Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

Бакалаврська дипломна робота на тему:

«Засіб байторієнтованого гешування даних»

Виконав: студент 2 курсу групи 1БС-21мс
спеціальність 125 – Кібербезпека

_____ Давимока М.В.

Керівник зав. кафедри ЗІ д.т.н., проф.

_____Лужецький В.А.

Рецензент к. т. н., доц. каф. ПЗ

_____ Бабюк Н.П.

Допущено до захисту
Завідувач кафедри ЗІ
д. т. н., професор
_____ Лужецький В.А.
«19» червня 2023 р.

Вінниця - 2023 року

Вінницький національний технічний університет
Факультет Інформаційних технологій та комп'ютерної інженерії
Кафедра Захисту інформації
Рівень вищої освіти І (бакалаврський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність 125 – Кібербезпека
Освітньо-професійна програма – «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ
Зав. кафедри ЗІ д.т.н., проф.
В. А. Лужецький
20 березня 2023 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Давимоці Миколі Вадимовичу

1. Тема роботи: «Засіб для байторієнтованого гешування даних»
   Керівник роботи: Лужецький Володимир Андрійович, д.т.н., проф,
   затверджені наказом ректора ВНТУ від 20 березня 2023 року № 67
2. Строк подання студентом роботи 19.06.2023 р.
3. Вихідні дані до роботи:
   - обсяг даних, що підлягають гешуваню – не більше 2^{64};
   - довжина геш-коду – 256 біт;
   - генератор псевдовипадкових чисел – РЗЛЗЗ;
   - метод гешування – «генератор – дані».
4. Зміст розрахунково-пояснювальної записки: Вступ. Огляд підходів до побудови геш-функцій. Розроблення байторієнтованого методу гешування даних. Розроблення апаратного засобу для гешування даних. Висновки. Список використаних джерел. Додатки.
5. Перелік ілюстративного матеріалу: Алгоритм процесора байторієнтованого гешування даних (плакат А4). Структура процесу генерації псевдовипадкового числа (плакат А4). Структура спеціалізованого процесора байторієнтованого гешування даних (плакат А4). Структура блоку логічних елементів (плакат А4). Структура 8-розрядного комбінаційного паралельного суматора(плакат А4).

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Лужецький В. А., д.т.н., проф., зав кафедри ЗІ	20.03.23	16.06.23
2	Лужецький В. А., д.т.н., проф., зав кафедри ЗІ	20.03.23	16.06.23
3	Лужецький В. А., д.т.н., проф., зав кафедри ЗІ	20.03.23	16.06.23

7. Дата видачі завдання 20 березня 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів Дипломної кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання. Вступ	20.03.23-26.03.23	
2	Аналіз літературних джерел за напрямком бакалаврської кваліфікаційної роботи	27.03.23-.09.04.23	
3	Розробка рішень, моделей, алгоритмів	10.04.23 – 23.04.23	
4	Практична реалізація, моделювання, експериментування, результати	24.04.23-21.05.23	
58	Оформлення пояснювальної записки, підготовка ілюстративного матеріалу	22.05.23- 24.05.23	
6	Попередній захист БКР	25.05.23-31.05.23	
7	Виправлення зауважень, перевірка БКР на плагіат	01.06.23-15.06.23	
8	Представлення БКР до захисту, рецензування	16.06.23-19.06.23	
9	Захист БКР	20.06.23-23.06.23	

Студент _____ (підпис) Давимока М.В.

Керівник роботи _____ (підпис) Лужецький В. А

ЗМІСТ

АНОТАЦІЯ

Давимока М.В. Засіб байторієнтованого гешування даних. Бакалаврська дипломна робота / Давимока Микола Вадимович – Вінниця ВНТУ, 2023 – 60с. Українською мовою. Рисунків – 14, бібліографі – 13.

Бакалаврська дипломна робота присвячена розробці алгоритму та засобу , що його реалізує. У роботі було здійснено аналіз відоміх підходів до будови геш-функцій, вказано на їхні переваги та недоліки та доцільність розроблення власного алгоритму. Розроблено власний алгоритм байторієнтованого гешування даних, що дав змогу зменьшити складність апаратної реалізації.

ABSTRACT

Davymoka M.V. A means of byte-oriented data gating. Bachelor's thesis / Davymoka Mykola Vadymovych - Vinnytsia VNTU, 2023 - 60 p. In Ukrainian. Figures - 14, bibliography - 13.

The bachelor's thesis is devoted to the development of an algorithm and a tool that implements it. The work analyzed the known approaches to the structure of hash functions, pointed out their advantages and disadvantages and the feasibility of developing their own algorithm. An own algorithm for byte-oriented data hashing has been developed, which made it possible to reduce the complexity of hardware implementation.

ВСТУП

Насамперед, у всіх оглядах з малоресурсної криптографії наголошується, що на сьогодні немає загальної теорії розроблення LWC-алгоритмів (можливо, і не буде). Ціла низка авторів пропонує виокремити розроблення надлегких криптографічних алгоритмів (ultra-lightweight algorithms) в окремий напрям криптографії. При цьому посилюється розбіжність між програмно- та апаратно-орієнтованими легковагими криптоалгоритмами. Фундаментальну відмінність у вимогах, що висуваються ресурсними обмеженнями до програмно- і апаратно-орієнтованих легковагих криптоалгоритмів було продемонстровано в роботі [1].

Використання малоресурсних геш-функцій на сьогоднішній час є дуже доцільним, оскільки зростає попит на криптографічні рішення для обмежених пристроїв і систем з обмеженими ресурсами, таких як бездротові сенсорні мережі, розумні картки, IoT-пристрої, пристрої з обмеженою витратою енергії та обчислювальні можливості.

Предметом дослідження є алгоритм та засіб для байторієнтованого гешування даних.

Метою бакалаврської кваліфікаційної роботи є зменьшення апаратних витрат пристрою для гешування шляхом розробки алгоритму байторієнтованого гешування даних та засобу що його реалізує

Для досягнення мети необхідно:

- аналіз відомих геш-функцій малоресурсної криптографії;
- розробити алгоритм байторієнтованого гешування;
- розробити пристрій для байторієнтованого гешування.

Практичну цінність роботи складає алгоритм та пристрій для байторієнтованого гешування.

