3, базовий механізм контролю цілісності на основі збереженого геш-значення не забезпечує автентифікації сторін та не захищає від атаки повторного відтворення, що свідчить про потребу в розробці протоколу, який поряд із взаємною автентифікацією забезпечував би також і перевірку цілісності даних RFID-мітки.

Модель загроз. Запропонований протокол орієнтований на застосування, у яких дані M, що зберігаються на RFID-мітці, не є конфіденційними, але потребують контролю цілісності. Передбачається, що порушник має можливість прослуховувати радіоканал між RFID-міткою та зчитувачем, перехоплювати

повідомлення, модифікувати їх, повторно відтворювати та блокувати передачу. Канал зв’язку між зчитувачем та сервером вважається захищеним.

У межах зазначеної моделі розглядаються такі атаки: підробка зчитувача, підробка RFID-мітки, повторне відтворення перехоплених повідомлень, активна атака типу «людина посередині» (модифікація повідомлень під час передавання) та порушення синхронізації одноразових параметрів між міткою та системою. Атаки фізичного клонування RFID-мітки, фізичного зчитування вмісту її пам’яті та атаки ретрансляції виходять за межі розглянутої моделі загроз і потребують застосування додаткових апаратних механізмів захисту. Забезпечення анонімності RFID-мітки також не є метою розробленого протоколу.

Автор пропонує протокол, що реалізує взаємну автентифікацію між RFID-міткою та системою із застосуванням криптографічної геш-функції $H(\cdot)$ та функцію оновлення псевдовипадкового числа $G(\cdot)$. Безпека протоколу ґрунтується на властивості незворотності та стійкості до колізій обраної геш-функції. Перелік умовних позначень, що використовуються в описі протоколу, наведено у таблиці 3.1.

Таблиця 3.1 – Перелік умовних позначень використаних у протоколі

Позначення	Опис
$H(\cdot)$	Криптографічна геш-функція
$G(\cdot)$	Функція оновлення псевдовипадкового числа
M	Інформація, що зберігається RFID-міткою
$H(M)$	Геш-значення інформації
$P_i^{(\text{с})}, P_i^{(\text{м})}, K_i^{(\text{с})}, K_i^{(\text{м})}, r_i^{(\text{с})}, r_i^{(\text{м})}$	Одноразові параметри для *i*-ї сесії на стороні системи та RFID-мітки відповідно
σ_i	Одноразові автентифікаційні дані для *i*-ї сесії
R_i	Відповідь від RFID-мітки для *i*-ї сесії

Протокол передбачає два етапи: ініціалізацію та автентифікацію.

Етап ініціалізації виконується один раз під час додавання нової RFID-мітки до системи. Основною метою цього етапу є встановлення між RFID-міткою та

базою даних системи спільних секретних одноразових параметрів, що використовуватимуться для подальших сесій автентифікації. Для кожної нової RFID-мітки у базі даних сервера додається новий запис, що містить такі поля:

$$\{ID,\ M,\ H(M),\ P_0,\ K_0,\ r_0\},$$

де ID – унікальний ідентифікатор RFID-мітки в системі, а P_0, K_0, r_0 – початкові значення одноразових параметрів. Водночас RFID-мітка зберігає такі параметри:

$$\{M,\ H(M),\ P_0,\ K_0,\ r_0\}.$$

Початкові значення P_0, K_0, r_0 генеруються сервером незалежно одне від одного псевдовипадковим чином. Передача початкових параметрів до RFID-мітки здійснюється шляхом фізичної прошивки її пам'яті, що унеможливлює перехоплення початкових параметрів по радіоканалу:

$$\text{Система}\rightarrow\text{RFID: } \{M,\ H(M),\ P_0,\ K_0,\ r_0\}. \quad (3.3)$$

Обов'язковою умовою коректності протоколу є $P_0 \neq K_0$, що забезпечує незалежність двох ланцюжків, які використовуються в протоколі, а саме ланцюжка підтвердження автентичності системи RFID-міткою з використанням одноразового параметра $P_i^{(\text{м})}$ та ланцюжка підтвердження автентичності RFID-мітки системою на основі одноразового параметра $K_i^{(\text{с})}$.

Перед початком кожної нової сесії автентифікації RFID-мітка проходить процедуру радіочастотної ідентифікації, за результатами якої система отримує ідентифікатор *ID* RFID-мітки та завантажує відповідний запис із бази даних. Ця процедура не є складовою протоколу і тому не розглядається в його межах.

Схему протоколу автентифікації наведено на рисунку 3.4. Процес автентифікації для *i*-ї сесії передбачає виконання декількох кроків.

Крок 1. Система обчислює значення одноразового параметра $P_i^{(\text{с})}$ та надсилає його до RFID-мітки:

$$P_i^{(\text{с})} = H(P_{i-1}^{(\text{с})} || r_{i-1}^{(\text{с})}). \quad (3.4)$$

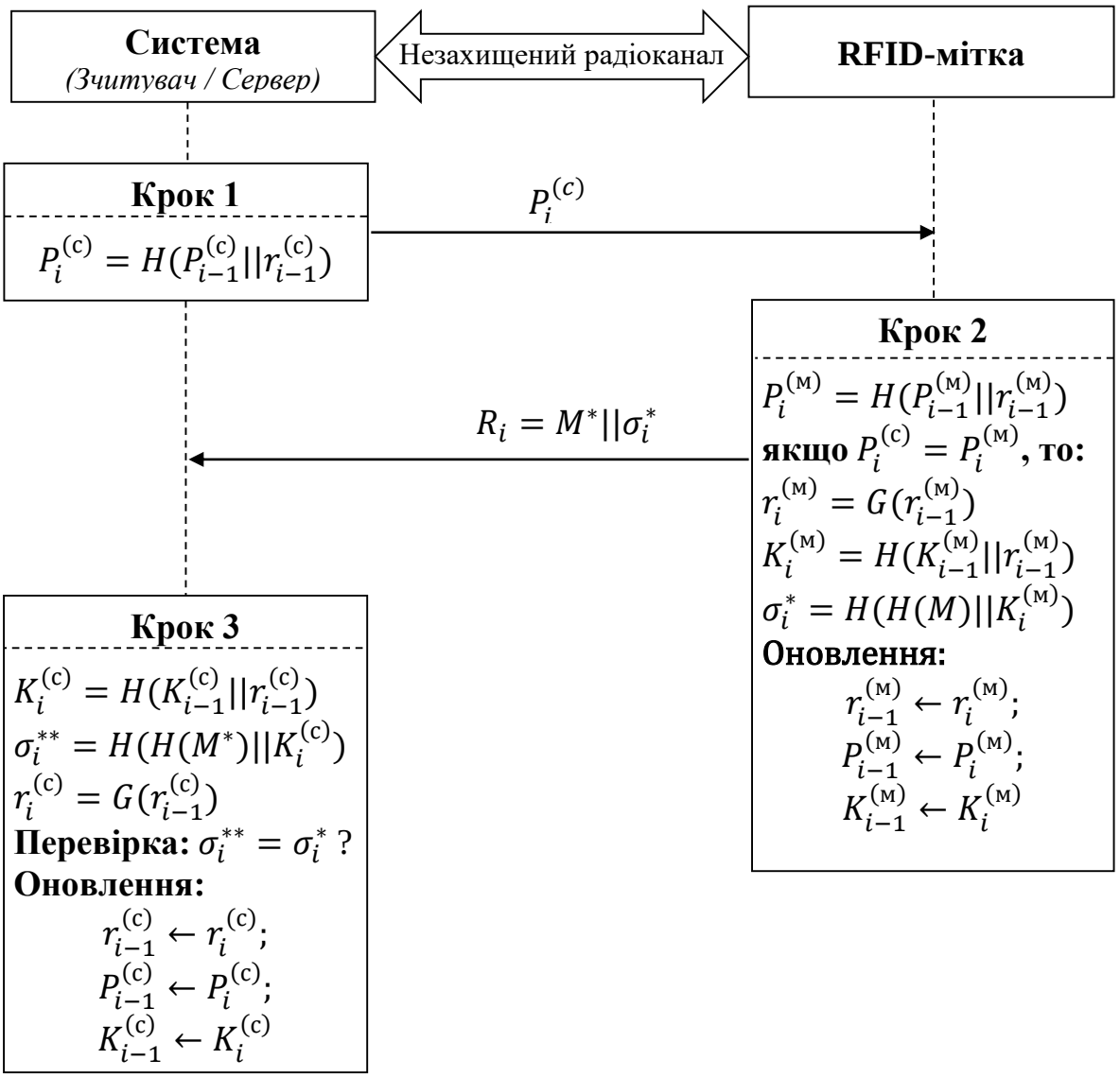

Рисунок. 3.4 – Схема протоколу автентифікації

Значення $P_i^{(c)}$ забезпечує підтвердження автентичності системи для RFID-мітки, оскільки його обчислення передбачає знання одноразових параметрів $P_{i-1}^{(c)}, r_{i-1}^{(c)}$ попередньої сесії.

Крок 2. RFID-мітка обчислює очікуване значення параметра $P_i^{(м)}$ на основі збережених у неї значень $(P_{i-1}^{(м)}, r_{i-1}^{(м)})$ з попередньої сесії:

$$P_i^{(м)} = H(P_{i-1}^{(м)}||r_{i-1}^{(м)}). \quad (3.5)$$

RFID-мітка перевіряє виконання умови $P_i^{(c)} = P_i^{(м)}$. Якщо $P_i^{(c)} \neq P_i^{(м)}$, то RFID-мітка ігнорує запит і не надсилає жодної відповіді. Якщо $P_i^{(c)} = P_i^{(м)}$, то

RFID-мітка переконалася в автентичності системи, після чого виконує обчислення параметра $K_i^{(м)}$ на основі якого формує одноразові автентифікаційні дані:

$$r_i^{(м)} = G(r_{i-1}^{(м)}), \tag{3.6}$$

$$K_i^{(м)} = H(K_{i-1}^{(м)} || r_{i-1}^{(м)}), \tag{3.7}$$

$$\sigma_i = H(H(M) || K_i^{(м)}). \tag{3.8}$$

Значення σ_i пов'язує разом автентичність RFID-мітки та цілісність її даних M. RFID-мітка надсилає системі відповідь $R_i = M || \sigma_i$ та оновлює значення одноразових параметрів $(P_{i-1}, K_{i-1}, r_{i-1})$:

$$r_{i-1}^{(м)} \leftarrow r_i^{(м)}; P_{i-1}^{(м)} \leftarrow P_i^{(м)}; K_{i-1}^{(м)} \leftarrow K_i^{(м)}. \tag{3.9}$$

Крок 3. Отримавши $R_i^* = M^* || \sigma_i^*$, система обчислює:

$$K_i^{(с)} = H(K_{i-1}^{(с)} || r_{i-1}^{(с)}), \tag{3.10}$$

$$\sigma_i^{**} = H(H(M^*) || K_i^{(с)}), \tag{3.11}$$

$$r_i^{(с)} = G(r_{i-1}^{(с)}). \tag{3.12}$$

Якщо $\sigma_i^{**} = \sigma_i^*$, то автентичність RFID-мітки підтверджено системою, а отримані дані M є цілісними. Система оновлює значення $(P_{i-1}^{(с)}, K_{i-1}^{(с)}, r_{i-1}^{(с)})$ в базі даних:

$$r_{i-1}^{(с)} \leftarrow r_i^{(с)}; P_{i-1}^{(с)} \leftarrow P_i^{(с)}; K_{i-1}^{(с)} \leftarrow K_i^{(с)}. \tag{3.13}$$

Якщо $\sigma_i^{**} \neq \sigma_i^*$ або зафіксовано відсутність відповіді на крок 1 протягом відведеного часу очікування, то значення $(P_{i-1}^{(с)}, K_{i-1}^{(с)}, r_{i-1}^{(с)})$ в базі даних залишаються незмінними.

Механізм відновлення синхронізації. Відсутність відповіді від RFID-мітки

може свідчити про можливе порушення синхронізації. RFID-мітка могла оновити значення $(P_{i-1}^{(м)}, K_{i-1}^{(м)}, r_{i-1}^{(м)})$ на крок уперед після попередньої сесії, відповідь якої не дійшла до системи. Оскільки різниця станів у такому разі завжди складає рівно одну сесію, протокол передбачає механізм відновлення синхронізації, що полягає в додатковій спробі автентифікації. Схему протоколу автентифікації у разі порушення синхронізації наведено на рисунку 3.5.

Кроки спроби 1 завершуються інформацією про порушення синхронізації. Спроба 2 (відновлення синхронізації) передбачає реалізацію кроків 4-6.

Крок 4. Система обчислює одноразовий параметр $P_{i+1}^{(с)}$ для наступної сесії, використовуючи обчислені на кроці 1 попередні значення $P_i^{(с)}$ та $r_i^{(с)}$, обчислені за формулами (3.5), (3.12):

$$P_{i+1}^{(с)} = H(P_i^{(с)} || r_i^{(с)}), \tag{3.14}$$

та надсилає $P_{i+1}^{(с)}$ до RFID-мітки.

Крок 5. При порушенні синхронізації в пам'яті RFID-мітки збережені значення $(P_i^{(м)}, K_i^{(м)}, r_i^{(м)})$, що є результатом завершення i-ї сесії. RFID-мітка обчислює очікуване значення $P_{i+1}^{(м)}$ за формулою (3.5) та порівнює з параметром $P_{i+1}^{(с)}$, отриманим від системи. У разі успішної перевірки RFID-мітка здійснює обчислення відповідно до формул (3.6), (3.7), (3.8) для значення i+1, формує результат $R_{i+1} = M||\sigma_{i+1}$ та надсилає до системи. Після цього RFID-мітка виконує оновлення значень:

$$r_i^{(м)} \leftarrow r_{i+1}^{(м)}; P_i^{(м)} \leftarrow P_{i+1}^{(м)}; K_i^{(м)} \leftarrow K_{i+1}^{(м)}. \tag{3.15}$$

Крок 6. Отримавши від RFID-мітки $R_{i+1}^* = M^*||\sigma_{i+1}^*$, система обчислює:

$$K_{i+1}^{(с)} = H(K_i^{(с)} || r_i^{(с)}), \tag{3.16}$$

$$\sigma_{i+1}^{**} = H(H(M^*)||K_{i+1}^{(с)}). \tag{3.17}$$

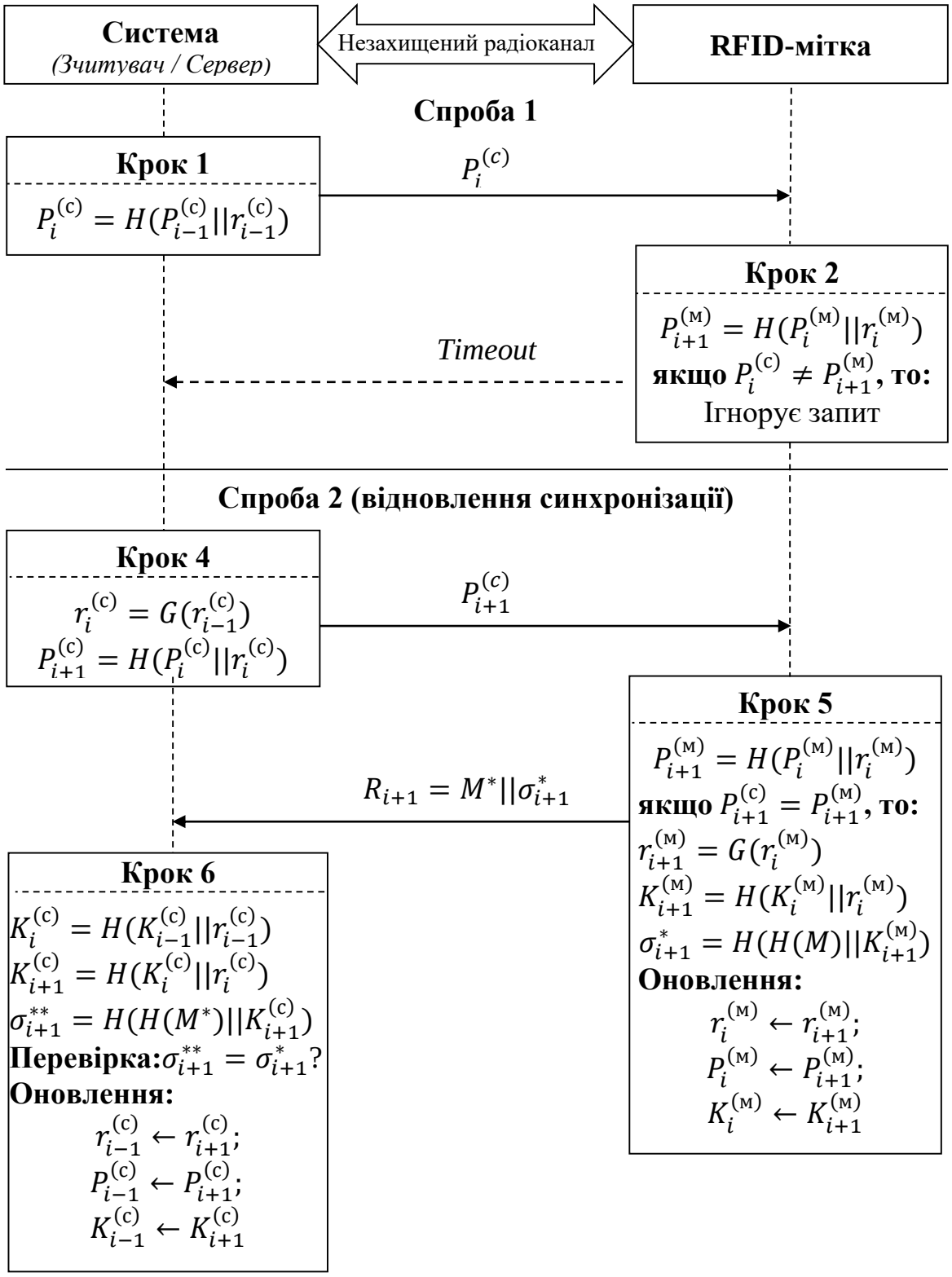

Рисунок 3.5 – Схема протоколу автентифікації у разі порушення синхронізації

Якщо $\sigma_{i+1}^{**} = \sigma_{i+1}^{*}$, то автентифікація є успішною та синхронізацію відновлено. Система оновлює значення в базі даних до $(P_{i+1}^{(c)}, K_{i+1}^{(c)}, r_{i+1}^{(c)})$, де $r_{i+1}^{(c)} = G(r_i^{(c)})$. Якщо друга спроба завершилась відсутністю відповіді або

помилкою при перевірці, то протокол завершується з помилкою автентифікації, а оновлення значень в базі даних системи не відбувається і наступна сесія розпочнеться з тих самих значень $(P_{i-1}^{(c)}, K_{i-1}^{(c)}, r_{i-1}^{(c)})$.