1 АНАЛІЗ ПІДХОДІВ ДО ПОБУДОВИ ГЕШ – ФУНКЦІЙ LWC

1.1 Вимоги до засобів малоресурсної криптографії

Як уже було зазначено, головною особливістю сучасного етапу розвитку Інтернету є дедалі більша кількість найрізноманітніших інтелектуальних пристроїв, що мають доступ до Інтернету. Через умови їх функціонування, а також жорсткі цінові обмеження, властиві масовому виробництву, властивих масовому виробництву, ці пристрої характеризуються значними обмеженнями на використовувані ресурси пам'яті, обчислювальну потужність, джерела живлення тощо. Звідси випливають обмеження на використовувані технології і технологічні рішення, що пред'являються до засобів малоресурсної криптографії.

Так, наприклад, жорсткі обмеження накладаються на енерговитратність реалізації криптографічних алгоритмів для пасивних інтелектуальних пристроїв таких, як радіочастотні мітки або безконтактні смарт-картки. Відповідно до стандарту ISO/IEC 8 пасивні RFID-мітки повинні мати рівень енергоспоживання не більше 15 μW для того, щоб гарантувати роботу пристрою в радіусі до 1 м.

Останнє, в свою чергу, обмежує можливості, наприклад, у розпаралелюванні обчислень з метою збільшення швидкодії алгоритму.

Інший приклад обмежень дають системи автоматичного здійснення дорожніх зборів (плати за проїзд платними дорогами): для цих систем автомобіль, що рухається з великою швидкістю, має бути ідентифікований (authenticate) зчитувальним пристроєм на значній відстані (10-12 м.) і за вельми нетривалий час (менше 10 мс.). Ясно, що у цьому випадку швидкість роботи значно більш істотні, ніж розміри мікросхеми або її енергоспоживання.

Таким чином, типовими обмеженнями, що зустрічаються в малоресурсній криптографії, є:

– для апаратної реалізації – розмір мікросхеми, споживана енергія, час витрачений на виконання програми;

– для програмної реалізації – розмір програмного коду, розмір оперативної пам'яті, час витрачений на виконання програми.

Можуть з'являтися й інші обмеження. Так, залежно від конкретних умов застосування розроблюваного засобу важливою може виявитися така характеристика, як ширина смуги робочих частот каналу зв'язку.

Кожен проектувальник у галузі малоресурсної криптографії повинен прагнути знайти баланс між безпекою, ціною та продуктивністю (рис. 1.1). Зазвичай легко оптимізувати будь-які дві з трьох цілей розроблення – безпеку і вартість, безпеку і продуктивність, або вартість і продуктивність; однак, дуже важко оптимізувати ці три параметра одночасно. Наприклад, безпечна і високопродуктивна апаратна реалізація може бути досягнута на конвеєрній архітектурі, стійкій до витоку інформації по побічними каналами, що веде до збільшення розміру мікросхеми і відповідно зростання її вартості. З іншого боку, можна спроектувати безпечне, недороге обладнання, однак це призведе до обмеженя продуктивності.

Ефективність реалізації того чи іншого перетворення на програмному або апаратному рівні іншого оцінюється по-різному. Для порівняння програмних реалізацій прийнято розглядати вимоги до пам'яті та час роботи, вимірюваний у тактах процесора.

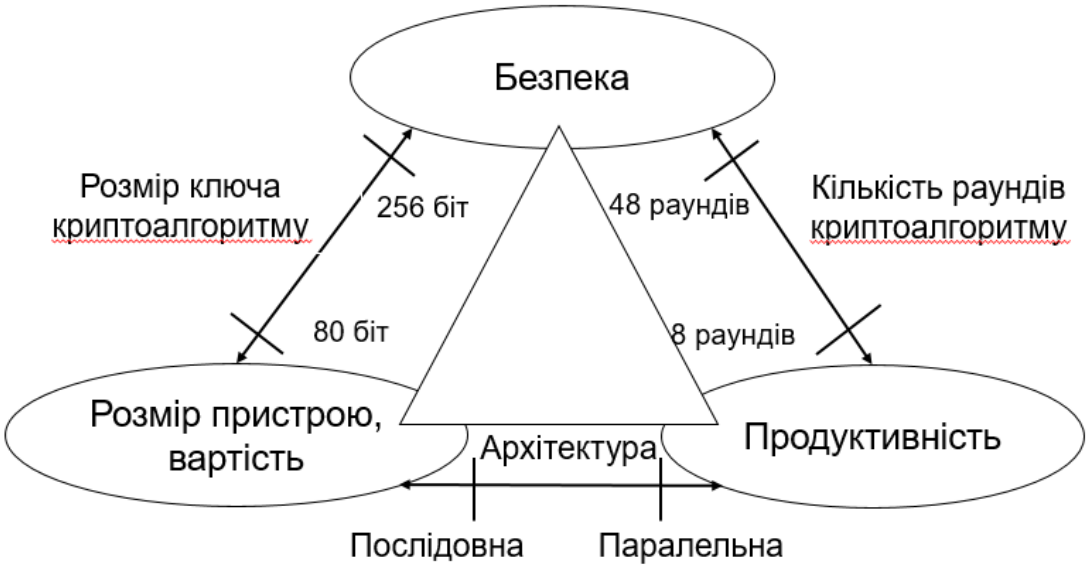


Рисунок 1.1 – Схема оптимізації продуктивності,архітектури і безпеки

Для апаратної реалізації критерієм ефективності є насамперед розмір мікросхеми і час роботи в тактах процесора, хоча для дуже багатьох додатків важливим фактором є енергоспоживання пристрою.

Зазначимо, що багато вимог, які висувають до алгоритмів, призначених для використання в низькоресурсних умовах, було закріплено в межах міжнародного стандарту ISO/ IEC FDIS 29192 – Information technology – Security techniques – Lightweight cryptography.

- Part 1: General
- Part 2: Block ciphers
- Part 3: Stream ciphers
- Part 4: Mechanisms using asymmetric techniques

ISO/IEC 29192 є міжнародним стандартом, що визначає засоби низькоресурсної криптографії для забезпечення секретності, автентичності, ідентифікації, безвідмовності та ключового обміну (data confi dentiality, authentication, identifi cation, non-repudiation, and key exchange).

1.2 Геш-функції на основі блокових шифрів

Існує декілька способів використання блочних шифрів для побудови криптографічних геш-функцій, особливо односторонньої функції стискання. Методи нагадують режими операцій блочних шифрів зазвичай використовні для шифрування.