Протокол не прив'язаний до конкретної геш-функції. Будь-яка функція, що задовольняє вимогам незворотності та стійкості до колізій, є прийнятною. Єдиним практичним обмеженням до геш-функції є складність її апаратної реалізації, оскільки для пасивних RFID-міток обсяг логічних вентилів, що відводиться на криптографічну складову, як правило, не перевищує 2000 GE. Функція $G(\cdot)$ повинна бути відома обом сторонам протоколу. Найпростішим варіантом є реалізація у вигляді лічильника $G(r) = r + 1$, однак для більшого рівня безпеки бажано, щоб значення r_i суттєво відрізнялось від попереднього r_{i-1} для кожної сесії. Тому рекомендованим є використання конструкцій на основі LFSR, що є простими при апаратній реалізації та забезпечують краще оновлення псевдовипадкового числа між сесіями. Безпека протоколу не залежить від непередбачуваності $G(\cdot)$, оскільки значення r_i ніколи не передається каналом зв'язку у відкритому вигляді, а захист повністю забезпечується незворотністю $H(\cdot)$.

3.3 Аналіз безпеки розробленого протоколу автентифікації

Аналіз стійкості запропонованого протоколу до основних атак на RFID-системи базується на структурних властивостях протоколу та криптографічних властивостях геш-функції $H(\cdot)$.

Взаємна автентифікація. Протокол забезпечує взаємну автентифікацію сторін. Система підтверджує свою автентичність RFID-мітці за допомогою одноразового параметра $P_i^{(c)}$. Лише легітимна система, що зберігає параметри попередньої сесії $(P_{i-1}^{(c)}, r_{i-1}^{(c)})$ здатна обчислити коректне значення одноразового параметра для i-ї сесії. У свою чергу, RFID-мітка підтверджує свою автентичність системі через автентифікаційні дані $\sigma_i = H(H(M)||K_i^{(м)})$. Лише легітимна RFID-мітка, що володіє одноразовим параметром $K_i^{(м)}$, може

сформувати коректне значення σ_i.

Стійкість до атаки підробки зчитувача. Для ініціювання i+1 сесії система повинна надіслати RFID-мітці коректне значення $P_{i+1}^{(c)} = H(P_i^{(\text{с})}||r_i^{(\text{с})})$. Хоча значення $P_i^{(\text{с})}$ передається каналом зв'язку відкрито і може бути перехоплене під час попередньої i-ї сесії, для обчислення наступного значення ланцюжка необхідно знати $r_i^{(\text{с})} = G(r_{i-1}^{(\text{с})})$, яке ніколи не передається каналом зв'язку. Задача знаходження значення $r_{i-1}^{(\text{с})}$ із перехопленого повідомлення $P_i^{(c)} = H(P_{i-1}^{(\text{с})}||r_{i-1}^{(\text{с})})$ зводиться до задачі знаходження прообразу геш-функції. За умови криптографічної стійкості $H(\cdot)$ така задача є обчислювально нездійсненною, тому зловмисник не може сформувати коректне значення $P_{i+1}^{(c)}$.

Стійкість до атаки підробки RFID-мітки. Формування коректного автентифікаційного значення $\sigma_i = H(H(M)||K_i^{(\text{м})})$ для і-ї сесії потребує знання повідомлення RFID-мітки та секретного одноразового параметра $K_i^{(\text{м})}$. Хоча повідомлення M передається відкритим каналом зв'язку і може бути перехоплене під час попередньої сесії, значення одноразового параметра $K_i^{(\text{м})}$ ніколи не передається під час реалізації протоколу. Його обчислення здійснюється рекурсивно відповідно до співвідношення $K_i^{(\text{м})} = H(K_{i-1}^{(\text{м})}||r_{i-1}^{(\text{м})})$, де значення $r_{i-1}^{(\text{м})}$ також не передається каналом зв'язку. Таким чином, формування коректного значення σ_i без знання параметрів $(K_{i-1}^{(\text{м})}, r_{i-1}^{(\text{м})})$, використаних у попередній сесії, є неможливим. Відновлення цих параметрів із перехоплених повідомлень також зводиться до задачі знаходження прообразу геш-функції, тому зловмисник не може сформувати коректну відповідь RFID-мітки.

Стійкість до атаки повторного відтворення. Кожна i-та сесія використовує власні значення $P_i^{(c)}, K_i^{(\text{м})}$ та σ_i, які оновлюються після кожної успішної автентифікації. Повторне використання перехопленої відповіді $R_i = M||\sigma_i$ у сесії i+1 буде відхилено системою, оскільки перевірка виконуватиметься із використанням оновленого параметра $K_{i+1}^{(\text{с})}$ для якого формується нове значення

автентифікаційних даних, що буде відрізнятися від σ_i.

Пряма секретність. Компрометація параметрів $(P_i^{(с)}, K_i^{(с)}, r_i^{(с)}, P_i^{(м)}, K_i^{(м)}, r_i^{(м)})$, які використовувалися у i-й сесії, не дозволяє відновити параметри попередніх сесій. Зокрема, відновлення значення $K_{i-1}^{(м)}$ зі співвідношення $K_i^{(м)} = H(K_{i-1}^{(м)} || r_{i-1}^{(м)})$ зводиться до задачі знаходження прообразу геш-функції, що унеможливлює ретроспективне розкриття повідомлень, перехоплених у попередніх сесіях.

Контроль цілісності даних. Протокол забезпечує контроль цілісності інформації M, що зберігається і передається RFID-міткою. Автентифікаційне значення $\sigma_i = H(H(M) || K_i^{(м)})$ криптографічно пов'язує інформацію M з одноразовим параметром $K_i^{(м)}$, який використовується у відповідній i-й сесії. Будь-яка зміна M під час передачі призведе до невиконання умови $\sigma_i^{**} \neq \sigma_i^{*}$, при перевірці на стороні системи. Формування іншого повідомлення M', для якого $H(M') = H(M)$, зводиться до задачі знаходження колізії геш-функції і є обчислювально нездійсненним.

Стійкість до активної атаки типу «людина посередині». Будь-яка модифікація переданих повідомлень призводить до порушення перевірок автентичності. Підміна значення $P_i^{(c)}$ призведе до його невідповідності з $P_i^{(м)}$, незалежно обчисленому на стороні RFID-мітки для тієї ж i-ї сесії, внаслідок чого запит буде відхилено. Аналогічно, підміна $R_i = M || \sigma_i$ призведе до невідповідності автентифікаційних даних при перевірці на стороні системи.

Стійкість до порушення синхронізації. Порушення синхронізації може бути наслідком, наприклад, блокування відповіді RFID-мітки. Для запобігання цьому система виконує повторну спробу автентифікації (див. рис. 3.5), переходячи до параметрів $(P_{i+1}^{(с)}, K_{i+1}^{(с)}, r_{i+1}^{(с)})$ та повторюючи запит. Оскільки різниця між параметрами, що використовуються сторонами після одиночного збою становить один крок ланцюжка, синхронізація відновлюється автоматично.

Порівняння властивостей безпеки запропонованого та відомих протоколів автентифікації для RFID-міток наведено у таблиці 3.2.

Таблиця 3.2 – Властивості безпеки протоколів автентифікації для RFID-міток

Властивість	SASI [98]	CR-Triggered Hash [99]	Dass & Om [100]	EPC Gen2 PRNG [101]	ULMAP [102]	Запропонований протокол
Взаємна автентифікація	так	так	так	так	так	так
Стійкість до підробки зчитувача	так	так	так	так	так	так
Стійкість до підробки RFID-мітки	так	так	так	так	так	так
Стійкість до replay-атак	так	так	так	так	так	так
Стійкість до активної MITM	так	так	так	так	так	так
Пряма секретність	ні	так	так	так	так	так
Стійкість до порушення синхронізації	ні	так	так	ні	так	так
Контроль цілісності даних	ні	ні	ні	ні	ні	так

Як видно з таблиці 3.2, запропонований протокол задовольняє усім переліченим властивостям. Лише запропонований протокол автентифікації забезпечують контроль цілісності даних. Це визначає перевагу нового протоколу для застосувань, у яких передані дані RFID-мітки не є конфіденційними, але потребують обов’язкового контролю цілісності даних.

3.4 Оцінка обчислювальних витрат протоколу автентифікації

Обчислювальні витрати запропонованого протоколу на стороні RFID-мітки визначаються кількістю базових криптографічних операцій, що виконуються протягом однієї сесії автентифікації. В запропонованому протоколі для звичайної сесії на стороні RFID-мітки виконуються перевірка одноразового

параметра $P_i^{(с)}$, оновлення параметрів $(P_i^{(м)}, K_i^{(м)}, r_i^{(м)})$ та формування автентифікаційних даних σ_i. З урахуванням формул (3.5), (3.6), (3.7) та (3.8), зазначена послідовність операцій передбачає три обчислення геш-функції $H(\cdot)$ та одне обчислення $G(\cdot)$. Тому обчислювальні витрати на стороні RFID-мітки для однієї сесії автентифікації визначаються виразом $C_{м} = 3C_H + C_G$, де C_H – витрати на одне обчислення геш-функції, а C_G – витрати на одне обчислення функції $G(\cdot)$. Для порівняння з відомими протоколами у таблиці 3.3 наведено перелік операцій, що виконуються на стороні RFID-мітки, їх узагальнену оцінку у вигляді обчислювальних витрат, а також кількість обмінів між RFID-міткою та системою в межах однієї сесії автентифікації.

Таблиця 3.3 – Порівняльні оцінки протоколів автентифікації на стороні RFID-мітки

Протокол	Операції	Оцінки обчислювальної складності	Комунікаційна складність
SASI [98]	XOR, ADD, ROT, OR	$8C_{XOR} + 3C_{ADD} + 2C_{ROT} + C_{OR}$	4
CR-Triggered Hash[99]	H, RNG	$3C_H + C_{RNG}$	3
Dass & Om [100]	H, RNG, XOR	$2C_H + 4C_{RND} + C_{XOR}$	3
EPC Gen2 PRNG [101]	$PRNG, XOR$	$4C_{PRNG}^{EPC} + 6C_{XOR}$	3
ULMAP [102]	XOR, AND, OR, ROT	$17C_{XOR} + 4C_{AND} + 4C_{ROT}+2C_{OR}$	3
RAFI [103]	$H, XOR, E/D$	$2C_H + 6C_{XOR} + C_{E/D}$	2
Запропонований	H, G	$3C_H + C_G$	2

Примітка: C_H – витрати на одне обчислення геш-функції; C_G – витрати на одне обчислення функції $G(\cdot)$; C_{RNG} – витрати на генерацію випадкового числа; C_{PRNG}^{EPC} – витрати на спеціалізований генератор псевдовипадкових чисел, сумісний з EPC Gen2; C_{XOR} – витрати на одну операцію додавання за модулем два; C_{ROT} – витрати на одну операцію циклічного зсуву; $C_{E/D}$ – витрати на одну операцію зашифрування або розшифрування.

Запропонований протокол характеризується помірними обчислювальними витратами на стороні RFID-мітки та потребує лише двох взаємодій у межах звичайної сесії автентифікації. На відміну від надлегких протоколів [98] та [102], запропонований протокол забезпечує вищий рівень криптографічної стійкості за рахунок використання геш-функції. Водночас, порівняно з [100], [101] та [103], протокол має простішу обчислювальну структуру, оскільки не потребує генератора випадкових чисел, додаткових елементів XOR або блоку зашифрування чи розшифрування на стороні RFID-мітки. Таким чином, запропонований протокол можна розглядати як компромісне рішення для пасивних RFID-міток, у яких необхідно поєднати помірні обчислювальні витрати, малу кількість взаємодій та контроль цілісності переданих даних.

Оцінка апаратної складності реалізації протоколу на RFID-мітці. Для виконання протоколу на стороні RFID-мітки необхідні такі функціональні компоненти: спеціалізований процесор для обчислення $H(\cdot)$, засіб обчислення $G(\cdot)$ та логіка порівняння одноразових параметрів.

Спеціалізований HDG-процесор для обчислення геш-функції з довжиною геш-значення l = 128 біт має апаратну складність до 998 GE (див. табл. 2.2). Протокол потребує трьох обчислень геш-функції за сесію, однак ці обчислення виконуються послідовно, а апаратні ресурси обмежені, тому достатньо одного екземпляра спеціалізованого процесора, який заново ініціалізується для обчислення нового геш-значення.

Функція $G(\cdot)$ може бути реалізована шляхом повторного використання блоку LFSR, що вже є складовою HDG-процесора. Після обчислення геш-значення LFSR звільняється і може бути використаний для виконання обчислення $G(r) = LFSR(r)$. Додаткові апаратні витрати на реалізацію $G(\cdot)$ за такого підходу відсутні.

Порівняння одноразових параметрів $P_i^{(\text{с})} = P_i^{(\text{м})}$ реалізується побайтним компаратором, що порівнює по одному байту за такт і нагромаджує результат в одному тригері. Апаратна складність такого компаратора становить приблизно

29 GE. Додаткова логіка керування протоколом (послідовність подачі вхідних даних на HDG-процесор, формування відповіді) оцінюється приблизно у 30–50 GE. Параметри протоколу $(M, H(M), P, K, r)$ зберігаються у незалежній пам'яті RFID-мітки (EEPROM), яка не враховується в оцінці GE, оскільки реалізується окремим блоком. Сумарну оцінку апаратної складності реалізації протоколу на RFID-мітці наведено у таблиці 3.4.

Таблиця 3.4 – Оцінка апаратної складності реалізації протоколу на RFID-мітці

Компонент	**Апаратна складність, GE**
HDG-процесор (l = 128)	998
G(·) через повторне використання LFSR	0
Побайтний компаратор	~29
Логіка керування протоколом	~30–50
Разом	~1057–1077

Отримана оцінка підтверджує, що запропонований протокол може бути реалізований на пасивних RFID-мітках відповідно до специфікації EPC Gen2, оскільки сумарна апаратна складність не перевищує обмеження 2000 GE.

3.5 Висновки до розділу 3

1. Розглянуті сценарії застосування розроблених методів та спеціалізованих процесорів для контролю цілісності даних у малоресурсних IoT-системах показали, що метод HDG-128 з LFSR зменшеної розрядності забезпечує найменші апаратні витрати при достатній пропускній здатності для контролю цілісності телеметричних повідомлень сенсорних вузлів, а метод HDG-256 з 32-розрядним LFSR є придатним для верифікації програмного забезпечення при завантаженні IoT-пристрою завдяки достатньому періоду генератора та побайтній потоковій обробці даних. Схему контролю цілісності телеметричних даних на основі розроблених методів впроваджено у ТОВ «Каскад-Безпека» у системі IoT-моніторингу внутрішніх ресурсів підприємства.

2. Розроблено протокол автентифікації для RFID-систем, який, на відміну від існуючих, одночасно забезпечує взаємну автентифікацію сторін взаємодії та контроль цілісності даних, що зберігаються і передаються RFID-міткою. Показано, що для виконання однієї сесії автентифікації на стороні RFID-мітки достатньо трьох обчислень геш-функції та одного обчислення функції оновлення псевдовипадкового числа при двох обмінах повідомленнями. Протокол автентифікації впроваджено у ТОВ «Солар Стар АГРО» у системі ідентифікації на основі RFID-міток.

3. Аналіз безпеки протоколу підтвердив його стійкість до атак підробки зчитувача та RFID-мітки, повторного відтворення, активної атаки типу «людина посередині» та порушення синхронізації, а також забезпечення прямої секретності. Порівняння з відомими протоколами показало, що лише запропонований протокол забезпечує контроль цілісності даних RFID-мітки.

4. Теоретична оцінка апаратної складності реалізації протоколу з використанням спеціалізованого HDG-процесора підтвердила можливість його реалізації на пасивній RFID-мітці відповідно до специфікації EPC Gen2.

Основні наукові результати, представлені в цьому розділі, опубліковано в статті автора [35].

РОЗДІЛ 4
ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ РОЗРОБЛЕНИХ МЕТОДІВ МАЛОРЕСУРСНОГО ГЕШУВАННЯ ТА ПРОТОКОЛУ АВТЕНТИФІКАЦІЇ

4.1 Програмні засоби для експериментального дослідження методів малоресурсного гешування

4.1.1 Програмна реалізація методів малоресурсного гешування

Для проведення експериментальних досліджень розроблено програмну бібліотеку мовою високого рівня C++, яка забезпечує реалізацію методів HDG, HDGm та HGD для довжин геш-значення 128, 192 та 256 біт. На рисунку 4.1 представлено ієрархію класів цієї бібліотеки.

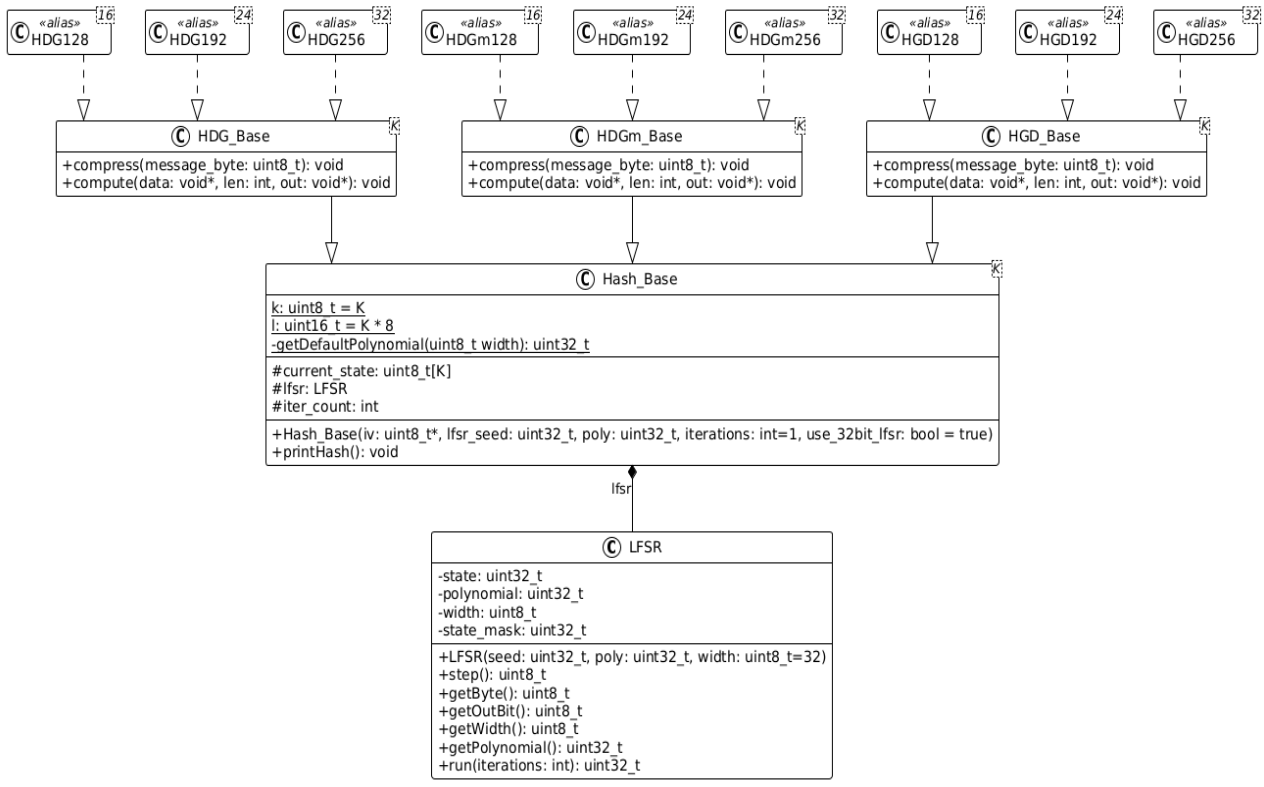

Рисунок 4.1 – UML-діаграма класів бібліотеки малоресурсного гешування

Клас *LFSR* описує генератор псевдовипадкової послідовності на основі регістра зсуву з лінійним зворотним зв’язком за конструкцією Ґалуа. Реалізація передбачає використання розрядності 16, 24 та 32 біти внутрішнього стану, що

відповідає параметрам, визначеним у підрозділі 2.2. Основними методами класу є *step*, що виконує один такт зсуву регістра, а також *getOutBit* та *getByte*, які повертають молодший біт та молодший байт стану генератора відповідно. Конструктор класу приймає початковий стан, поліном представлений у вигляді цілого числа та степінь цього полінома.

Базовий клас шаблон *Hash_Base<K>* є спільною основою для всіх класів методів гешування, де параметр шаблону *K* визначає довжину геш-значення відповідно до формули *K = l/8*. Клас містить масив *current_state* розміром *K* байтів для зберігання проміжного геш-значення, екземпляр класу *LFSR* та параметри конфігурації. Конструктор класу *Hash_Base* приймає початкове геш-значення у вигляді байтового масиву, початковий стан та поліном для *LFSR*, а також додаткові параметри *iterations* та *use_32bit_lfsr*, що використовуються для дослідження методів гешування. Параметр *iterations* дозволяє встановити кількість ітерацій, протягом яких буде обчислюватись функція ущільнення для гешування кожного байту даних. Прапорець *use_32bit_lfsr* встановлений за замовчуванням та вказує на розрядність, з якою потрібно ініціалізувати *LFSR* – 32 біти або з розрядністю *K*-біт. Якщо поліном для *LFSR* не задано, використовується один із визначених у статичному методі *getDefaultPolynomial* поліномів відповідної розрядності, що забезпечують максимальний період послідовності.

Методи гешування HDG, HDGm та HGD реалізовано у вигляді похідних від *Hash_Base* шаблонних класів *HDG_Base<K>*, *HDGm_Base<K>* та *HGD_Base<K>* відповідно, кожен з яких містить власну реалізацію методу *compress*, що перевизначає функцію ущільнення відповідно до алгоритму конкретної геш-функції та виконує гешування одного байта даних. Метод *compute* забезпечує обчислення геш-значення для повідомлення довільної довжини шляхом послідовного виклику методу compress для кожного байту повідомлення.

Для кожного з шаблонів визначено псевдоніми типів з відповідними параметрами. Зокрема, типи *HDG128, HDG192* та *HDG256* фіксують параметр

шаблону *K* на рівнях 16, 24 та 32 відповідно, що визначає вихідний розмір геш-значення у байтах. Аналогічний підхід застосовано для HDGm та HGD.

Розроблену програмну бібліотеку скомпільовано у вигляді Python-модуля під назвою *hash_module* з використанням pybind11 [104, 105]. В результаті цього *hash_module* дозволяє викликати розроблені класи геш-функцій безпосередньо з Python-коду, що забезпечує інтеграцію з програмними засобами для дослідження розроблених геш-функції. Перевірку коректності програмної реалізації методів малоресурсного гешування здійснено шляхом порівняння результатів обчислення геш-значень з результатами моделювання відповідних спеціалізованих процесорів у середовищі Logisim-evolution (підрозділ 2.3). Для набору тестових повідомлень різної довжини підтверджено повний збіг геш-значень, отриманих програмною бібліотекою та апаратною моделлю.

4.1.2 Програмний засіб для дослідження статистичних характеристик методів гешування

Для дослідження статистичних характеристик розроблених методів гешування створено програмний засіб мовою Python. Засіб автоматизує генерацію тестових даних та виконання статистичних тестів з використанням пакетів NIST STS [106] та Dieharder [107]. Програмний засіб реалізовано у вигляді застосунку з інтерфейсом командного рядка (CLI).

Програмний засіб працює у два етапи. На першому етапі формуються тестові дані шляхом обчислення геш-значень для псевдовипадкових повідомлень зазначеної довжини. Обчислені геш-значення поєднуються у єдину бітову послідовність, довжина якої є достатньою для виконання статистичних тестів. Для скорочення часу підготовки тестових даних генерацію розподілено на декілька паралельних процесів, кожен з яких формує відповідний блок тестових даних. Розпаралелення реалізовано засобами модуля *multiprocessing* мови Python [104]. Результатом першого етапу є бінарний файл, що містить сформовану послідовність геш-значень.

Другий етап полягає у взаємодії із Linux-сервером, на якому встановлено пакети NIST STS та Dieharder. Сформований файл передається на віддалений сервер із використанням протоколу SSH [108]. Для реалізації цієї взаємодії розроблено клас *SSHSession*, що забезпечує встановлення з'єднання, передавання файлів і виконання команд на віддаленому сервері засобами бібліотеки *Paramiko* [109]. Після завершення тестування програмний засіб отримує результати та зберігає їх у відповідній локальній директорії.

Геш-функція, що підлягає тестуванню, підключається до програмного засобу у вигляді окремого Python-модуля, який розміщується в директорії *modules*. Для додавання нової геш-функції достатньо створити відповідний файл, що реалізує інтерфейс виклику у вигляді класу *HASH*, який приймає параметри гешування, та методу *hash_value*, призначеного для оброблення байтового повідомлення довільної довжини з поверненням відповідного геш-значення. Розроблений програмний засіб зареєстровано як об'єкт авторського права, що підтверджується свідоцтвом про реєстрацію авторського права на твір [41], а також впроваджено в освітній процес кафедри захисту інформації ВНТУ при викладанні дисциплін «Криптографічний захист інформації» та «Прикладна криптологія», що підтверджується актом впровадження.

4.2 Дослідження статистичних характеристик методів гешування

Статистичні характеристики розроблених методів малоресурсного гешування досліджено за допомогою пакета NIST STS 800-22 [106], що містить 15 статистичних тестів для перевірки гіпотези випадковості двійкових послідовностей. Усі тести спрямовані на виявлення різних дефектів псевдовипадковості.

Для проведення одного статистичного випробування для пакету NIST STS рекомендованою є двійкова послідовність довжиною 1000000 біт. З урахуванням цього тестування розроблених методів гешування проведено для послідовностей, сформованих шляхом обчислення 1000000 геш-значень довжиною 128 біт або 500000 геш-значень довжиною 256 біт, чого достатньо для

проведення серії зі ста випробувань для одного експерименту. Геш-значення обчислювались для різних псевдовипадкових повідомлень довжиною L = 8, 16, 32, 64 байти. В усіх експериментах використовувалися однакові фіксовані початковий стан LFSR та початкове геш-значення, для коректності порівняння результатів. Відповідно до рекомендацій NIST STS [106] статистичний тест вважається успішним, якщо його успішно пройдено не менше ніж для 96 зі 100 сформованих бітових послідовностей.

Розроблені у розділі 2 методи використовують одне обчислення функції ущільнення для гешування одного байта даних. У межах дослідження введено параметр n – кількість ітерацій виконання функції ущільнення f над i-м байтом повідомлення. Цей параметр дозволяє, за необхідності, покращити внутрішнє перемішування даних геш-функції та оцінити, наскільки це впливає на статистичні характеристики геш-значень.

Досліджено статистичні характеристики методів HDG та HGD для 128-бітного геш-значення при використанні k-розрядного LFSR (k = 16).

Параметри експерименту:

– методи: HDG, HGD;
– довжина геш-значення: 128 біт;
– довжина повідомлення: L = 8, 16, 32, 64 байти;
– поліном LFSR: $x^{16} + x^{15} + x^{13} + x^{10} + x^8 + x^6 + x^4 + x^2 + 1$.

Результати таблиці 4.1 показують, що при $L = 16$ або $L = 64$, а також більших значень L, усі статистичні тести є успішними для кожного з методів за заданих параметрів експерименту. При $L = 8$, для геш-функції HDG успішними є лише 10 тестів. Таким чином, HGD-128 забезпечує потрібні статистичні характеристики геш-значень для повідомлень довжиною 8 байт і більше, тоді як HDG-128 – лише для повідомлень довжиною не менше 16 байт. Оскільки малоресурсні IoT-пристрої часто обробляють повідомлення невеликої довжини, далі досліджено можливість покращення статистичних характеристик геш-функції HDG-128 для таких повідомлень за рахунок збільшення кількості

ітерацій функції ущільнення.

Таблиця 4.1 – Результати тестування методів HDG та HGD за пакетом NIST STS для 128-бітних геш-значень при використанні 16-розрядного LFSR

Номер тесту	L = 8		L = 16		L = 64	
	HDG	HGD	HDG	HGD	HDG	HGD
1	97/100	97/100	98/100	100/100	99/100	99/100
2	**16/100**	98/100	99/100	98/100	98/100	98/100
3	99/100	97/100	99/100	99/100	99/100	100/100
4	**79/100**	100/100	99/100	97/100	99/100	100/100
5	98/100	100/100	100/100	99/100	98/100	100/100
6	**90/100**	99/100	96/100	98/100	98/100	98/100
7	98/100	99/100	99/100	99/100	99/100	99/100
8	94/100	99/100	99/100	100/100	99/100	99/100
9	99/100	98/100	100/100	98/100	98/100	100/100
10	**0/100**	99/100	100/100	97/100	100/100	100/100
11	**0/100**	99/100	100/100	99/100	99/100	99/100
12	97/100	98/100	98/100	100/100	100/100	98/100
13	58/59	55/55	59/60	54/54	64/64	67/68
14	58/59	54/55	60/60	54/54	64/64	67/68
15	97/100	100/100	98/100	98/100	97/100	97/100

Параметри експерименту:

– метод: HDG;

– довжина геш-значення: 128 біт;

– довжина повідомлення: $L = 8$;

– поліном LFSR: $x^{16} + x^{15} + x^{13} + x^{10} + x^8 + x^6 + x^4 + x^2 + 1$;

– кількість ітерацій: $n = 2 \dots 8$.

Результати таблиці 4.2 показують, що вже при $n = 2$ метод HDG забезпечує успішне проходження всіх тестів NIST STS для повідомлень довжиною $L = 8$ байт. Подальше збільшення кількості ітерацій не призводить до погіршення результатів тестування, тобто потрібні статистичні

характеристики геш-значень зберігаються і при більших значеннях n.

Таблиця 4.2 – Результати тестування методу HDG за пакетом NIST STS для 128-бітних геш-значень при L = 8 та різних значеннях n

Номер тесту	n = 2	n = 3	n = 4	n = 5	n = 6	n = 7
1	100/100	100/100	100/100	98/100	100/100	98/100
2	100/100	98/100	96/100	99/100	99/100	100/100
3	100/100	97/100	99/100	99/100	100/100	98/100
4	100/100	96/100	100/100	99/100	100/100	99/100
5	99/100	100/100	99/100	99/100	99/100	100/100
6	100/100	98/100	100/100	100/100	99/100	96/100
7	99/100	99/100	99/100	98/100	99/100	99/100
8	97/100	99/100	99/100	99/100	98/100	98/100
9	100/100	99/100	97/100	100/100	100/100	100/100
10	99/100	100/100	98/100	98/100	100/100	100/100
11	99/100	98/100	100/100	98/100	98/100	100/100
12	100/100	100/100	100/100	98/100	100/100	98/100
13	69/69	70/71	58/59	59/60	69/70	67/68
14	68/69	70/71	59/59	60/60	69/70	67/68
15	100/100	100/100	98/100	98/100	99/100	99/100

У таблиці 4.3 наведено результати статистичного тестування методів HDG та HGD для 128-бітних геш-значень при використанні 32-розрядного LFSR.

Параметри експерименту:

– методи: HDG, HGD;

– довжина геш-значення: 128 біт;

– довжина повідомлення: L = 8, 16, 32, 64 байти;

– поліном: $x^{32} + x^{31} + x^{28} + x^{23} + x^{21} + x^{17} + x^{10} + x^5 + x^3 + x + 1$.