Стандартний блочний шифр такий як AES можна використати замість цих спеціальних блочних шифрів; це може бути корисним коли вбудована система має забезпечувати шифрування і гешування з мінімальним за розміром кодом або розміром плати. Однак, такий підхід відбувається на дієвості і безпеці. Шифри в геш-функціях створені для гешування: вони використовують великі ключі і блоки, можуть дієво змінювати ключ щоблока, і розроблені і перевірені на стійкість до атак з пов'язаними ключами. Шифри загального призначення мають різні цілі. Зокрема, розміри ключа і блоку в AES роблять нетривіальним використання його для утворення довгих геш-значень, шифрування з AES стає менш дієвим коли ключ

змінюється щоблока і атака з пов'язаними ключами робить його менш безпечним для використання в геш-функціях ніж для шифрування.

1.2.1 Компактні геш-функції з розміром дайджесту 64 біти.

DM-PRESENT це хеш-функція, яка використовує блоковий шифр PRESENT і конструкцію Девіса-Мейєра. Існує два типи хеш-функцій DM-PRESENT: DM-PRESENT-80 і DM-PRESENT-128. Обидва варіанти використовують 64-бітний захист. Розробники стверджують, що хеш-функції забезпечують достатній компроміс між простором і пропускною здатністю [3].

Використання блочного шифру як будівельного блоку при розробці геш-функції [2] є таким же старим, як і DES [3]. Блек та Преніл представили ряд безпечних функцій стиснення від 2n до n-біт, побудованих на основі n-бітового блочного шифру, який вимагає n-бітового ключа. Серед них добре відомі конструкції Девіса-Мейєра, Матіаса-Мейєра-Осеаса та Міягучі-Преніла.

Геш-функція з виходом в n біт може запропонувати лише рівень безпеки 2n операцій для атак на попереднє зображення і друге попереднє зображення та 2n/2 операцій проти пошуку колізій. У той час як рівень безпеки 128 біт є типовим для основних додатків, 80-бітовий рівень безпеки часто є розумною метою для додатків на основі RFID-міток. У будь-якому випадку, існує проблема, оскільки геш-функції, які нам потрібні, не завжди можуть бути негайно сконструйовані з наявних у нас блокових шифрів. Це не нова проблема. Але її також нелегко вирішити, і в побудові 2n-бітових геш-функцій з n-бітового блочного шифру були неоднозначні успіхи [6, 3]. Хоча в багатьох конструкціях були виявлені обмеження, робота Хіросе [7, 8] визначила сімейство геш-функцій довжиною в два блоки, які мають доказ безпеки. Вони використовують блокові шифри з довжиною ключа, яка вдвічі перевищує довжину блоку. Таку властивість мають AES-256 та PRESENT-128.

Коли мова йде про заміну SHA-1, розміри параметрів створюють складну проблему. Якщо ми хочемо використовувати 64-бітний блоковий шифр, такий як PRESENT-128, то для отримання геш-функції з виходом принаймні 160 біт нам потрібна конструкція, яка забезпечує вихід, втричі більший за розмір блоку (таким чином, ми отримуємо 192-бітову геш-функцію). Для цього не існує "класичних"

конструкцій, тому проілюстровано два можливих напрямки проектування. Вони дають репрезентативні конструкції.

Існує багато варіантів побудови 64-бітної геш-функції з 64-бітового блочного шифру. Наприклад режим Девіса-Мейєра, в якому одна 64-бітна ланцюжкова змінна H_i оновлюється за допомогою витягу повідомлення M_i відповідно до обчислення:

$$H_{ij} = E(H_i, M) \oplus H_i$$

Де E – означає шифрування за допомогою PRESENT-80 або PRESENT-128, наведено рис. 1.2. Такі геш-функції будуть корисними лише у додатках, які вимагають односторонньої властивості та 64-бітної безпеки. На кожній ітерації функції стиснення стискаються 64 біти ланцюгової змінної та 80 біт (відповідно 128 біт) вхідних даних, пов'язаних з повідомленням. Таким чином, дві пропозиції DM- PRESENT-80 та DM-PRESENT-128 забезпечують простий компроміс між простором та пропускною здатністю, також надаються дані для послідовної та паралельної реалізації PRESENT.

При використанні методу Дейвіса-Мейєра, важливо зазначити, що ці цифри є гарним показником вартості побудови геш-функції довжиною в один блок. Якщо надавати перевагу застосуванню методів Матіаса-Мейєра-Осіаса або Міягучі-Преніла на основі PRESENT (замість Дейвіса-Мейєра), то вартість DM-PRESENT-80 буде розумним орієнтиром. Відхід від PRESENT до іншого блочного шифру майже напевно призведе до збільшення простору, необхідного для реалізації.

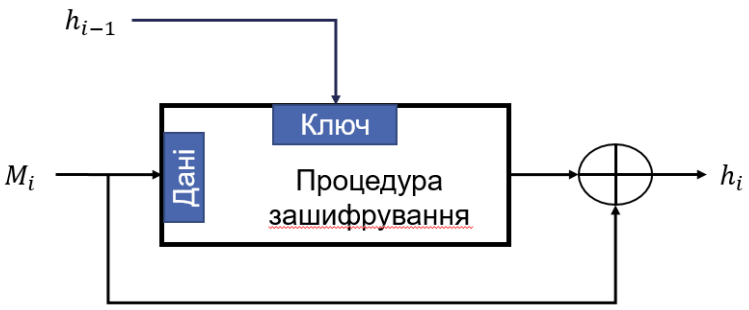


Рисунок 1.2 – Функція стиснення для 64-бітних геш-функцій DM-PRESENT-80 та DM-PRESENT-128.

Архітектура модуля DM-PRESENT-80 з довжиною каналу даних 64 біти складається з 64-розрядного регістра Temp, модуля FSM, 64-розрядного АБО та реалізації PRESENT-80 на основі раундів. PRESENT-80 містить 64-розрядний та 80-розрядний регістр, 64-розрядний АБО, 17 S-боксів (16 у доріжці даних та один у списку ключів), P-рівень та 61-розрядний поворот і 5-розрядний лічильник АБО у списку ключів.

1.2.2 Компактні геш-функції з розміром дайджесту 128 біт

H-PRESENT-128 - це хеш-функція з 128-бітною безпекою. Богданов та ін. розробили H-PRESENT-128, використовуючи конструкцію Хіросе. H-PRESENT-128 є хеш-функцією з подвійною довжиною блоку (DBL). Функція стиснення H-PRESENT-128 використовує дві 64-розрядні ланцюжкові змінні та одне 64-бітне повідомлення (H1, H2, M) і повертає на виході пару оновлених ланцюгових змінних (H1r, H2r).