Відповідно до таблиці 4.3 використання 32-розрядного LFSR забезпечує успішне проходження всіх тестів NIST STS методами HDG та HGD для повідомлень довжиною від 8 до 64 байт.

Таблиця 4.3 – Результати тестування методів за пакетом NIST STS для 128-бітних геш-значень при використанні 32-розрядного LFSR

Номер тесту	L = 8		L = 32		L = 64	
	HDG	HGD	HDG	HGD	HDG	HGD
1	98/100	99/100	100/100	99/100	96/100	99/100
2	98/100	100/100	99/100	100/100	99/100	99/100
3	99/100	99/100	100/100	99/100	99/100	98/100
4	100/100	98/100	99/100	99/100	100/100	99/100
5	100/100	100/100	99/100	97/100	96/100	100/100
6	98/100	99/100	98/100	98/100	99/100	97/100
7	99/100	99/100	99/100	99/100	99/100	99/100
8	99/100	96/100	99/100	99/100	99/100	97/100
9	100/100	100/100	97/100	99/100	100/100	99/100
10	100/100	100/100	98/100	100/100	99/100	99/100
11	97/100	100/100	99/100	100/100	98/100	98/100
12	99/100	98/100	100/100	99/100	96/100	99/100
13	58/59	55/56	62/62	65/65	55/56	63/64
14	59/59	56/56	61/62	65/65	56/56	63/64
15	100/100	100/100	99/100	96/100	100/100	98/100

Таким чином, для 128-бітних реалізацій геш-функцій розглянута конфігурація генератора псевдовипадкової послідовності забезпечує потрібні статистичні характеристики геш-значень. Для 256-бітних реалізацій методів HDG та HGD також використано 32-розрядний LFSR. Для початку розглянуто поліном з меншою кількістю зворотних зв’язків, який використано для моделі HDG у розділі 2.

Параметри експерименту:

– методи: HDG та HGD;

– довжина геш-значення: 256 біт;

– поліном LFSR: $x^{32} + x^{30} + x^{26} + x^{25} + 1$.

Таблиця 4.4 – Результати тестування методів за пакетом NIST STS для 256-бітних геш-значень при LFSR з кількістю зворотних зв’язків 4

Номер тесту	L = 8		L = 32		L = 64	
	HDG	HGD	HDG	HGD	HDG	HGD
1	100/100	99/100	99/100	98/100	99/100	100/100
2	98/100	**19/100**	99/100	99/100	99/100	100/100
3	100/100	100/100	100/100	100/100	99/100	100/100
4	99/100	**59/100**	98/100	**70/100**	100/100	99/100
5	**95/100**	100/100	100/100	99/100	100/100	100/100
6	99/100	99/100	98/100	100/100	100/100	**91/100**
7	99/100	**55/100**	99/100	**58/100**	99/100	**75/100**
8	99/100	**10/100**	98/100	**36/100**	100/100	**94/100**
9	99/100	98/100	100/100	96/100	99/100	98/100
10	100/100	**0/100**	98/100	**0/100**	98/100	**0/100**
11	100/100	**0/100**	99/100	**0/100**	100/100	**0/100**
12	100/100	100/100	99/100	98/100	99/100	100/100
13	63/64	52/52	60/60	54/54	61/62	68/69
14	63/64	52/52	59/60	54/54	61/62	69/69
15	100/100	**67/100**	97/100	**31/100**	100/100	**0/100**

Відповідно до таблиці 4.4 метод HDG для $L = 8$ не проходить лише один тест NIST STS, для якого отримано результат 95/100. Для більших довжин повідомлення всі тести є успішними. Водночас метод HGD за цих самих умов не забезпечує успішного проходження статистичних тестів для жодної з досліджених довжин повідомлення. Таким чином, використання LFSR з кількістю зворотних зв’язків 4 є прийнятним для методу HDG, але недостатнім для методу HGD. Тому у таблиці 4.5 представлено результати тестування при використанні іншого полінома з більшою кількістю зворотних зв’язків.

Параметри експерименту:

– методи: HDG, HGD;

– довжина геш-значення: 256 біт;

– довжина повідомлення: L = 8, 16, 32, 64 байти;

– поліном: $x^{32} + x^{31} + x^{28} + x^{23} + x^{21} + x^{17} + x^{10} + x^5 + x^3 + x + 1$.

Таблиця 4.5 – Результати тестування методів за пакетом NIST STS для 256-бітних геш-значень при LFSR з кількістю зворотних зв’язків 10

Номер тесту	***L* = 8**		***L* = 32**		***L* = 64**	
	HDG	HGD	HDG	HGD	HDG	HGD
1	99/100	98/100	98/100	99/100	98/100	100/100
2	99/100	100/100	99/100	99/100	99/100	99/100
3	99/100	100/100	98/100	100/100	99/100	100/100
4	99/100	100/100	99/100	100/100	99/100	99/100
5	100/100	99/100	100/100	100/100	100/100	98/100
6	97/100	100/100	100/100	98/100	98/100	100/100
7	99/100	99/100	99/100	99/100	99/100	99/100
8	99/100	100/100	99/100	100/100	98/100	100/100
9	100/100	99/100	99/100	100/100	97/100	99/100
10	99/100	100/100	98/100	100/100	99/100	99/100
11	96/100	96/100	100/100	98/100	100/100	100/100
12	100/100	98/100	98/100	100/100	98/100	100/100
13	60/61	57/58	60/61	64/64	62/62	60/60
14	61/61	58/58	60/61	64/64	62/62	59/60
15	97/100	100/100	97/100	99/100	98/100	99/100

Відповідно до результатів таблиці 4.5 методи HDG та HGD забезпечують успішне проходження всіх тестів NIST STS для повідомлень довжиною від 8 до 64 байт. Таким чином, для 256-бітних реалізацій така конфігурація LFSR забезпечує потрібні статистичні характеристики геш-значень для обох методів.

Окрім тестів пакета NIST STS, статистичні характеристики розроблених геш-функцій додатково досліджено з використанням тестів Diehard, реалізованих у пакеті dieharder [107]. У межах дослідження розглядалися 256-бітні геш-значення, сформовані при використанні 32-розрядного LFSR з більшою кількістю зворотних зв’язків, який за результатами попереднього

тестування (див. табл. 4.5) забезпечив успішне проходження статистичних тестів двома розробленими методами гешування. Для дослідження використано 17 тестів Diehard, перелік яких наведено в таблиці 4.6. Перелік саме цих тестів та їх нумерацію також використано надалі при графічному поданні результатів.

Таблиця 4.6 – Результати тестування методів HDG та HGD за тестами Diehard для 256-бітних геш-значень при L =8

№	Назва тесту	HDG		HGD	
		p-value	Оцінка	*p-value*	Оцінка
1	Diehard Birthdays	0.85162655	PASSED	0.686097	PASSED
2	Diehard OPERM5	0.85984323	PASSED	0.958374	PASSED
3	Diehard 32x32 Rank	0.82200788	PASSED	0.318266	PASSED
4	Diehard 6x8 Rank	0.85253846	PASSED	0.902071	PASSED
5	Diehard Bitstream	0.88981341	PASSED	0.078945	PASSED
6	Diehard OPSO	0.00665536	PASSED	0.014525	PASSED
7	Diehard OQSO	0.70299823	PASSED	0.797195	PASSED
8	Diehard DNA	**0.00119895**	**WEAK**	0.647304	PASSED
9	Diehard Count the 1s (stream)	0.90733276	PASSED	0.762476	PASSED
10	Diehard Count the 1s (byte)	0.21501218	PASSED	0.059031	PASSED
11	Diehard Parking Lot	0.48712953	PASSED	0.262731	PASSED
12	Diehard 2d Sphere	0.76193682	PASSED	0.607885	PASSED
13	Diehard 3d Sphere	0.87689433	PASSED	0.598117	PASSED
14	Diehard Squeeze	0.09002493	PASSED	0.301378	PASSED
15	Diehard Sums	0.23853843	PASSED	0.230685	PASSED
16	Diehard Runs	0.38668944 0.01264742	PASSED PASSED	0.950337 0.51674	PASSED PASSED
17	Diehard Craps	0.35270841 0.52883666	PASSED PASSED	0.729991 0.963589	PASSED PASSED

Для всіх експериментів за пакетом Diehard використано спільні параметри:

– методи: HDG, HGD;

– довжина геш-значення: 256 біт;

– поліном: $x^{32} + x^{31} + x^{28} + x^{23} + x^{21} + x^{17} + x^{10} + x^5 + x^3 + x + 1$;

– кількість обчислених геш-значень для одного експерименту: 10 000 000.

На відміну від тестування за пакетом NIST STS, у дослідженні за тестами Diehard геш-значення обчислювалися як для повідомлень фіксованої довжини, так і для повідомлень випадкової довжини в заданому діапазоні. Повідомлення довжиною $L = 8$ та $L = 64$ байти розглянуто окремо як граничні випадки досліджуваного діапазону. Для проміжних довжин результати подано для інтервалів $L = 8 \ldots 16$, $L = 16 \ldots 32$ та $L = 32 \ldots 64$, що дозволило виконати узагальнену оцінку статистичних характеристик геш-значень у межах відповідних піддіапазонів довжин без надмірного збільшення кількості експериментів.

Аналіз таблиці 4.6 показує, що метод HGD при L=8 байт успішно проходить усі 17 тестів Diehard. Для методу HDG при L=8 усі тести, окрім Diehard DNA, також є успішними. Отриманий результат свідчить про наявність локального статистичного відхилення, виявленого саме цим тестом для найкоротших розглянутих повідомлень. Водночас, з урахуванням успішного проходження решти тестів Diehard, а також результатів попереднього тестування NIST STS, цей результат не є підставою для висновку про статистичну непридатність методу HDG загалом.

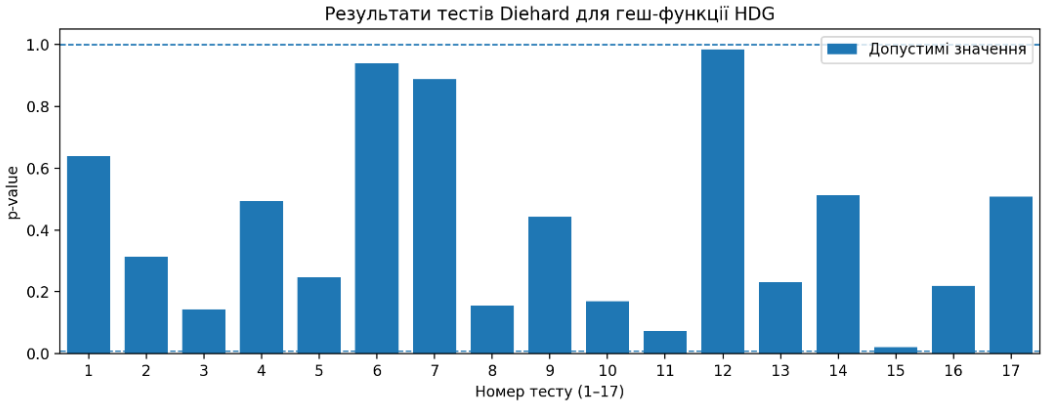

Рисунок 4.2 – Значення *p-value* тестів Diehard для методу HDG при L=8…16

Результати інших експериментів за тестами Diehard подано у вигляді діаграм значень *p-value*. По осі абсцис на рисунках відкладено номер тесту відповідно до таблиці 4.6, а по осі ординат – значення *p-value*.

Рисунок 4.3 – Значення *p-value* тестів Diehard для методу HGD при L=16…32

Рисунок 4.4 – Значення *p-value* тестів Diehard для методу HDG при L=32…64

Рисунок 4.5 – Значення *p-value* тестів Diehard для методу HDG при L=64

Рисунок 4.6 – Значення *p-value* тестів Diehard для методу HGD при *L*=64

Відповідно до рисунків 4.2–4.6, для наведених експериментів усі тести Diehard завершуються з оцінкою PASSED. Це стосується як повідомлень фіксованої довжини *L*=64, так і повідомлень випадкової довжини у вказаних інтервалах. Таким чином, для методу HGD усі розглянуті експерименти за пакетом Diehard є успішними. Метод HDG також проходить тести Diehard, однак для *L*=8 можуть виникати окремі значення типу WEAK. Одним із шляхів усунення цього є збільшення кількості ітерацій функції ущільнення.

Отже, результати тестування за пакетом Diehard узгоджуються з результатами, отриманими раніше за пакетом NIST STS, і підтверджують придатність розроблених методів до формування геш-значень з належними статистичними характеристиками.

4.3 Дослідження лавинного ефекту

Для дослідження лавинного ефекту розроблених методів гешування визначалася відстань Гемінга між геш-значеннями початкового та модифікованого повідомлень. Результат подано у вигляді відсотка змінених бітів геш-значення. Ідеальним вважалося значення, близьке до 50 %, тобто зміна приблизно половини вихідних бітів. У кожному експерименті використовувались повідомлення довжиною 64 байти. Модифікації розглядалися в усіх можливих позиціях вхідного повідомлення. При дослідженні зміни одного біта інвертувався відповідний біт, а при дослідженні зміни байтів

для кожної позиції розглядалися 50 випадкових варіантів заміни. Для порівняння в експериментах використано малоресурсну геш-функцію SPONGENT-128, яка забезпечує належний лавинний ефект. Спочатку досліджено лавинний ефект при зміні одного біта вхідного повідомлення.

Параметри експерименту:

– методи: HDG, HDGm, HGD, SPONGENT;
– тип зміни повідомлення: інверсія одного біта;
– кількість спостережень: 51200.
– довжина геш-значення: 128 біт;
– кількість ітерацій функції ущільнення для розроблених методів: $n = 1$.

Таблиця 4.7 – Результати дослідження лавинного ефекту при зміні одного біта вхідного повідомлення

Метод	Середнє значення, %	Середньоквадратичне відхилення, %	Медіана, %	Мінімум, %	Максимум, %
HDG	11.23	4.32	10.94	3.13	32.81
HDGm	22.06	5.32	21.88	12.5	42.97
HGD	49.52	6.29	50	17.19	71.8
SPONGENT	49.98	4.4	50	32.03	67.97

Відповідно до таблиці 4.7, при інверсії одного біта вхідного повідомлення метод HGD забезпечує середнє значення лавинного ефекту 49,52 %, що є близьким до результату SPONGENT-128. Середньоквадратичне відхилення для HGD є дещо більшим, ніж у SPONGENT-128, однак загалом отримані результати свідчать про належний рівень дифузії. Для методів HDGm та HDG середній відсоток змінених бітів геш-значення становить лише 22,06 % та 11,23 % відповідно, тому вони не забезпечують належного лавинного ефекту при зміні одного біта. Дослідження 256-бітних реалізацій показало аналогічні результати. На рисунку 4.7 наведено гістограму розподілу значень лавинного ефекту для HGD-256, з якої видно, що значення зосереджені поблизу 50 %.

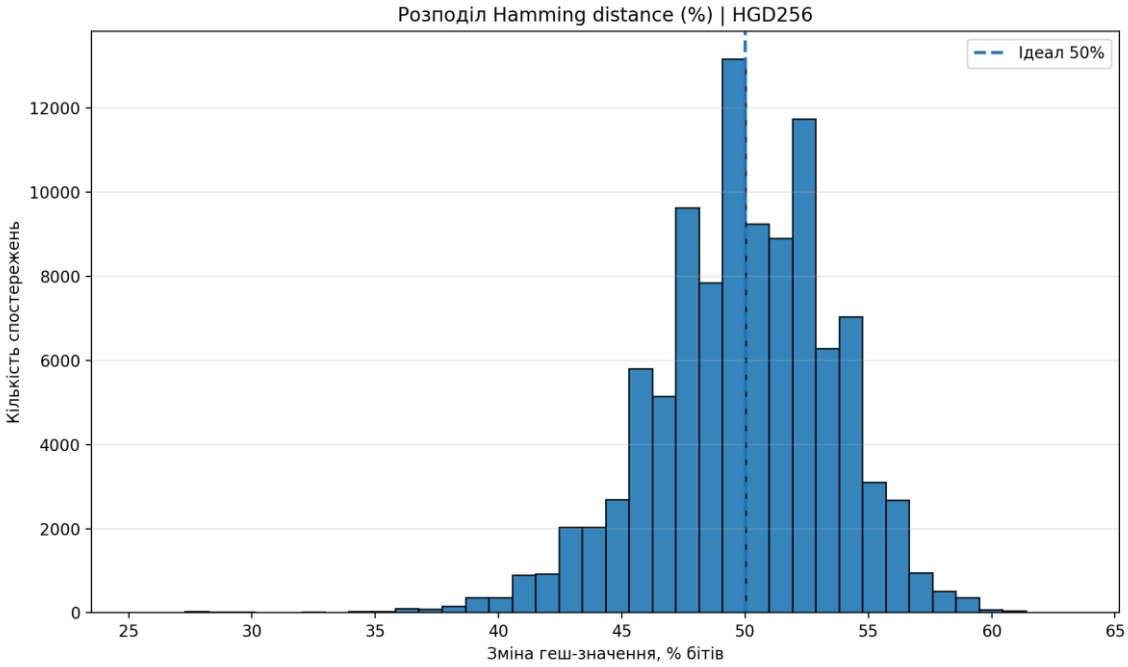

Рисунок 4.7 – Гістограма розподілу значень лавинного ефекту для методу HGD-256 при зміні одного біта вхідного повідомлення

Досліджено лавинний ефект при зміні одного байта вхідного повідомлення. Параметри експерименту:

– методи: HDG, HDGm, HGD, SPONGENT;
– тип зміни повідомлення: заміна одного байта;
– кількість спостережень: 320000.
– довжина геш-значення: 128 біт;
– кількість ітерацій функції ущільнення: $n = 1$ та $n = 7$.

Відповідно до таблиці 4.8 та рисунка 4.8, при зміні одного байта вхідного повідомлення метод HGD забезпечує значення лавинного ефекту, близьке до 50 %, і за цим показником практично відповідає SPONGENT. Для методу HDGm середнє значення також є близьким до 50 %, однак середньоквадратичне відхилення є істотно більшим. Для методу HDG при $n = 1$ середній відсоток зміни бітів геш-значення залишається недостатнім. Збільшення кількості ітерацій до $n = 7$ дозволяє покращити результат методу HDG. Середнє значення лавинного ефекту для нього зростає до 44,71 %, однак все ще залишається нижчим за бажаний рівень.

Таблиця 4.8 – Результати дослідження лавинного ефекту при зміні одного байта вхідного повідомлення

Метод	*n*	Середнє значення, %	Середньоквадратичне відхилення, %	Медіана, %	Мінімум, %	Максимум, %
HDG	1	25.86	9.47	25	3.13	69.53
HDGm	1	50.22	11.73	52.34	12.5	77.34
HGD	1	49.98	4.68	50	20.31	71.09
HDG	7	44.71	9.8	46.88	3.13	69.53
HDGm	7	50.17	10.16	52.34	12.5	75
HGD	7	50	4.41	50	31.25	70.31
SPONGENT	1	50	4.41	50	28.91	70.31

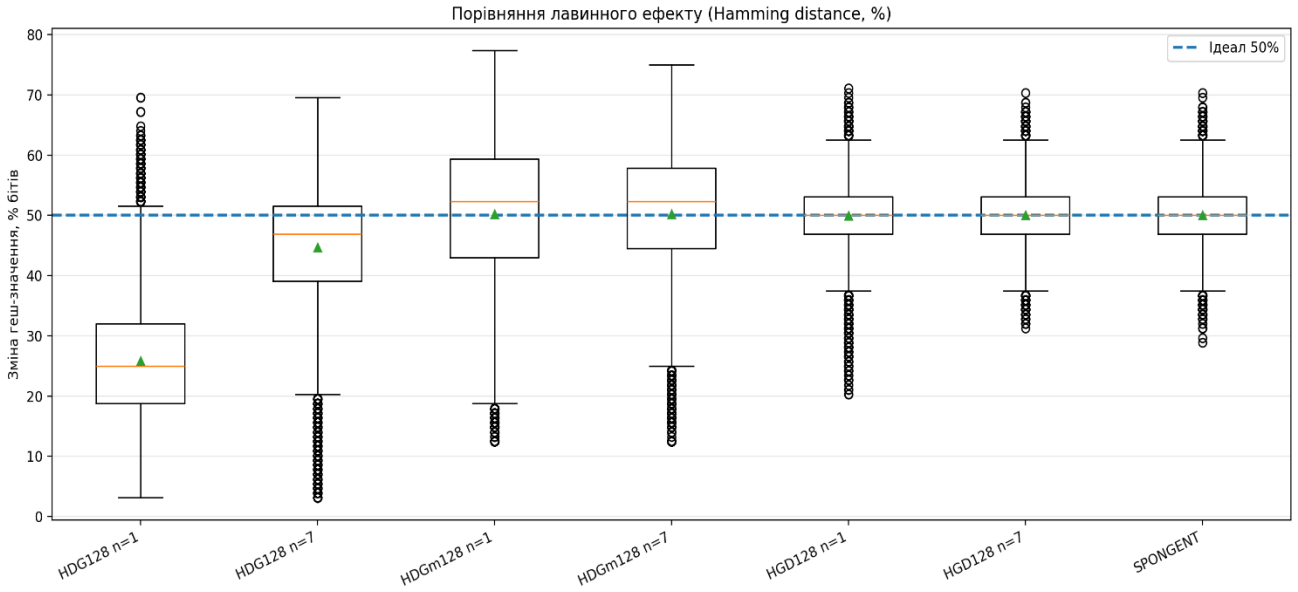

Рисунок 4.8 – Діаграма розмаху значень лавинного ефекту при зміні одного байта вхідного повідомлення

Окремо досліджено лавинний ефект при зміні двох байтів вхідного повідомлення. Експеримент виконано за аналогічних параметрів, однак у цьому випадку модифікації підлягали два сусідні байти.

Відповідно до таблиці 4.9, при модифікації двох байтів вхідного

повідомлення метод HGD продовжує забезпечувати лавинний ефект на рівні 50%. Для методу HDGm середнє значення також є близьким до 50 %, однак середньоквадратичне відхилення залишається більшим, ніж для HGD. Для методу HDG при $n = 1$ середнє значення становить 38,94 %, що є недостатнім. Збільшення кількості ітерацій функції ущільнення до $n = 7$ дозволяє підвищити це значення до 47,86 %, тобто суттєво наблизити його до бажаного рівня 50 %, однак результат все ще значно поступається методу HGD.

Таблиця 4.9 – Результати дослідження лавинного ефекту при модифікації двох байтів вхідного повідомлення

Метод	*n*	Середнє значення, %	Середньоквадратичне відхилення, %	Медіана, %	Мінімум, %	Максимум, %
HDG	1	38.94	8.72	39.06	4.69	71.88
HDGm	1	49.98	10.04	50.78	15.5	78.13
HGD	1	50	4.45	50	27.97	71.09
HDG	7	47.86	6.32	48.44	3.13	69.53
HDGm	7	50	8.57	50.78	14.5	75.78
HGD	7	50	4.41	50	30.47	68.75
SPONGENT	1	49.99	4.42	50	28.13	70.31

Отже, результати дослідження лавинного ефекту показали, що серед розроблених методів найкращі результати забезпечує геш-функція HGD. Для неї при зміні одного біта, одного байта та двох байтів вхідного повідомлення середній відсоток змінених бітів геш-значення є близьким до 50 %. Метод HDGm також забезпечує середній відсоток змінених бітів геш-значення, близький до 50%, однак характеризується більшим значенням середньоквадратичного відхилення. Метод HDG не забезпечує належного лавинного ефекту при зміні одного біта, а при модифікації одного та двох байтів потребує збільшення кількості ітерацій функції ущільнення. Водночас слід зазначити, що для задач контролю цілісності з використанням секретного ключа (див. формулу 3.1)

зловмисник не має доступу до значення ключа K, а отже не може цілеспрямовано підбирати модифікацію повідомлення для збереження геш-значення. У такому сценарії навіть помірний рівень лавинного ефекту забезпечує виявлення випадкових та навмисних змін у повідомленні.

4.4 Порівняльні оцінки швидкості методів малоресурсного гешування

Для дослідження швидкості розроблених методів малоресурсного гешування використано програмну бібліотеку *hash_module*, розроблену в підрозділі 4.1. Керування експериментами, передавання параметрів і збирання результатів виконувались засобами Python, однак безпосереднє обчислення геш-значень і вимірювання часу відбувалось у єдиному модулі на C++, що дозволило усунути вплив інтерпретатора Python на результати вимірювання. Тестування виконувалося на комп’ютері з операційною системою Windows 10 Pro 22H2, процесором Intel(R) Core(TM) i7-10750H 2.60GHz та 32 ГБ оперативної пам’яті.

Параметри експерименту:

– методи: HDG, HDGm, HGD;
– довжина геш-значення: 128, 192, 256 біт;
– довжина повідомлення: 8, 64, 1024, 2048, 10000 байт;
– кількість повторень для кожної конфігурації: 1000;
– характеристика, що оцінюється: швидкість гешування, МБ/с.

Під час тестування визначався лише час безпосереднього обчислення геш-значення. Час ініціалізації об’єкта геш-функції, генерації вхідних повідомлень, передавання параметрів та інших допоміжних операцій до оцінювання не включався. Для кожної конфігурації експеримент повторювався 1000 разів, після чого обчислювалось середнє значення швидкості гешування. Результати вимірювання швидкості гешування, що реалізується на основі розроблених методів наведено в таблиці 4.10.

Аналіз таблиці 4.10 показує, що найвищу швидкість серед розроблених методів забезпечують HDG та HDGm. Для 128-бітних реалізацій їх швидкість становить близько 10,5…11,5 МБ/с та закономірно зменшується зі збільшенням

довжини геш-значення. Для 192-бітних реалізацій швидкість становить близько 7…7,5 МБ/с, а для 256-бітних – близько 5,3…5,5 МБ/с. Метод HGD є суттєво повільнішим за HDG та HDGm. Для однакової довжини геш-значення його швидкість є меншою приблизно у 8 разів, що пояснюється побітною організацією обробки даних у функції ущільнення. Для всіх методів при збільшенні довжини повідомлення від 64 до 10000 байт швидкість гешування змінюється неістотно, тобто для довших повідомлень вона практично стабілізується. Для коротких повідомлень довжиною 8 байт в окремих випадках спостерігаються більші відхилення швидкості, що пояснюється впливом постійних витрат на один виклик функції гешування.

Таблиця 4.10 – Швидкість гешування, МБ/с

Метод	Довжина геш-значення	Довжина повідомлення, байт				
		8	64	1024	2048	10000
HDG	128	10.48	10.6	10.61	10.57	10.56
HDGm	128	11.45	11.51	11.48	11.49	11.46
HGD	128	5.34	5.35	5.34	5.32	5.32
HDG	192	7.02	7.08	7.02	7.02	7.03
HDGm	192	7.55	7.52	7.5	7.47	7.45
HGD	192	0.9	0.9	0.89	0.89	0.89
HDG	256	4.79	5.34	5.28	4.89	5.19
HDGm	256	5.57	5.65	5.61	5.6	5.61
HGD	256	0.67	0.67	0.67	0.67	0.67

Для порівняння розроблених методів з відомими малоресурсними геш-функціями використано реалізації SPONGENT-128, PHOTON-128 та U-QUARK мовою C [110], які було інтегровано до єдиного модуля-обгортки, аналогічного бібліотеці *hash_module*, що забезпечило можливість їх підключення до єдиного застосунку для тестування. Порівняльні оцінки швидкості програмних реалізацій малоресурсних геш-функцій наведено в таблиці 4.11.

Таблиця 4.11 – Оцінки швидкості 128-бітних реалізацій розроблених та відомих малоресурсних геш-функцій, МБ/с

Метод	Довжина повідомлення, байт				
	8	64	1024	2048	10000
HDG	10.48	10.6	10.61	10.57	10.56
HDGm	11.45	11.51	11.48	11.49	11.46
HGD	1.32	1.32	1.31	1.31	1.31
PHOTON	0.03	0.08	0.1	0.1	0.1
SPONGENT	0.01	0.03	0.04	0.04	0.04
U-QUARK	0.03	0.07	0.09	0.09	0.09

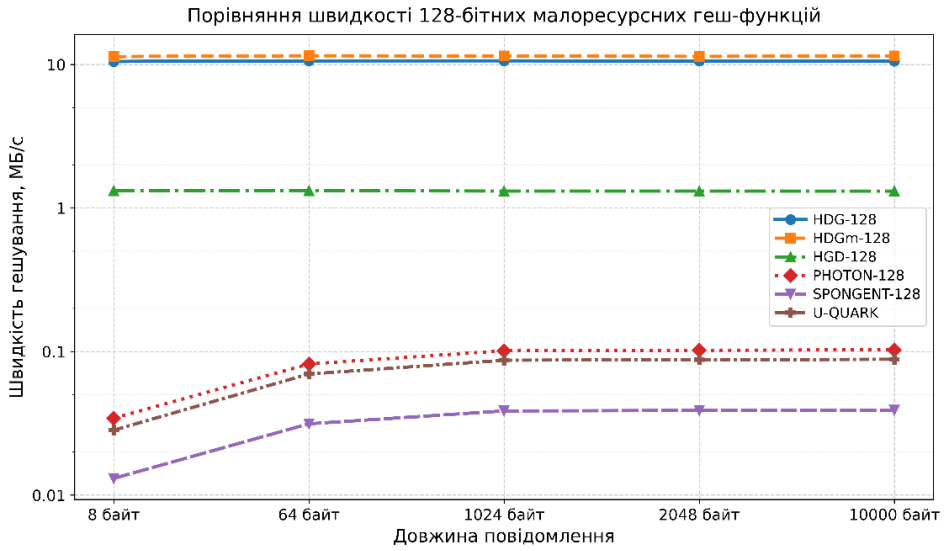

Рисунок 4.9 – Графік залежності швидкості гешування від кількості даних

Аналіз таблиці 4.11 та рисунку 4.9 показує, що розроблені методи суттєво перевищують за швидкістю геш-функції PHOTON, SPONGENT та U-QUARK для всіх досліджених довжин повідомлень. Зокрема, для HDG та HDGm перевага за швидкістю гешування становить 100–286,5 рази. Метод HGD також перевищує розглянуті відомі малоресурсні геш-функції за цим показником у 11–33 рази. Це пояснюється тим, що PHOTON, SPONGENT та U-QUARK орієнтовані насамперед на апаратну реалізацію і використовують складніші побітові перетворення, тоді як розроблені методи мають простішу структуру

внутрішніх перетворень, що вимагає меншої кількості операцій як у програмній, так і в апаратній реалізації.

4.5 Програмна реалізація протоколу автентифікації

Для перевірки коректності роботи протоколу автентифікації RFID-міток з контролем цілісності даних, розробленого у підрозділі 3.2, створено програмну модель мовою Python, що реалізує взаємодію між RFID-міткою та системою. Модель відтворює усі етапи протоколу, включаючи ініціалізацію, штатну сесію автентифікації (кроки 1–3) та механізм відновлення синхронізації (кроки 4–6).

У програмній моделі RFID-мітку представлено класом *RFIDTag*, що зберігає дані M, геш-значення $H(M)$ та одноразові параметри (P, K, r). Систему, до складу якої входить зчитувач та сервер, представлено класом *System*, що містить базу даних зареєстрованих RFID-міток. В якості геш-функції $H(\cdot)$ використано HGD-128 з програмної бібліотеки *hash_module* (див. підрозділ 4.1). Функція оновлення псевдовипадкового числа G(·) реалізована як один крок виконання функції *step* класу об’єкту *LFSR* бібліотеки *hash_module* для демонстрації можливості використання того ж регістра зсуву з лінійним зворотнім зв’язком, що використано в геш-функції HGD.

Для перевірки коректності протоколу та його стійкості до типових атак виконано п’ять сценаріїв тестування.

Сценарій 1: штатна сесія автентифікації. Виконано три послідовні сесії автентифікації між RFID-міткою та системою. У кожній сесії системою формується одноразовий параметр $P_i^{(c)}$ який надсилається RFID-мітці (крок 1), мітка перевіряє автентичність системи, обчислює автентифікаційні дані σ_i та надсилає відповідь $R_i = M||\sigma_i$ (крок 2), після чого система перевіряє автентичність RFID-мітки та цілісність даних (крок 3). Усі три сесії завершились успішно з підтвердженням автентичності та цілісності даних, що свідчить про коректність реалізації основного циклу протоколу та правильність оновлення одноразових параметрів між сесіями.

Сценарій 2: атака повторного відтворення. Під час першої штатної сесії

перехоплено відповідь RFID-мітки $R_i = M||\sigma_i$. У наступній сесії замість відповіді RFID-мітки системі надіслано перехоплену відповідь попередньої сесії. Система відхилила перехоплену відповідь, оскільки перевірка виконувалась із використанням оновленого параметра $K_{i+1}^{(c)}$, для якого обчислене значення σ_{i+1}^{**}, не збігається з перехопленим σ_i.

Сценарій 3: атака модифікації даних. Під час передавання відповіді RFID-мітки зловмисник замінив дані M на модифіковані дані M', зберігши при цьому автентифікаційне значення σ_i. Система відхилила відповідь, оскільки обчислене значення $\sigma_i^{**} = H(H(M')||K_i^{(\text{с})})$ не збіглося з отриманим $\sigma_i = H(H(M)||K_{i+1}^{(\text{м})})$, що підтверджує здатність протоколу виявляти порушення цілісності даних мітки.