При проектуванні 128-бітної геш-функції з 64-бітним вихідним блоковим шифром PRESENT, звертаються до так званих конструкцій геш-функцій з подвійною довжиною блоку. Природними кандидатами є конструкції MDC-2 та Хіросе [7, 8]. Ці схеми мають докази стійкості в ідеальній моделі шифрування, де базовий блоковий шифр моделюється як сім'я випадкових перестановок, причому одна перестановка вибирається незалежно для кожного ключа. Однак MDC-2 не є ідеальною конструкцією [9]. Схему H-PRESENT-128 показано на рис. 1.4. Функція стиснення приймає на вхід дві 64-розрядні ланцюжкові змінні та 64-розрядний фрагмент повідомлення, позначений трійкою (H_1 , H_2 , M), і виводить пару оновлених ланцюжкових змінних (H_{1j}, H_{2j}) відповідно до результатів обчислень:

$$H_{1j} = E_{H_2}\|M(H_1) \oplus H_1 \text{ і } H_{2j} = E_{H_2}\|M(H_1 \oplus c) \oplus H_1$$

де E – позначає PRESENT-128, а c – ненульова константа, яку потрібно зафіксувати [6]. Таким чином, ланцюгова змінна H_1 H_2 має довжину 128 біт, а 64 біти вхідних даних, пов'язаних з повідомленням, гешуються за ітерацію.

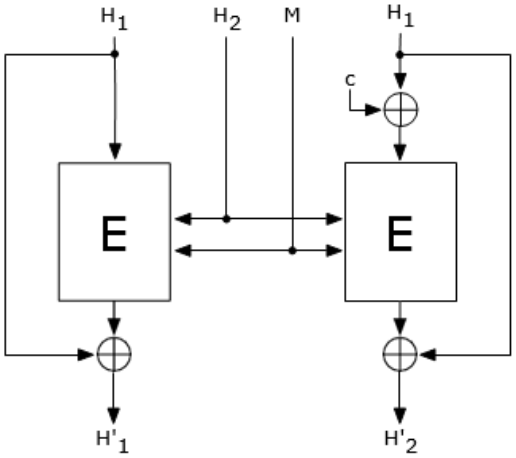


Рисунок 1.3 – Функція стиснення для 128-бітної геш-функції H-PRESENT-128.

Хіросе показав, що в ідеальній моделі шифру атакуючий повинен зробити щонайменше 2n запитів до шифру, щоб знайти колізію з ненульовою перевагою, де n – розмір блоку шифру. Оскільки на кожній ітерації функції стиснення потрібно обчислювати лише один ключовий графік, конструкція Хіросе, ймовірно, є однією з найефективніших конструкцій цього типу, наприклад, у випадку PRESENT таким чином можна заощадити близько 1 000 GE.

Реалізація H-PRESENT-128/128 складається з вентильного регістра з 64-бітним входом і виходом для ланцюгової змінної (384 GE), двох 64-бітних MUX (298 GE), двох 64-бітних вентилів XOR (342 GE), скінченного автомата, модифікованого ядра PRESENT-128 і додаткового шляху даних PRESENT (1 000 GE). Зауважте, що оскільки константа c була обрана як 0x00000000 00000001, то XOR з константою фактично є лише вентилем NOT, який вимагає 0.5 GE.

Хоча компактні геш-функції часто пропонуються в протоколах для RFID-міток, наразі немає достатньо компактних кандидатів. Тут ми розглянули можливість побудови геш-функції на основі блочного шифру, такого як PRESENT. Ми описали геш-функцію які пропонують 64- та 128-бітовий вихід на основі поточних стратегій проектування. За своїми наборами параметрів вони є найбільш компактними кандидатами на геш-функції, доступними на сьогодні. Зокрема, H-PRESENT-128 вимагає близько 4 000 GE, що подібно до найвідомішої реалізації

AES і приблизно на 50% менше, ніж найкраща з відомих реалізацій MD5. У той же час, H-PRESENT-128 вимагає в 20-30 разів менше тактів, ніж компактні реалізації AES і MD5, що дає йому значну перевагу в часі.

Очевидно, що 128-бітові геш-функції актуальні для додатків, де потрібен компроміс між безпекою та продуктивністю. Отримання більших результатів гешування пов'язане з серйозними ускладненнями, більш доцільним буде використання спеціальних конструкцій.

1.3 Геш-функції побудовані з нуля

Геш-функції побудовані з нуля, є криптографічними функціями, які перетворюють вхідні дані будь-якого розміру в фіксований геш-код, також відомий як дайджест або відбиток (digest). [10]

Основний принцип побудови геш-функцій з нуля полягає у використанні різних блокових шифрів, шифрувальних алгоритмів, геш-функцій, логічних операцій та інших криптографічних примітивів для створення надійної та безпечної функції. [10]

1.3.1 Алгоритм SHA-1

SHA-1 (Secure Hash Algorithm 1) є однією з найпоширеніших і широко використовуваних хеш-функцій. Вона була розроблена американським Національним інститутом стандартів і технологій (NIST) і вперше опублікована в 1995 році. Хоча SHA-1 все ще використовується в деяких системах, вона вважається застарілою з точки зору криптографічної безпеки через виявлені уразливості. [12]

Основні характеристики SHA-1:

– Довжина вихідного хешу: SHA-1 генерує хеш-значення фіксованої довжини 160 бітів (20 байтів). Це означає, що будь-яке вхідне повідомлення, незалежно від його розміру, буде перетворено на хеш-значення фіксованого розміру.

– Блочна функція: SHA-1 опрацьовує повідомлення блоками розміром 512 бітів. Якщо вхідне повідомлення не кратне 512 бітам, воно доповнюється до кратності 512 бітів, використовуючи спеціальні правила доповнення.

– Внутрішня структура: SHA-1 базується на конструкції Merkle-Damgård, яка використовує послідовність раундів для обробки блоків повідомлення. Кожен блок пройшовши через цю конструкцію піддається послідовним ітераціям і операціям, таким як логічні операції І/АБО/ВИКЛЮЧНЕ АБО, циклічні зсуви та операції XOR.