Сценарій 4: атака підробки зчитувача. Зловмисник надіслав RFID-мітці випадкове значення замість коректного параметра $P_i^{(\text{с})}$. RFID-мітка обчислила очікуване значення $P_i^{(\text{м})}$, порівняла його з отриманим та відхилила запит без надсилання будь-якої відповіді, що запобігає розкриттю даних мітки підробленому зчитувачу.

Сценарій 5: порушення синхронізації та її відновлення. Після першої штатної сесії змодельовано ситуацію, в якій RFID-мітка обробила запит системи та оновила свої параметри, однак відповідь була заблокована зловмисником і не дійшла до системи. Внаслідок цього параметри RFID-мітки випередили параметри системи на одну сесію. Наступну спробу автентифікації RFID-мітка відхилила через невідповідність отриманого одноразового параметра. Після цього система виконала спробу відновлення синхронізації, що відповідає крокам 4–6 протоколу (див. підрозділ 3.2), обчисливши параметри для наступної сесії. RFID-мітка отримала тимчасовий параметр для наступної сесії та підтвердила автентичність системи, надіслала відповідь після чого автентифікацію було успішно завершено. Подальша звичайна наступна сесія автентифікації також пройшла успішно, що підтвердило повне відновлення синхронізації.

Отримані результати різних сценаріїв підтверджують коректність роботи протоколу у штатному режимі та його здатність протидіяти розглянутим атакам,

що узгоджуються з теоретичним аналізом безпеки протоколу (див. підрозділ 3.3).

4.6 Висновки до розділу 4

1. Програмна реалізація розроблених методів гешування мовою C++ з компіляцією у Python-модуль дозволила провести їх комплексне експериментальне дослідження. Коректність реалізації підтверджено порівнянням результатів обчислення геш-значень з результатами моделювання відповідних спеціалізованих процесорів у середовищі Logisim-evolution.

2. Статистичне тестування за пакетами NIST STS та Dieharder показало, що розроблені методи формують геш-значення з належними статистичними характеристиками. Для методу HDG незадовільні результати окремих тестів при обробці коротких повідомлень усуваються збільшенням кількості ітерацій функції ущільнення.

3. Дослідження лавинного ефекту показало, що метод HGD забезпечує середній відсоток змінених бітів геш-значення на рівні 50 % при зміні одного біта, одного та двох байтів вхідного повідомлення, що практично відповідає результатам геш-функції SPONGENT. Метод HDGm забезпечує середнє значення, близьке до 50 %, однак з більшим середньоквадратичним відхиленням. Метод HDG потребує збільшення кількості ітерацій функції ущільнення для наближення лавинного ефекту до бажаного рівня.

4. Порівняння швидкості програмних реалізацій підтвердило суттєву перевагу розроблених методів над геш-функціями SPONGENT, PHOTON та U-QUARK. Для 128-бітних реалізацій HDG та HDGm перевищують зазначені геш-функції за швидкістю у 100–287 разів, а HGD – у 11–33 рази.

5. Програмна реалізація протоколу автентифікації RFID-міток підтвердила коректність роботи протоколу та його здатність протидіяти атакам повторного відтворення, модифікації даних, підробки зчитувача та порушення синхронізації, що узгоджується з теоретичним аналізом безпеки, наведеним у розділі 3.

Основні наукові результати, представлені в цьому розділі, опубліковано в роботах автора [33, 35, 38, 41].

ВИСНОВКИ

У дисертаційній роботі розв’язано актуальну науково-технічну задачу зменшення апаратних витрат на реалізацію засобів гешування даних для пристроїв Інтернету речей шляхом розробки нових методів малоресурсного гешування та спеціалізованих процесорів для їх апаратної реалізації. Основні наукові та практичні результати роботи є такими:

1. Порівняльний аналіз відомих малоресурсних геш-функцій показав, що існуючі рішення за сукупністю показників апаратної складності, довжини геш-значення та пропускної здатності не повною мірою відповідають потребам найбільш обмежених класів IoT-пристроїв, що підтверджує актуальність розробки нових методів малоресурсного гешування.

2. Вперше запропонований метод малоресурсного гешування HDG має особливість, яка полягає в тому, що формування проміжних геш-значень здійснюється шляхом нагромадження байтів вхідних даних у позиціях, які визначаються псевдовипадковою послідовністю, сформованою генератором. Для апаратної реалізації цього методу використовуються лише набір регістрів з однорідною структурою, восьмирозрядний комбінаційний суматор та регістр зсуву з лінійним зворотним зв’язком, що забезпечує апаратну складність на рівні 915–1680 GE для геш-значень довжиною 128–256 біт. Це на 14–24 % менше порівняно з найближчими за складністю відомими апаратними засобами гешування, а за інтегральним коефіцієнтом апаратної ефективності FoM реалізація HDG перевершує усі відомі засоби у 2,7–4,5 рази.

3. Для покращення криптографічних властивостей методу гешування HDG запропоновано метод HGD, який передбачає формування проміжних геш-значень шляхом нагромадження сформованих генератором псевдовипадкових байтів, яким керують біти вхідного повідомлення, що підлягає гешуванню. Такий підхід забезпечує кращу дифузію (лавинний ефект на рівні 50 %), але вимагає більшої кількості кроків обчислень. Як і для методу HDG, апаратна реалізація цього методу передбачає набір регістрів з однорідною структурою,

восьмирозрядний комбінаційний суматор та регістр зсуву з лінійним зворотним зв’язком. У цьому випадку забезпечується апаратна складність 972–1752 GE для геш-значень довжиною 128–256 біт, що на 8–20 % менше порівняно з найближчими за складністю відомими апаратними засобами гешування.

4. Вперше запропоновані моделі спеціалізованих процесорів для гешування, які містять генератор псевдовипадкової послідовності, набір регістрів та комбінаційний суматор, що забезпечують нагромадження проміжних геш-значень, та блок керування. Такі моделі дозволяють реалізувати розроблені методи в межах єдиної апаратно-орієнтованої архітектури з мінімальними апаратними витратами, що забезпечує уніфікацію реалізацій методів HDG, HDGm та HGD при відмінностях лише у комутаційних елементах та логіці керування. Розроблені у середовищі Logisim-evolution автоматні моделі спеціалізованих процесорів HDG та HGD забезпечили отримання теоретичних оцінок апаратної складності та пропускної здатності.

5. Удосконалено протокол автентифікації RFID-міток, який, на відміну від відомих, поряд із взаємною автентифікацією сторін забезпечує контроль цілісності даних, що зберігаються та передаються RFID-міткою, з використанням лише геш-функції та функції оновлення псевдовипадкового числа як криптографічних примітивів. Аналіз безпеки підтвердив стійкість протоколу до атак підробки зчитувача та мітки, повторного відтворення, активної атаки типу «людина посередині» та порушення синхронізації, а теоретична оцінка апаратної складності підтвердила можливість його реалізації на пасивній RFID-мітці з використанням розробленого HDG-процесора.

6. Експериментальні дослідження розроблених методів з використанням пакетів для статистичного тестування NIST STS та Dieharder підтвердили належні статистичні характеристики формованих геш-значень. Порівняння швидкостей гешування програмними засобами, що реалізують розроблені та відомі методи малоресурсного гешування, показало перевагу розроблених методів над відомими. Для 128-бітних геш-значень методи HDG та HDGm перевершують відомі аналоги за швидкістю у 100–287 разів, а метод HGD – у 11–

33 рази. Це є наслідком того, що запропоновані методи використовують лише нагромаджувальне додавання, тоді як відомі методи гешування передбачають виконання багатораундових перетворень. Програмна реалізація протоколу автентифікації RFID-міток підтвердила коректність його роботи та здатність протистояти основним атакам.

7. Результати наукових досліджень впроваджено у ТОВ «Солар Стар АГРО» у системі ідентифікації на основі RFID-міток, ТОВ «Каскад-Безпека» у системі IoT-моніторингу внутрішніх ресурсів підприємства, а також у навчальний процес кафедри захисту інформації Вінницького національного технічного університету.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Security of Ubiquitous Computing Systems / ред.: G. Avoine, J. Hernandez-Castro. Cham: Springer International Publishing, 2021. DOI: https://doi.org/10.1007/978-3-030-10591-4.
2. Thakor V. A., Razzaque M. A., Khandaker M. R. A. Lightweight Cryptography Algorithms for Resource-Constrained IoT Devices: A Review, Comparison and Research Opportunities. *IEEE Access*. 2021. Т. 9. С. 28177–28193. DOI: https://doi.org/10.1109/access.2021.3052867.
3. Rozlomii I., Yarmilko A., Naumenko S. Innovative resource-saving security strategies for IoT devices. *Journal of Edge Computing*, 4(1), 35–56, 2025. DOI: https://doi.org/10.55056/jec.748.
4. Aumasson J.-P., Henzen L., Meier W., Naya-Plasencia M. QUARK: A lightweight hash. *Cryptographic Hardware and Embedded Systems – CHES 2010*. Cham, Switzerland: Springer, 2010. С. 1–15. DOI: https://doi.org/10.1007/978-3-642-15031-9_1.
5. Aumasson J.-P., Henzen L., Meier W., Naya-Plasencia M. Quark: A lightweight hash. *Journal of Cryptology*. 2012. Т. 26, № 2. С. 313–339. DOI: https://doi.org/10.1007/s00145-012-9125-6.
6. Guo J., Peyrin T., Poschmann A. The photon family of lightweight hash functions. *Advances in Cryptology – CRYPTO 2011*. Berlin, Germany: Springer, 2011. С. 222–239. DOI: https://doi.org/10.1007/978-3-642-22792-9_13.
7. SPN-Hash: Improving the provable resistance against differential collision attacks / J. Choy та ін. *Progress in Cryptology – AFRICACRYPT 2012*. Springer, 2012. С. 270–286. DOI: https://doi.org/10.1007/978-3-642-31410-0_17.
8. Hash functions and RFID tags: Mind the gap / A. Bogdanov та ін. *Cryptographic Hardware and Embedded Systems – CHES 2008*. Berlin, Germany: Springer, 2008. С. 283–299. DOI: https://doi.org/10.1007/978-3-540-85053-3_18.
9. SPONGENT: The Design Space of Lightweight Cryptographic Hashing / A. Bogdanov та ін. *IEEE Transactions on Computers*. 2013. Т. 62, № 10. С. 2041–

2053. DOI: https://doi.org/10.1109/TC.2012.196.

10. PRESENT: An Ultra-Lightweight Block Cipher / A. Bogdanov та ін. *Cryptographic Hardware and Embedded Systems - CHES 2007*. Berlin, Heidelberg. С. 450–466. DOI: https://doi.org/10.1007/978-3-540-74735-2_31.
11. Catalog and Illustrative Examples of Lightweight Cryptographic Primitives / A. Mileva та ін. *Security of Ubiquitous Computing Systems*. Cham, 2021. С. 21–47. DOI: https://doi.org/10.1007/978-3-030-10591-4_2.
12. Windarta S., Suryadi S., Ramli K. Pranggono B. Gunawan T. S. Lightweight Cryptographic Hash Functions: Design Trends, Comparative Study, and Future Directions. *IEEE Access*. 2022. С. 1. DOI: https://doi.org/10.1109/access.2022.3195572.
13. Buchanan W. J., Li S., Asif R. Lightweight cryptography methods. *Journal of Cyber Security Technology*. 2017. Т. 1, № 3-4. С. 187–201. DOI: https://doi.org/10.1080/23742917.2017.1384917.
14. Khan, M., Kozyri, E., Johansen, D., Dagenborg, H. Survey of Lightweight Hardware-Based Hash Functions for Security in Constrained IoT Devices. *IEEE Access*. 2025. С. 1. DOI: https://doi.org/10.1109/access.2025.3611280.
15. Gupta D. N., Kumar R. Sponge based Lightweight Cryptographic Hash Functions for IoT Applications. *2021 International Conference on Intelligent Technologies (CONIT)*, м. Hubli, India, 25–27 черв. 2021 р. 2021. DOI: https://doi.org/10.1109/conit51480.2021.9498572.
16. Manyam M. K. S., Rao K. N. Performance analysis of THF: A novel lightweight cryptography hash function. *Journal of Theoretical and Applied Information Technology*. 2025. Т. 103, № 9. URL: https://www.jatit.org/volumes/Vol103No9/16Vol103No9.pdf (дата звернення: 01.12.2025).
17. Gupta D. N., Kumar R. DeeR-Hash: A lightweight hash construction for Industry 4.0/IoT. *Journal of Scientific & Industrial Research*. 2023. Т. 82, № 1. С. 142–150. DOI: https://doi.org/10.56042/jsir.v82i1.69938.
18. HVH: A lightweight hash function based on dual pseudo-random transformation / Y. Huang та ін. *Security, Privacy, and Anonymity in Computation,*

Communication, and Storage. Cham, Switzerland: Springer, 2021. С. 492–505. DOI: https://doi.org/10.1007/978-3-030-68884-4_41.

19. An AES Based 256-bit Hash Function for Lightweight Applications: Lesamnta-LW / S. Hirose та ін. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*. 2012. Т. E95-A, № 1. С. 89–99. DOI: https://doi.org/10.1587/transfun.E95.A.89.
20. Hirose S. Some Plausible Constructions of Double-Block-Length Hash Functions. *Fast Software Encryption*. Berlin, Heidelberg, 2006. С. 210–225. DOI: https://doi.org/10.1007/11799313_14.
21. Oliynykov R., Gorbenko I., Kazymyrov O., Ruzhentsev V., Kuznetsov O., Gorbenko Y., Boiko A., Dyrda O., Dolgov V., Pushkaryov A. A New Standard of Ukraine: The Kupyna Hash Function [Електронний ресурс]. URL: https://eprint.iacr.org/2015/885 (дата звернення: 17.05.2024).
22. Євсеєв С. П., Йохов О. Ю., Король О. Г. Гешування даних в інформаційних системах : монографія. Харків: Вид. ХНЕУ, 2013. 312 с.
23. Лужецький В., Захарченко С., Кисюк Д. Метод байт-орієнтованого гешування з підвищеним рівнем дифузії та спеціалізований MH-процесор для приватних хмарних інфраструктур. *Вимірювальна та обчислювальна техніка в технологічних процесах*. 2026. № 1. С. 129–136. DOI: https://doi.org/10.31891/2219-9365-2026-85-16.
24. Zabolotnii S., Rozlomii I., Yarmilko A., Naumenko S. Reconfigured CoARX architecture for implementing ARX hashing in microcontrollers of IoT systems with limited resources. *Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska*, 15(4), 164–169, 2025. DOI: https://doi.org/10.35784/iapgos.7782.
25. Баришев Ю. В., Казміревський В. В. Аналіз атак на основі мультиколізій на деревоподібні геш-функції. *Наукові праці ВНТУ*. 2025. № 3. С. 37–46. DOI: https://doi.org/10.31649/2307-5376-2025-3-37-46.
26. Дудник А., Виговський С., Торошанко О., Фесенко А. Метод інтеграції криптопроцесора у роботизовану сенсорну мережу системи розмінування.

Безпека інформаційних систем і технологій, № 2 (10), С. 70–76. 2025. DOI: https://doi.org/10.17721/ISTS.2025.10.70-76.

27. Смірнова Т.В., Якименко Н.М., Смірнов С.А., Поліщук Л.І., Смірнов О.А. Дослідження стійкості до диференціального криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах. *Системи управління, навігації та зв'язку,* № 3(69), 2022. С. 93-98. DOI: https://doi.org/10.26906/SUNZ.2022.3.093.
28. Смірнова Т.В., Якименко Н.М., Смірнов О.А., Поліщук Л.І., Смірнов С.А. Дослідження статистичної стійкості та швидкісних характеристик запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах. *Вісник Хмельницького національного університету. Серія: «Технічні науки»,* № 2 (307). С. 46-52. 2022 DOI: https://www.doi.org/10.31891/2307-5732-2022-307-2-46-52.
29. ISO/IEC 29192-1:2012. Information technology – Security techniques – Lightweight cryptography – Part 1: General. Вид. офіц. 2012. 2012. 18 с.
30. ISO/IEC 29192-5:2016. Information technology – Security techniques – Lightweight cryptography – Part 5: Hash functions. Вид. офіц. 2016. 24 с.
31. Want R. An Introduction to RFID Technology. *IEEE Pervasive Computing*. 2006. Т. 5, № 1. С. 25–33. DOI: https://doi.org/10.1109/mprv.2006.2.
32. Лужецький В. А., Селезньов В. І. Огляд підходів та методів малоресурсного гешування даних. *Вісник Вінницького політехнічного інституту*, 2025. № 6. С. 99–112. DOI: https://doi.org/10.31649/1997-9266-2025-183-6-99-112.
33. Селезньов В. І., Лужецький В. А. Метод малоресурсного гешування типу «Дані – Генератор». *Кібербезпека: освіта, наука, техніка,* 2023. Т. 2, № 22. С. 84–95. DOI: https://doi.org/10.28925/2663-4023.2023.22.8495.
34. Luzhetskyi V., Seleznov V. Hardware implementation of the HDG hash function. *Bulletin of Cherkasy State Technological University*. 2025. Т. 30, №

2, С. 10–21. DOI: https://doi.org/10.62660/bcstu/2.2025.10.