– Криптографічна безпека: SHA-1 страждає від криптографічних уразливостей і була важливим об'єктом досліджень і атак. У 2005 році було показано, що SHA-1 має можливість колізій, коли два різних повідомлення можуть мати одне і те ж хеш-значення. Внаслідок цього, використання SHA-1 в криптографічних застосунках і безпеки не рекомендується, особливо для цілей, пов'язаних з безпекою даних та інформації.

– Застосування: SHA-1 була широко використана в різних протоколах, системах безпеки та алгоритмах, таких як SSL/TLS, PGP, SSH, IPsec, а також для генерації підписів і контролю цілісності даних. Однак, у зв'язку зі знайденими уразливостями, рекомендується переходити на більш безпечні алгоритми хешування, такі як SHA-256 або SHA-3.

Важливо зазначити, що SHA-1 вважається застарілим і небезпечним для багатьох сучасних криптографічних застосувань, і рекомендується використовувати більш безпечні алгоритми хешування, які пропонуються сучасними стандартами безпеки. [12]

1.3.2 Алгоритм SHA-256

SHA-256 (Secure Hash Algorithm 256-bit): SHA-256 є частиною сімейства SHA-2 геш-функцій. Вона використовує 256-бітовий геш і включає розширений набір операцій, таких як побітові операції AND, OR, XOR, циклічні зсуви, а також арифметичні операції. [10]

1. Попередня обробка, що полягає у доповненні початкового повідомлення та його розбиття на блоки по 512 біт.

2. Ініціалізація значень гешу. Використовуються константи, що представляють собою перші 32 біта дробових частин квадратних коренів перших 8 простих чисел:(2, 3, 5, 7, 11, 13, 17, 19) наведено на рис. 1.6.

Створення масиву констант k [0…63]. Використовуються ще 64 константи – це перші 32 біта дробових частин кубічних коренів перших 64 простих чисел від 2 до 311.

3. Основний цикл. Наступні кроки будуть виконуватися для кожного 512-бітного блоку вхідних даних. На кожній ітерації циклу буде змінюватися значення h0 – h7: [10]

– Блок ділиться на 16 слів (кожне слово – 32 біти) та записується у масив w [0…15];

– Додається (в кінець масиву) ще 48 слів, ініціалізованих нулями, щоб отримати масив w [0…63];

– Нульові елементи w [16…63] замінюються на нові за алгоритмом;

– Ініціалізація змінних a, b, c, d, e, f, g, h поточними значенням ⁻⁻шу відповідно h0, h1, h2, h3, h4, h5, h6, h7;

– Виконується цикл стиснення, який буде змінювати усі значення від a до h;

– До значень h0…h7 додаються відповідні змінні a…h за модулем 232;

– Фінальний геш є конкатенацією.

1.3.3 Алгоритм Blake2

Blake2 є сімейством Геш-функцій, яке включає різні варіанти для різних застосувань. Найпоширенішим і широко використовуваним варіантом є Blake2b, який оптимізований для 64-бітних платформ і має хорошу швидкодію та безпеку.

Основні характеристики Blake2b:

– Розмір вихідного гешу: Blake2b виробляє геш-значення фіксованого розміру 256 бітів (32 байти), але також може бути налаштований на інші розміри.

– Ключування: Blake2b дозволяє вказати опціональний секретний ключ довжиною до 64 байтів. Якщо використовується ключ, він додає додаткову безпеку до геш-функції.

– Повідомлення з фіксованим розміром блоку: Blake2b обробляє повідомлення порціями фіксованого розміру блоку, рівним 128 байтам. Це полегшує обробку повідомлень різних розмірів.

– Внутрішня структура: Blake2b базується на конструкції Merkle-Damgård, де вхідні блоки повідомлення комбінуються за допомогою компресійної функції, використовуючи різні раунди і перестановки даних.

– Безпека: Blake2b вважається криптографічно стійким, тобто важким для зламу атаками, такими як знаходження колізій або обертання. Він пройшов багато тестів безпеки та аналізів, що підтверджують його надійність.

– Blake2b має простий інтерфейс для використання, який дозволяє генерувати геш-значення з будь-якого вхідного повідомлення. Він є ефективним і широко використовується в різних областях, включаючи блокчейн, криптовалюти, безпеку мережі та інші додатки. [11].

1.3.4 Алгоритм Keccak

Keccak (також відомий як SHA-3) є однією з найбільш відомих геш-функцій, розроблених для забезпечення безпеки та ефективності в малоресурсних системах. Він був обраний Національним інститутом стандартів та технологій США (NIST) як новий стандарт геш-функцій після конкурсу SHA-3, в якому були оцінені різні кандидати.

Основними характеристиками Keccak є його безпечність, швидкодія та простота реалізації. Він заснований на конструкції подвійної гребневої перестановки (sponge construction), яка дозволяє гнучко налаштовувати розмір хеш-функції та потокових шифрів.

Основні принципи, на яких ґрунтується Keccak, включають:

– Підстановка: Використовується нелінійна функція заміни, що змішує біти вхідного блоку даних.

– Перестановка: Використовується лінійна функція перестановки, яка розміщує біти вхідного блоку даних у новому порядку.

Ці дві операції по черзі застосовуються до блоків даних з додатковими операціями зсуву та використанням ключа. Повторюючи цей процес декілька раундів, отримуємо хеш-значення вихідного повідомлення.

Однією з головних переваг Keccak є його спроможність працювати на різних розмірах хеш-функції, включаючи 224, 256, 384 та 512 біт. Це дозволяє використовувати Keccak для різних застосувань, від криптографічного хешування до генерації випадкових чисел.

Keccak також відомий своєю високою швидкодією та ефективністю в різних обчислювальних середовищах. Це робить його особливо підходящим для використання в малоресурсних пристроях, де обмежені обчислювальні ресурси та енергоефективність є важливими факторами.

Окрім того, Keccak має високий рівень безпеки і виявився стійким до різних атак, таких як колізійні атаки та попередні атаки.

Існують різні варіації Keccak, які можуть бути використані для різних цілей. Наприклад, Keccak-p використовується для SHA-3, а Keccak[c] може бути використаний для шифрування.

Keccak є сучасною та надійною геш-функцією, яка підходить для застосування в малоресурсних системах. Його безпечність, швидкодія та простота реалізації роблять його привабливим вибором для різних криптографічних застосувань.