35. Лужецький В., Селезньов В., Хохлачова Ю. Протокол автентифікації засобів інтернету речей з використанням RFID-міток. *Кібербезпека: освіта, наука, техніка*. 2026. Т. 4, № 32. С. 987–1001. DOI: https://doi.org/10.28925/2663-4023.2026.32.1205.
36. Селезньов В. І. Аналіз методів малоресурсного гешування. Матеріали LII науково-технічної конференції підрозділів ВНТУ. Вінниця, 21–23 червня 2023 р. URL: https://ir.lib.vntu.edu.ua/handle/123456789/38623 (дата звернення: 14.04.2026).
37. Селезньов В. І. Програмний засіб для віддаленого статистичного тестування методів малоресурсного гешування за допомогою пакету NIST STS 822. Матеріали LIII науково-технічної конференції підрозділів ВНТУ. Вінниця, 20–22 березня 2024 р. URL: https://ir.lib.vntu.edu.ua//handle/123456789/41890 (дата звернення: 14.04.2026).
38. Селезньов В. І., Лужецький В. А. Удосконалення методу малоресурсного гешування HDG. Матеріали XIII Міжнародної науково-технічної конференції «ITSec-2024 Безпека інформаційних технологій». Львів, 9–11 травня 2024 р. URL: https://sci.ldubgd.edu.ua/bitstream/123456789/14350/1/2024-ITSec_%D0%B7%D0%B1%D1%96%D1%80%D0%BD%D0%B8%D0%BA_v_1.pdf (дата звернення: 14.04.2026).
39. Лужецький В. А., Селезньов В. І. Метод малоресурсного гешування типу «генератор-дані». Матеріали міжнародної науково-практичної конференції «ІТКМ-2024». Івано-Франківськ, 21–24 травня 2024 р. URL: https://journal.comp-sc.if.ua/test/public/journals/zbirnyk_2024.pdf (дата звернення: 14.04.2026).
40. Селезньов В. І. Засіб для гешування даних методом HDG. Матеріали LIV Всеукраїнської науково-технічної конференції підрозділів ВНТУ. Вінниця, 24–27 березня 2025 р. URL: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/48653/24074.pdf (дата звернення: 14.04.2026).
41. Селезньов В. І., Лужецький В. А. Свідоцтво про реєстрацію авторського

права на твір № 127025 «Комп'ютерна програма «Засіб для віддаленого статистичного тестування методів малоресурсного гешування за допомогою пакету NIST STS 822». Державна організація «Український національний офіс інтелектуальної власності та інновацій». Дата реєстрації 31 липня 2024р.

42. S. Sinha, "State of IoT 2025: Number of connected IoT devices growing 14% to 21.1 billion globally," IoT Analytics, 30 жов. 2025. URL: https://iot-analytics.com/number-connected-iot-devices/. (дата звернення: 21.11.2025).
43. Atzori L., Iera A., Morabito G. The Internet of Things: A survey. *Computer Networks*. 2010. Т. 54, № 15. С. 2787–2805. DOI: https://doi.org/10.1016/j.comnet.2010.05.010.
44. Saif S., Das P., Biswas S., Khan S., Haq M. A., Kovtun V. A secure data transmission framework for IoT enabled healthcare. *Heliyon*, Т. 10, № 16, e36269. 2024. DOI: https://doi.org/10.1016/j.heliyon.2024.e36269.
45. Rana M., Mamun Q., Islam R. Lightweight cryptography in IoT networks: A survey. *Future Generation Computer Systems*. 2022. Т. 129. С. 77–89. DOI: https://doi.org/10.1016/j.future.2021.11.011.
46. A Secure Mutual authentication approach to fog computing environment / R. Kalaria та ін. *Computers & Security*. 2021. Т. 111. С. 102-483. DOI: https://doi.org/10.1016/j.cose.2021.102483.
47. Secure Session Key Generation Method for LoRaWAN Servers / K.-L. Tsai та ін. *IEEE Access*. 2020. Т. 8. С. 54631–54640. DOI: https://doi.org/10.1109/access.2020.2978100.
48. A Lightweight Hash Function for Cryptographic and Pseudo-Cryptographic Applications / I. El Hanouti та ін. *Lecture Notes in Electrical Engineering*. Singapore, 2021. С. 495–505. DOI: https://doi.org/10.1007/978-981-33-6893-4_46.
49. Kamel I., Juma H. A Lightweight Data Integrity Scheme for Sensor Networks. *Sensors*. 2011. Т. 11, № 4. С. 4118–4136. DOI: https://doi.org/10.3390/s110404118.

50. Ferguson N., Schneier B. Practical cryptography. Indianapolis : Wiley, 2003. 432 p.
51. Damgård I. B. A Design Principle for Hash Functions. *Advances in Cryptology – CRYPTO' 89 Proceedings*. New York, NY: Springer New York. С. 416–427. DOI: https://doi.org/10.1007/0-387-34805-0_39.
52. Menezes A. J., van Oorschot P. C., Vanstone S. A. Hash Functions and Data Integrity. *Handbook of Applied Cryptography*. 2018. P. 321–383. DOI: https://doi.org/10.1201/9780429466335-9.
53. Rogaway P., Shrimpton T. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. *Fast Software Encryption*. Berlin, Heidelberg, 2004. С. 371–388. DOI: https://doi.org/10.1007/978-3-540-25937-4_24.
54. Investigating the Avalanche Effect of Various Cryptographically Secure Hash Functions and Hash-Based Applications / D. Upadhyay та ін. *IEEE Access*. 2022. Т. 10. С. 112472–112486. DOI: https://doi.org/10.1109/access.2022.3215778.
55. Matyas S. M., Meyer C. H., Oseas J. Generating Strong One-Way Functions with Cryptographic Algorithm. *IBM Technical Disclosure Bulletin*. 1985. Т. 27, № 10A. С. 5658–5659.
56. Hash Functions Based on Block Ciphers: A Synthetic Approach / B. Preneel та ін. *Advances in Cryptology – CRYPTO' 93*. Berlin, Heidelberg: Springer Berlin Heidelberg. С. 368–378. DOI: https://doi.org/10.1007/3-540-48329-2_31.
57. Bertoni G. M., Daemen J., Peeters M., Van Assche G. Sponge functions. *ECRYPT Workshop on Hash Functions*. 2007. URL: https://www.researchgate.net/publication/242285874_Sponge_Functions (дата звернення: 06.05.2024).
58. Lucks S. A Failure-Friendly Design Principle for Hash Functions. *Lecture Notes in Computer Science*. Berlin, Heidelberg, 2005. С. 474–494. DOI: https://doi.org/10.1007/11593447_26.

59. Armadillo: A multi-purpose cryptographic primitive dedicated to hardware / S. Badel та ін. *Cryptographic Hardware and Embedded Systems – CHES 2010*. Berlin, Germany: Springer, 2010. С. 398–412. DOI: https://doi.org/10.1007/978-3-642-15031-9_27.

60. Poschmann A. Y. Lightweight cryptography cryptographic engineering for a pervasive world. 2009. URL: https://eprint.iacr.org/2009/516.pdf (дата звернення: 12.05.2024).

61. Joux A. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. *Advances in Cryptology – CRYPTO 2004*. Berlin, Heidelberg, 2004. С. 306–316. DOI: https://doi.org/10.1007/978-3-540-28628-8_19.

62. Kelsey J., Schneier B. Second Preimages on n-Bit Hash Functions for Much Less than 2 n Work. *Lecture Notes in Computer Science*. Berlin, Heidelberg, 2005. С. 474–490. DOI: https://doi.org/10.1007/11426639_28.

63. Fixed-point attack on Davies–Meyer hash function scheme based on SIMON, SPECK, and SIMECK algorithms / O. J. Permana та ін. *VII International Conference "Safety Problems of Civil Engineering Critical Infrastructures" (SPCECI2021)*. 2023. DOI: https://doi.org/10.1063/5.0119689.

64. Coron J.-S., Patarin J., Seurin Y. The Random Oracle Model and the Ideal Cipher Model Are Equivalent. *Lecture Notes in Computer Science*. Berlin, Germany: Springer, 2008. С. 1–20. DOI: https://doi.org/10.1007/978-3-540-85174-5_1.

65. Keccak / G. Bertoni та ін. *Advances in Cryptology – EUROCRYPT 2013*. Berlin, Heidelberg, 2013. С. 313–314. DOI: https://doi.org/10.1007/978-3-642-38348-9_19.

66. Sepehrdad P., Sušil P., Vaudenay S. Fast Key Recovery Attack on ARMADILLO1 and Variants. *Smart Card Research and Advanced Applications*. Berlin, Heidelberg, 2011. С. 133–150. DOI: https://doi.org/10.1007/978-3-642-27257-8_9.

67. Cryptanalysis of ARMADILLO2 / M. A. Abdelraheem et al. *Lecture Notes in Computer Science*. Berlin, Heidelberg, 2011. P. 308–326.

DOI: https://doi.org/10.1007/978-3-642-25385-0_17.

68. Wang M. Differential Cryptanalysis of Reduced-Round PRESENT. *Progress in Cryptology – AFRICACRYPT 2008*. Berlin, Heidelberg. P. 40–49. DOI: https://doi.org/10.1007/978-3-540-68164-9_4 .

69. Koyama T., Sasaki Y., Kunihiro N. Multi-differential Cryptanalysis on Reduced DM-PRESENT-80: Collisions and Other Differential Properties. *Lecture Notes in Computer Science*. Berlin, Heidelberg, 2013. С. 352–367. DOI: https://doi.org/10.1007/978-3-642-37682-5_25.

70. Cryptanalysis on PHOTON hash function using cube attack / C.-Y. Lu та ін. *2012 International Conference on Information Security and Intelligence Control (ISIC)*, м. Yunlin, Taiwan, 14–16 серп. 2012 р. 2012. DOI: https://doi.org/10.1109/isic.2012.6449760.

71. Improved conditional differential attacks on lightweight hash family QUARK / X. Lu та ін. *Cybersecurity*. 2022. Т. 5, № 1. DOI: https://doi.org/10.1186/s42400-021-00108-3.

72. The GLUON family: A lightweight hash function family based on FCSRs / T. P. Berger та ін. *Progress in Cryptology – AFRICACRYPT 2012*. Springer, 2012. С. 306–323. DOI: https://doi.org/10.1007/978-3-642-31410-0_19.

73. Dar A. B., Lone M. J., Hussain N. Revisiting Lightweight Block Ciphers: Review, Taxonomy and Future Directions. *SSRN Electronic Journal*. 2021. DOI: https://doi.org/10.2139/ssrn.3825816.

74. Hash-One: A lightweight cryptographic hash function / P. M. Mukundan та ін. *IET Information Security*. 2016. Т. 10, № 5. С. 225–231. DOI: https://doi.org/10.1049/iet-ifs.2015.0385.

75. Терещенко Т. О., Тодоренко В. А., Батрак Л. М., Ямненко Ю. С. Мікропроцесорні пристрої: навч. посібник для студентів зі спец-ті «Електроніка». Київ: «Кафедра», 2017. 244 с. ISBN 978-617-7301-37-9.

76. Kohavi Z., Jha N. K. Switching and Finite Automata Theory. 3rd ed. Cambridge: Cambridge University Press, 2009. 652 p.

77. Two New Lightweight Cryptographic Hash Functions Based on Saturnin and

Beetle for the Internet of Things / S. Windarta та ін. *IEEE Access*. 2023. С. 1. DOI: https://doi.org/10.1109/access.2023.3301128.

78. Encryption and hash based security in Internet of Things / B. Vinayaga Sundaram та ін. *2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN)*, м. Chennai, India, 26–28 берез. 2015 р. 2015. DOI: https://doi.org/10.1109/icscn.2015.7219926.

79. IEEE Std 1076-2019. IEEE Standard for VHDL Language Reference Manual. IEEE, 2019. DOI: https://doi.org/10.1109/IEEESTD.2019.8938196.

80. IEEE Std 1800-2023. IEEE Standard for SystemVerilog – Unified Hardware Design, Specification, and Verification Language. IEEE, 2024. DOI: https://doi.org/10.1109/IEEESTD.2024.10458102.

81. Logisim-evolution: digital logic design tool and simulator [Електронний ресурс]. URL: https://github.com/logisim-evolution/logisim-evolution (дата звернення: 25.09.2023).

82. Siemens EDA. ModelSim User's Manual [Електронний ресурс]. Siemens EDA, 2024. URL: https://ww1.microchip.com/downloads/aemDocuments/documents/FPGA/swdocs/modelsim/modelsim_user_2024_2.pdf (дата звернення: 27.01.2025).

83. Dhanda S. S., Singh B., Jindal P. Lightweight Cryptography: A Solution to Secure IoT. *Wireless Personal Communications*. 2020. Т. 112, № 3. С. 1947–1980. DOI: https://doi.org/10.1007/s11277-020-07134-3.

84. Wali H. G., Iyer N. C. Power, Performance and Area Analysis of Sponge Based Lightweight HASH Function. *International Journal of Electrical and Electronic Engineering & Telecommunications*. 2023. С. 245–256. DOI: https://doi.org/10.18178/ijeetc.12.4.245-256.

85. Bormann C., Ersue M., Keranen A. Terminology for Constrained-Node Networks. RFC Editor, 2014. DOI: https://doi.org/10.17487/rfc7228.

86. Peterson W. W., Weldon E. J. Error-Correcting Codes. 2nd ed. Cambridge, MA: MIT Press, 1972. 576 p.

87. Ward R., Molteno T. C. A. Table of Linear Feedback Shift Registers. Electronics

Technical Report No. 2012-1. Dunedin : University of Otago, 2012. 11 p. URL: http://www.physics.otago.ac.nz/reports/electronics/ETR2012-1.pdf (дата звернення: 21.08.2024).

88. Arndt J. 100 “random” degree-32 binary primitive polynomials [Електронний ресурс]. 2005. URL: https://www.jjj.de/mathdata/rand32-primpoly.txt (дата звернення: 17.02.2026).
89. Arndt J. Matters Computational: Ideas, Algorithms, Source Code. Springer, 2016. 980 с. URL: https://www.jjj.de/fxt/fxtbook.pdf (дата звернення: 11.09.2025).
90. Virtual Silicon Inc. 0.18 µm VIP standard cell library tape out ready (part number: UMCL18G212T3). Process: UMC logic 0.18 µm generic II technology. – 2004.
91. Brendan Moran, Milosch Meriac, Hannes Tschofenig, and David Brown. 2021. *A firmware update architecture for internet of things devices.* RFC 9019. RFC Editor. URL: https://www.rfc-editor.org/rfc/rfc9019.txt (дата звернення: 15.04.2025).
92. Regenscheid A. Platform firmware resiliency guidelines. Gaithersburg, MD : National Institute of Standards and Technology, 2018. DOI: https://doi.org/10.6028/nist.sp.800-193.
93. Rozlomii I., Faure E., Yarmilko A., Naumenko S. The method for verifying firmware integrity in IoT devices for secure boot using lightweight hash functions // Proceedings of the Cyber Security and Data Protection (CSDP’2025). Lviv, 2025. URL: https://ceur-ws.org/Vol-4042/paper8.pdf (дата звернення: 15.02.2026).
94. Baashirah R., Abuzneid A. Survey on Prominent RFID Authentication Protocols for Passive Tags. *Sensors*. 2018. Т. 18, № 10. С. 3584. DOI: https://doi.org/10.3390/s18103584.
95. A Lightweight RFID Mutual Authentication Protocol with PUF / F. Zhu та ін. Sensors. 2019. Т. 19, № 13. С. 2957. DOI: https://doi.org/10.3390/s19132957.
96. EPCglobal. EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF

RFID. Specification for RFID Air Interface Protocol for Communications at 860 MHz–960 MHz. Version 2.0.1 [Електронний ресурс]. GS1, 2015. 152 p. URL: https://www.gs1.org/sites/default/files/docs/epc/Gen2_Protocol_Standard.pdf (дата звернення: 13.02.2026).

97. Scott D. A survey of RFID authentication protocols. TechRxiv, 2024. DOI: https://doi.org/10.36227/techrxiv.171216642.23764824/v1.