2 РОЗРОБЛЕНЯ БАЙТОРІЄНТОВАНОГО МЕТОДУ ГЕШУВАННЯ ДАНИХ

2.1 Процес байторієнтованого гешування даних

Проаналізувавши підходи до побудови геш-функції LWC та їх вимоги, було розроблено власний алгоритм байторієнтваного гешування даних, що дає змогу зменьшити апаратну складність.

Для прикладу здійсним гешування даних «Гешування_даних », для цього нам необхідно визначити байтове значення кожної літери. Байт – одиниця зберігання і обробки цифрової інформації, що представляє собою сукупність бітів, які система може обробляти одночасно. В кожної літери чи символа є байтове значення в діапазоні від 0 до 255, переглянути необхідну нам частину можна в табл. 2.1.

Таблиця 2.1 – Таблиця кодування Windows-1251(ASCII)

Код	Символ	Код	Символ	Код	Символ	Код	Символ	Код	Символ	Код	Символ	Код	Символ	Код	Символ
128	Ђ	144	ђ	160	Нерозрив-ний пропуск	176	°	192	А	208	Р	224	а	240	р
129	Ѓ	145	‘	161	Ў	177	±	193	Б	209	С	225	б	241	с
130	‚	146	’	162	ў	178	І	194	В	210	Т	226	в	242	т
131	ѓ	147	“	163	J	179	і	195	Г	211	У	227	г	243	у
132	„	148	„	164	¤	180	r	196	Д	212	Ф	228	д	244	ф
133	...	149	•	165	Ґ	181	µ	197	Е	213	Х	229	е	245	х
134	†	150	-	166	¦	182	fj	198	Ж	214	Ц	230	ж	246	ц
135	‡	151	_	167	§	183	·	199	З	215	Ч	231	з	247	ч
136	?	152		168	Ё	184	è	200	И	216	Ш	232	и	248	ш
137	‰	153	™	169	©	185	№	201	Й	217	Щ	233	й	249	щ

Продовження таблиці 2.1

138	**Љ**	**154**	**љ**	**170**	**Є**	**186**	**є**	**202**	**К**	**218**	**Ъ**	**234**	**к**	**250**	**ъ**
139	**<**	**155**	**>**	**171**	**«**	**187**	**»**	**203**	**Л**	**219**	**Ы**	**235**	**л**	**251**	**ы**
140	**Њ**	**156**	****	**172**	**¬**	**188**	**j**	**204**	**М**	**220**	**Ь**	**236**	**м**	**252**	**ь**
141	**Ќ**	**157**	**:**	**173**	**-**	**189**	**S**	**205**	**Н**	**221**	**Э**	**237**	**н**	**253**	**э**
142	**Ћ**	**158**	**ћ**	**174**	**®**	**190**	**s**	**206**	**О**	**222**	**Ю**	**238**	**о**	**254**	**ю**
143	**Џ**	**159**	**џ**	**175**	**Ї**	**191**	**ї**	**207**	**П**	**223**	**Я**	**239**	**п**	**255**	**я**

ASCII (від American Standart Code for Information Interchange – американський стандартний код для обміну інформацією) – таблиця кодування, що фіксує лише першу половину кодів, тобто символи з номерами від 0 (00000000) до (01111111). Таким чином кодують літери латиниці (не розширеної), цифри, розділові знаки, дужки і деякі інші службові символи (наприклад невидимі ознаки кінця рядка).

Відповідно даній таблиці байтові значення літер «Гешування_даних », мають наступний вигляд:

Г – 195;
д – 228;
е – 229;
а – 224;
ш – 248;
н – 237;
у – 243;
и – 232;
в – 226;
х – 245;
а – 224;
н – 237;
н – 237;
я – 255;
_ – 151.

Початкове геш-значення регістрів «0». Відбувається псевдовипадкова генерація чисел для 16 розрядів. Відповідно згенерованим розрядам при значені «0» дані комірки залишаються незмінними, а при значені «1» здійснюється додавання за mod 256 байтового значення літер «Гешування_даних». Після чого

здійснюється зсув вліво утворюючи проміжкове геш-значення до якого буде додаватись байтове значення наступної літери. На виході ми отримаємо геш-значення (бітовий рядок).

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 – початкове геш-значення;

Генерація 16 розрядів.

0 1 1 0 1 0 0 1 1 0 1 0 1 0 0 1

Додавання байтового значення літери «Г – 195».

0 195 195 0 195 0 0 195 195 0 195 0 195 0 0 195

←——

Зсув вліво.

195 195 0 195 0 0 195 195 0 195 0 195 0 0 195 0 – проміжкове геш значення.

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0

Додавання байтового значення літери «е – 229».

195 195 0 195 0 0 195 195 0 195 0 195 0 0 195 0

+

229 0 229 0 229 0 229 0 229 0 229 0 229 0 0 0

Результат:

168 195 229 195 229 0 168 195 229 195 229 195 229 0 195 0

←——

Зсув вліво.

195 229 195 229 0 168 195 229 195 229 195 229 0 195 0 168

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

Додавання байтового значення літери «ш – 248».

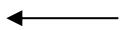
195 229 195 229 0 168 195 229 195 229 195 229 0 195 0 168

+

0 248 0 248 0 248 0 248 0 248 0 248 0 248 0 248

Результат:

195 221 195 221 0 160 195 221 195 221 195 221 0 187 0 160



Зсув вліво.

221 195 221 0 160 195 221 195 221 195 221 0 187 0 160 195

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

0 0 1 0 0 1 0 0 1 0 0 1 1 1 0 0

Додавання байтового значення літери «у – 243»;

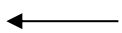
221 195 221 0 160 195 221 195 221 195 221 0 187 0 160 195

+

0 0 243 0 0 243 0 0 243 0 0 243 243 243 0 0

Результат:

221 195 208 0 160 182 221 195 208 195 221 243 174 243 160 195



Зсув вліво.

195 208 0 160 182 221 195 208 195 221 243 174 243 160 195 221

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

1 1 0 1 1 0 1 1 0 0 1 0 1 1 0 0

Додавання байтового значення літери «в – 226».

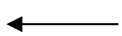
195 208 0 160 182 221 195 208 195 221 243 174 243 160 195 221

+

226 226 0 226 226 0 226 226 0 0 226 0 226 226 0 0

Результат:

165 178 0 130 152 221 165 178 195 221 213 174 213 130 195 221



Зсув вліво.

178 0 130 152 221 165 178 195 221 213 174 213 130 195 221 165

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

0 0 1 0 0 0 1 1 1 1 0 0 0 1 1 1

Додавання байтового значення літери «а – 224».

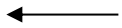
178 0 130 152 221 165 178 195 221 213 174 213 130 195 221 165

+

0 0 224 0 0 0 224 224 224 224 0 0 0 224 224 224

Результат:

178 0 98 152 221 165 146 163 189 181 174 213 130 163 189 133



Зсув вліво.

0 98 152 221 165 146 163 189 181 174 213 130 163 189 133 178

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

1 0 1 0 1 1 1 0 0 1 0 1 0 1 1 1

Додавання байтового значення літери «н – 237»;

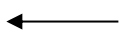
0 98 152 221 165 146 163 189 181 174 213 130 163 189 133 178

+

237 0 237 0 237 237 237 0 0 237 0 237 0 237 237 237

Результат:

237 98 133 221 146 127 144 189 181 155 213 111 163 170 114 159



Зсув вліво.

98 133 221 146 127 144 189 181 155 213 111 163 170 114 159 237

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

0 0 0 0 1 1 1 1 0 0 1 0 1 0 1 0

Додавання байтового значення літери «н – 237».

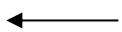
98 133 221 146 127 144 189 181 155 213 111 163 170 114 159 237

+

0 0 0 0 237 237 237 237 0 0 237 0 237 0 237 0

Результат:

98 133 221 146 108 125 170 162 155 213 92 163 151 114 140 237



Зсув вліво.

133 221 146 108 125 170 162 155 213 92 163 151 114 140 237 98

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

1 1 0 1 1 0 1 1 0 0 1 0 0 1 0 1

Додавання байтового значення літери «я – 255».

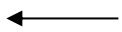
133 221 146 108 125 170 162 155 213 92 163 151 114 140 237 98

+

255 255 0 255 255 0 255 255 0 0 255 0 0 255 0 255

Результат:

132 220 146 107 124 170 161 154 213 92 162 151 114 139 237 97



Зсув вліво.

220 146 107 124 170 161 154 213 92 162 151 114 139 237 97 132

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0

Додавання байтового значення символа «_ – 151».

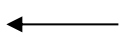
220 146 107 124 170 161 154 213 92 162 151 114 139 237 97 132

+

0 0 0 0 151 0 0 0 0 151 0 0 0 0 151 0

Результат:

220 146 107 124 65 161 154 213 92 57 151 114 139 237 248 132



Зсув вліво.

146 107 124 65 161 154 213 92 57 151 114 139 237 248 132 220

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1

Додавання байтового значення літери «д – 228».

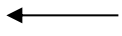
146 107 124 65 161 154 213 92 57 151 114 139 237 248 132 220

+

228 228 228 228 0 228 228 228 228 0 228 228 228 228 0 228

Результат:

118 79 96 37 161 126 185 64 29 151 86 111 209 220 132 192



Зсув вліво.

79 96 37 161 126 185 64 29 151 86 111 209 220 132 192 118

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1

Додавання байтового значення літери «а – 224».

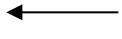
79 96 37 161 126 185 64 29 151 86 111 209 220 132 192 118

+

0 0 0 224 224 224 0 0 0 224 224 224 0 0 0 224

Результат

79 96 37 129 94 153 64 29 151 54 79 177 220 132 192 86



Зсув вліво.

96 37 129 94 153 64 29 151 54 79 177 220 132 192 86 79

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0

Додавання байтового значення літери «н – 237».

96 37 129 94 153 64 29 151 54 79 177 220 132 192 86 79

+

237 237 237 0 0 0 237 237 237 0 0 0 237 237 237 0

Результат:

77 18 111 94 153 64 10 132 35 79 177 220 113 173 67 79

Зсув вліво.

18 111 94 153 64 10 132 35 79 177 220 113 173 67 79 77

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0

Додавання байтового значення літери «и – 232».

18 111 94 153 64 10 132 35 79 177 220 113 173 67 79 77

+

232 232 0 0 232 232 0 0 232 232 0 0 232 232 0 0

Результат

250 87 94 153 40 242 132 35 55 153 220 113 149 43 79 77

Зсув вліво.

87 94 153 40 242 132 35 55 153 220 113 149 43 79 77 250

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1

Додавання байтового значення літери «х – 245».

87 94 153 40 242 132 35 55 153 220 113 149 43 79 77 250

+

0 0 245 245 0 0 245 245 0 0 245 245 0 0 245 245

Результат:

87 94 142 29 242 132 24 44 153 220 102 138 43 79 66 239

Зсув вліво.

94 142 29 242 132 24 44 153 220 102 138 43 79 66 239 87

Отримали вихідне геш-значення(бітовий рядок) для прикладу «Гешування даних».

2.2 Перевірка методу байторієнтованого гешування на колізії

Основним показником ефективності геш-функцій є ймовірність збігу (колізії), тобто ймовірність такої події, коли для різних вхідних даних вихідні геш-значення (бітовий рядок) співпадають Таким чином колізії являються вихідним параметром, щодо оцінки криптостійкості геш-функцій.

Для перевірки на колізії вищенаведеного прикладу ми замінимо вхідне повідомлення «Гешування_даних» на «Додавання_даних» та проведемо гешування по такому ж алгоритму. Після чого проіналізуєм на збіг вихідних геш-значеня для двох майже одинакових вхідних повідомлен.

Д – 196; д – 228;

о – 238; а – 224;

д – 228; н – 237;

а – 224; и – 232;

в – 226; х – 245;

а – 224;

н – 237;

н – 237;

я – 255;

_ – 151.

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

0 1 1 0 1 0 0 1 1 0 1 0 1 0 0 1

Додавання байтового значення літери «Д – 196».

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

+

0 196 196 0 196 0 0 196 196 0 196 0 196 0 0 196

Результат:

0 196 196 0 196 0 0 196 196 0 196 0 196 0 0 196

Зсув вліво.

196 196 0 196 0 0 196 196 0 196 0 196 0 0 196 0

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

1 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0

Додавання байтового значення літери «о – 238».

196 196 0 196 0 0 196 196 0 196 0 196 0 0 196 0

+

238 0 238 0 238 0 238 0 238 0 238 0 238 0 0 0

Результат:

178 196 238 196 238 0 178 196 238 196 238 196 238 0 196 0

Зсув вліво.

196 238 196 238 0 178 196 238 196 238 196 238 0 196 0 178

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

Додавання байтового значення літери «д – 228».