98. Hung-Yu Chien. SASI: A New Ultralightweight RFID Authentication Protocol Providing Strong Authentication and Strong Integrity. *IEEE Transactions on Dependable and Secure Computing*. 2007. Т. 4, № 4. С. 337–340. DOI: https://doi.org/10.1109/tdsc.2007.70226.

99. Lim T.-L., Li T., Gu T. Secure RFID Identification and Authentication with Triggered Hash Chain Variants. *2008 14th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, м. Melbourne, Australia, 8–10 груд. 2008 р. 2008. DOI: https://doi.org/10.1109/icpads.2008.46.

100. Dass P., Om H. A Secure Authentication Scheme for RFID Systems. *Procedia Computer Science*. 2016. Т. 78. С. 100–106. DOI: https://doi.org/10.1016/j.procs.2016.02.017.

101. Caballero-Gil P., Caballero-Gil C., Molina-Gil J. RFID authentication protocol based on a novel EPC Gen2 PRNG. arXiv, 2022. DOI: https://doi.org/10.48550/arXiv.2208.05345.

102. Alhasan A. Q. A., Rohani M. F., Abuali M. S. Ultra-lightweight mutual authentication protocol to prevent replay attacks for Low-Cost RFID Tags. *IEEE Access*. 2024. С. 1. DOI: https://doi.org/10.1109/access.2024.3386100.

103. RAFI: Robust Authentication Framework for IoT-Based RFID Infrastructure / V. Kumar та ін. *Sensors*. 2022. Т. 22, № 9. С. 3110. DOI: https://doi.org/10.3390/s22093110.

104. Python 3.12.13 documentation [Електронний ресурс]. URL: https://docs.python.org/3.12/ (дата звернення: 15.03.2026).

105. pybind11 Documentation. pybind11 (v3) – Seamless interoperability between C++ and Python [Електронний ресурс] URL: https://pybind11.readthedocs.io/

en/stable/ (дата звернення: 15.03.2026).

106. A statistical test suite for random and pseudorandom number generators for cryptographic applications / L. E. Bassham та ін. Gaithersburg, MD : National Institute of Standards and Technology, 2010. DOI: https://doi.org/10.6028/nist.sp.800-22r1a.

107. R. G. Brown. Dieharder: A random number test suite. URL: https://webhome.phy.duke.edu/~rgb/General/dieharder.php (дата звернення: 25.09.2025).

108. Ylonen T. The Secure Shell (SSH) Protocol Architecture / ред. C. Lonvick. RFC Editor, 2006. DOI: https://doi.org/10.17487/rfc4251.

109. Paramiko: A Python implementation of SSHv2. [Електронний ресурс] URL: https://www.paramiko.org/ (дата звернення: 14.03.2024).

110. HashFunctions: GitHub repository. [Електронний ресурс] URL: https://github.com/ehsanaerabi/HashFunctions (дата звернення: 14.03.2026).

ДОДАТКИ

ДОДАТОК А. Акти впровадженя результатів дисертаційної роботи

АКТ

про впровадження результатів дисертаційної роботи

«Методи та засоби малоресурсного гешування для пристроїв Інтернету речей»

аспіранта Вінницького національного технічного університету

Селезньова Віталія Ігоровича

Результати дисертаційної роботи «Методи та засоби малоресурсного гешування для пристроїв Інтернету речей» аспіранта Вінницького національного технічного університету Селезньова Віталія Ігоровича, що полягають у розробці:

- методу малоресурсного гешування типу «дані-генератор» (HDG), що передбачає формування проміжних геш-значень шляхом нагромадження байтів вхідних даних у позиціях, керованих псевдовипадковою послідовністю генератора;
- методу малоресурсного гешування типу «генератор-дані» (HGD), який передбачає формування проміжних геш-значень шляхом нагромадження псевдовипадкових чисел, що формуються генератором і керуються байтами даних які підлягають гешуванню;
- програмної бібліотеки, що реалізує запропоновані методи малоресурсного гешування,

впроваджені у систему IoT-моніторингу внутрішніх ресурсів ТОВ «Каскад-Безпека» для перевірки цілісності телеметричних даних.

Директор ТОВ «Каскад-Безпека»
Коваль Артем Миколайович
06.04 2026 р.

ЗАТВЕРДЖУЮ
Директор ТОВ «Солар Стар АГРО»
Сергій ШЕСТОПАЛЮК
«19» квітня 2026 р.

АКТ
про впровадження результатів дисертаційної роботи
«Методи та засоби малоресурсного гешування для пристроїв Інтернету речей»
Селезньова Віталія Ігоровича

Цей акт підтверджує, що результати дисертаційної роботи на здобуття наукового ступеня доктора філософії Селезньова В. І. впроваджено у практичну діяльність при створенні та використання IoT-системи ідентифікації на основі RFID-міток. Зокрема впроваджено такі результати дослідження:

- методи малоресурсного гешування HDG та HGD для обчислення геш-значень у пристроях Інтернету речей з обмеженими апаратними ресурсами;
- моделі спеціалізованих процесорів для малоресурсного гешування при побудові засобів контролю цілісності даних у RFID-пристроях та IoT-системах;
- протокол автентифікації RFID-міток, що поряд із взаємною автентифікацією сторін забезпечує контроль цілісності даних, які зберігаються та передаються RFID-міткою.

Впровадження зазначених результатів дозволило підвищити рівень захищеності системи ідентифікації від несанкціонованої модифікації даних та підробки RFID-міток.

Директор ТОВ «Солар Стар АГРО» Сергій ШЕСТОПАЛЮК

ЗАТВЕРДЖУЮ

Проректор з науково-педагогічної роботи
та організації освітнього процесу
Вінницького національного технічного
університету

Олександр ПЕТРОВ

«10» квітня 2026 р.

АКТ

про впровадження результатів дисертаційної роботи
СЕЛЕЗНЬОВА ВІТАЛІЯ ІГОРОВИЧА
за темою: «Методи та засоби малоресурсного гешування для пристроїв Інтернету речей», яка подана на здобуття ступеня доктора філософії з галузі знань 12 «Інформаційні технології»
за спеціальністю 125 «Кібербезпека»

Комісія у складі декана факультету інформаційних технологій та комп'ютерної інженерії, к.пед.н., доцента Кирилащук Світлани Анатоліївни, к.т.н., доцента кафедри захисту інформації Войтович Олесі Петрівни та к.т.н., доцента кафедри захисту інформації Гарнаги Володимира Анатолійовича склала цей акт про те, що на кафедрі захисту інформації Вінницького національного технічного університету у навчальний процес впроваджено такі результати дисертаційної роботи Селезньова В. І.:

- методи малоресурсного гешування HDG та HGD;
- моделі спеціалізованих процесорів для малоресурсного гешування;
- програмний засіб для статистичного тестування методів малоресурсного гешування з використанням пакета NIST STS.

Зазначені результати використовуються під час проведення лекційних, лабораторних і практичних занять з дисциплін «Прикладна криптологія» та «Криптографічний захист інформації», а також при виконанні курсових робіт з цих дисциплін здобувачами вищої освіти.

Декан ФІТКІ, к.пед.н., доцент Світлана КИРИЛАЩУК

к.т.н., доцент Олеся ВОЙТОВИЧ

к.т.н., доцент Володимир ГАРНАГА

ДОДАТОК Б. Результати тестування програмної реалізації протоколу автентифікації RFID-міток

```
======================================================================
  ПРОГРАМНА ДЕМОНСТРАЦІЯ ПРОТОКОЛУ АВТЕНТИФІКАЦІЇ
  RFID-МІТОК З КОНТРОЛЕМ ЦІЛІСНОСТІ ДАНИХ
======================================================================
  Геш-функція: HGD-128
  Довжина параметрів: 128 біт

======================================================================
  СЦЕНАРІЙ 1: Штатна сесія автентифікації
======================================================================
  Дані мітки M: ISBN:978-0471117094;Author:Bruce Schneier;Title:Applied Cryptography;Edition:2nd;Year:1996;
  Довжина M: 91 байт
  H(M): 45e7f43cbc0a21919828690030be677d

--- Сесія 1 ---
  Крок 1: Система надіслала P_i = f967031a87e3ae1d...
  Крок 2: RFID-мітка прийняла запит, надіслала відповідь(M||Sigma): 4953424e3a393738...||abf26d8b9e903916...
  Крок 3: Перевірка системою
  Автентифікація: ПІДТВЕРДЖЕНО
  Цілісність даних: ЦІЛІСНІ
  Деталі: Автентифікацію підтверджено

--- Сесія 2 ---
  Крок 1: Система надіслала P_i = 5a13547a42ab5768...
  Крок 2: RFID-мітка прийняла запит, надіслала відповідь(M||Sigma): 4953424e3a393738...||595ad99e7f800647...
  Крок 3: Перевірка системою
  Автентифікація: ПІДТВЕРДЖЕНО
  Цілісність даних: ЦІЛІСНІ
  Деталі: Автентифікацію підтверджено

--- Сесія 3 ---
  Крок 1: Система надіслала P_i = 1ca6b6c1f2ca8043...
  Крок 2: RFID-мітка прийняла запит, надіслала відповідь(M||Sigma): 4953424e3a393738...||988610d716af2365...
  Крок 3: Перевірка системою
  Автентифікація: ПІДТВЕРДЖЕНО
  Цілісність даних: ЦІЛІСНІ
  Деталі: Автентифікацію підтверджено

  Результат: три послідовні сесії виконано успішно



======================================================================
  СЦЕНАРІЙ 2: Атака повторного відтворення
======================================================================

--- Сесія 1 (штатна, зловмисник перехоплює відповідь) ---
  Сесія 1 виконана успішно: True
  Зловмисник перехопив відповідь: 4953424e3a3937382d30343731313137...

--- Сесія 2 (зловмисник відтворює перехоплену відповідь) ---
  Крок 1: Система надіслала новий P_i = 48d87d143b05dcf8...
  Зловмисник надсилає перехоплену відповідь замість мітки
  Крок 3: Перевірка системою
  Автентифікація: ВІДХИЛЕНО
  Деталі: Автентифікаційні дані не збігаються

  Результат: replay-атаку відхилено, оскільки sigma обчислено
  з оновленим K_i, який відрізняється від перехопленого



======================================================================
  СЦЕНАРІЙ 3: Атака модифікації даних RFID-мітки
======================================================================
  Оригінальні дані M: ISBN:978-0471117094;Author:Bruce Schneier;Title:Applied Cryptography;Edition:2nd;Year:1996;
  Модифіковані дані M': ISBN:978-1111111111;Author:John Snow;Title:Applied Philosophy;Edition:3nd;Year:2222;

--- Сесія (зловмисник підміняє M у відповіді) ---
  RFID-мітка надіслала відповідь з оригінальними даними
  Зловмисник замінив M на M' у відповіді
  Перевірка системою:
  Автентифікація: ВІДХИЛЕНО
  Деталі: Автентифікаційні дані не збігаються

  Результат: модифікацію даних виявлено, оскільки H(M') != H(M)
```

```
======================================================================
  СЦЕНАРІЙ 4: Атака підробки зчитувача
======================================================================

  Зловмисник надсилає підроблений P = fdadea72ccbc9cd1...
  RFID-мітка відхилила запит: P не збігається з очікуваним
  Мітка не надіслала жодної відповіді

  Результат: підробку зчитувача виявлено, мітка не скомпрометована



======================================================================
  СЦЕНАРІЙ 5: Порушення синхронізації та відновлення
======================================================================

--- Сесія 1 ---
  Крок 1: Система надіслала P_i = e651a3a882183301...
  Крок 2: RFID-мітка прийняла запит, надіслала відповідь(M||Sigma): 4953424e3a393738...||bb009f34e1ede4c5...
  Крок 3: Перевірка системою
  Автентифікація: ПІДТВЕРДЖЕНО
  Цілісність даних: ЦІЛІСНІ
  Деталі: Автентифікацію підтверджено

--- Сесія 2 (відповідь мітки блокована зловмисником) ---
  Мітка прийняла запит та оновила параметри
  Зловмисник заблокував відповідь — система не отримала R_i
  Система фіксує таймаут

--- Спроба 1: звичайна автентифікація (має не спрацювати) ---
  Мітка відхилила запит — параметри розсинхронізовані

--- Спроба 2: відновлення синхронізації (кроки 4-6) ---
  Крок 4: Система надіслала P_{i+1} = 3585818f9b6eba32...
  Крок 5: Мітка прийняла запит та надіслала відповідь
  Крок 6: Перевірка системою
  Автентифікація: ПІДТВЕРДЖЕНО
  Цілісність даних: ЦІЛІСНІ
  Деталі: Синхронізацію відновлено

--- Сесія після відновлення (має працювати штатно) ---

--- Сесія 4 ---
  Крок 1: Система надіслала P_i = f544342b9b02f085...
  Крок 2: RFID-мітка прийняла запит, надіслала відповідь(M||Sigma): 4953424e3a393738...||a45859bd9cabfb25...
  Крок 3: Перевірка системою
  Автентифікація: ПІДТВЕРДЖЕНО
  Цілісність даних: ЦІЛІСНІ
  Деталі: Автентифікацію підтверджено

  Результат: синхронізацію відновлено, протокол працює штатно



======================================================================
  ПІДСУМОК ДЕМОНСТРАЦІЇ
======================================================================
  Сценарій 1 (штатна сесія):             УСПІШНО
  Сценарій 2 (replay-атака):             ВІДХИЛЕНО
  Сценарій 3 (модифікація даних):        ВИЯВЛЕНО
  Сценарій 4 (підробка зчитувача):       ВІДХИЛЕНО
  Сценарій 5 (порушення синхронізації):  ВІДНОВЛЕНО
======================================================================
```

ДОДАТОК В. Список публікацій за темою дисертації

1. Селезньов В. І., Лужецький В. А. Метод малоресурсного гешування типу «Дані – Генератор». *Кібербезпека: освіта, наука, техніка,* 2023. Т. 2, № 22. С. 84–95. URL: https://doi.org/10.28925/2663-4023.2023.22.8495.
2. Лужецький В. А., Селезньов В. І. Огляд підходів та методів малоресурсного гешування даних. *Вісник Вінницького політехнічного інституту*, 2025. № 6. С. 99–112. URL: https://doi.org/10.31649/1997-9266-2025-183-6-99-112.
3. Luzhetskyi V., Seleznov V. Hardware implementation of the HDG hash function. *Bulletin of Cherkasy State Technological University*. 2025. Т. 30, № 2, С. 10–21. URL: https://doi.org/10.62660/bcstu/2.2025.10.
4. Лужецький В., Селезньов В., Хохлачова Ю. Протокол автентифікації засобів інтернету речей з використанням RFID-міток. *Кібербезпека: освіта, наука, техніка*. 2026. Т. 4, № 32. С. 987–1001. URL: https://doi.org/10.28925/2663-4023.2026.32.1205.
5. Селезньов В. І. Аналіз методів малоресурсного гешування. Матеріали LII науково-технічної конференції підрозділів ВНТУ. Вінниця, 21–23 червня 2023 р. URL: https://ir.lib.vntu.edu.ua/handle/123456789/38623.
6. Селезньов В. І. Програмний засіб для віддаленого статистичного тестування методів малоресурсного гешування за допомогою пакету NIST STS 822. Матеріали LIII науково-технічної конференції підрозділів ВНТУ. Вінниця, 20–22 березня 2024 р. URL: https://ir.lib.vntu.edu.ua//handle/123456789/41890.
7. Селезньов В. І., Лужецький В. А. Удосконалення методу малоресурсного гешування HDG. Матеріали XIII Міжнародної науково-технічної конференції «ITSec-2024 Безпека інформаційних технологій». Львів, 9–11 травня 2024 р. URL: https://sci.ldubgd.edu.ua/bitstream/123456789/14350/1/2024-ITSec_%D0%B7%D0%B1%D1%96%D1%80%D0%BD%D0%B8%D0%BA_v_1.pdf.
8. Лужецький В. А., Селезньов В. І. Метод малоресурсного гешування типу

«генератор-дані». Матеріали міжнародної науково-практичної конференції «ІТКМ-2024». Івано-Франківськ, 21–24 травня 2024 р. URL: https://journal.comp-sc.if.ua/test/public/journals/zbirnyk_2024.pdf.

9. Селезньов В. І. Засіб для гешування даних методом HDG. Матеріали LIV Всеукраїнської науково-технічної конференції підрозділів ВНТУ. Вінниця, 24–27 березня 2025 р. URL: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/48653/24074.pdf.
10. Селезньов В. І., Лужецький В. А. Свідоцтво про реєстрацію авторського права на твір № 127025 «Комп'ютерна програма «Засіб для віддаленого статистичного тестування методів малоресурсного гешування за допомогою пакету NIST STS 822». Державна організація «Український національний офіс інтелектуальної власності та інновацій». Дата реєстрації 31 липня 2024 р.