196 238 196 238 0 178 196 238 196 238 196 238 0 196 0 178

+

0 228 0 228 0 228 0 228 0 228 0 228 0 228 0 228

Результат:

196 210 196 210 0 150 196 210 196 210 196 210 0 168 0 150

Зсув вліво.

210 196 210 0 150 196 210 196 210 196 210 0 168 0 150 196

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

0 0 1 0 0 1 0 0 1 0 0 1 1 1 0 0

Додавання байтового значення літери «а – 224».

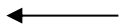
210 196 210 0 150 196 210 196 210 196 210 0 168 0 150 196

+

0 0 224 0 0 224 0 0 224 0 0 224 224 224 0 0

Результат:

210 196 178 0 150 164 210 196 178 196 210 224 136 224 150 196



Зсув вліво.

196 178 0 150 164 210 196 178 196 210 224 136 224 150 196 210

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

1 1 0 1 1 0 1 1 0 0 1 0 1 1 0 0

Додавання байтового значення літери «в – 226».

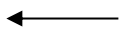
196 178 0 150 164 210 196 178 196 210 224 136 224 150 196 210

+

226 226 0 226 226 0 226 226 0 0 226 0 226 226 0 0

Результат:

166 148 0 120 134 210 166 148 196 210 194 136 194 120 196 210



Зсув вліво.

148 0 120 134 210 166 148 196 210 194 136 194 120 196 210 166

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

0 0 1 0 0 0 1 1 1 1 0 0 0 1 1 1

Додавання байтового значення літери «а – 224».

148 0 120 134 210 166 148 196 210 194 136 194 120 196 210 166

+

0 0 224 0 0 0 224 224 224 224 0 0 0 224 224 224

Результат:

148 0 88 134 210 166 116 164 178 162 136 194 120 164 178 134

Зсув вліво.

0 88 134 210 166 116 164 178 162 136 194 120 164 178 134 148

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

1 0 1 0 1 1 1 0 0 1 0 1 0 1 1 1

Додавання байтового значення літери «н – 237».

0 88 134 210 166 116 164 178 162 136 194 120 164 178 134 148

+

237 0 237 0 237 237 237 0 0 237 0 237 0 237 237 237

Результат:

237 88 115 210 147 97 145 178 162 117 194 101 164 159 115 129

Зсув вліво.

88 115 210 147 97 145 178 162 117 194 101 164 159 115 129 237

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

0 0 0 0 1 1 1 1 0 0 1 0 1 0 1 0

Додавання байтового значення літери «н – 237».

88 115 210 147 97 145 178 162 117 194 101 164 159 115 129 237

+

0 0 0 0 237 237 237 237 0 0 237 0 237 0 237 0

Результат:

88 115 210 147 78 126 159 143 117 194 82 164 140 115 110 237

Зсув вліво.

115 210 147 78 126 159 143 117 194 82 164 140 115 110 237 88

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

1 1 0 1 1 0 1 1 0 0 1 0 0 1 0 1

Додавання байтового значення літери «я – 255».

115 210 147 78 126 159 143 117 194 82 164 140 115 110 237 88

+

255 255 0 255 255 0 255 255 0 0 255 0 0 255 0 255

Результат:

114 209 147 77 125 159 142 116 194 82 163 140 115 109 237 87

Зсув вліво.

209 147 77 125 159 142 116 194 82 163 140 115 109 237 87 114

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0

Додавання байтового значення символа «_» – 151.

209 147 77 125 159 142 116 194 82 163 140 115 109 237 87 114

+

0 0 0 0 151 0 0 0 0 151 0 0 0 0 151 0

Результат:

209 147 77 125 54 142 116 194 82 58 140 115 109 237 238 114

Зсув вліво.

147 77 125 54 142 116 194 82 58 140 115 109 237 238 114 209

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1

Додавання байтового значення літери «д – 228».

147 77 125 54 142 116 194 82 58 140 115 109 237 238 114 209

+

228 228 228 228 0 228 228 228 228 0 228 228 228 228 0 228

Результат:

119 49 97 26 142 88 166 54 30 140 87 81 209 208 114 181

Зсув вліво.

49 97 26 142 88 166 54 30 140 87 81 209 208 114 181 119

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1

Додавання байтового значення літери «а – 224».

49 97 26 142 88 166 54 30 140 87 81 209 208 114 181 119

+

0 0 0 224 224 224 0 0 0 224 224 0 0 0 224

Результат:

49 97 26 110 56 134 54 30 140 55 49 177 208 114 181 87

Зсув вліво.

97 26 110 56 134 54 30 140 55 49 177 208 114 181 87 49

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0

Додавання байтового значення літери «н – 237».

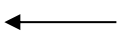
97 26 110 56 134 54 30 140 55 49 177 208 114 181 87 49

+

237 237 237 0 0 0 237 237 237 0 0 0 237 237 237 0

Результат:

78 7 91 56 134 54 11 121 36 49 177 208 95 162 68 49



Зсув вліво.

7 91 56 134 54 11 121 36 49 177 208 95 162 68 49 78

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0

Додавання байтового значення літери «и – 232».

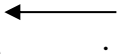
7 91 56 134 54 11 121 36 49 177 208 95 162 68 49 78

+

232 232 0 0 232 232 0 0 232 232 0 0 232 232 0 0

Результат:

239 67 56 134 30 243 121 36 25 153 208 95 138 44 49 78



Зсув вліво.

67 56 134 30 243 121 36 25 153 208 95 138 44 49 78 239

Початкове значення розрядів.

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Генерація 16 розрядів.

0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1

Додавання байтового значення літери «х – 245».

67 56 134 30 243 121 36 25 153 208 95 138 44 49 78 239

+

0 0 245 245 0 0 245 245 0 0 245 245 0 0 245 245

Результат

67 56 123 19 243 121 25 14 153 208 84 127 44 49 67 228

Зсув вліво.

56 123 19 243 121 25 14 153 208 84 127 44 49 67 228 67

Результат байторієнтованого гешування для прикладу «Гешування_даних» має наступний вигляд:

94 142 29 242 132 24 44 **153** 220 102 138 43 79 66 239 87.

Результат байторієнтованого гешування для прикладу «Додавання_даних» має наступний вигляд:

56 123 19 243 121 25 14 **153** 208 84 127 44 49 67 228 67.

Отже замінивши перші 4 літери з «Гешування_даних» на «Додавання_даних» та загешувавши їх за даним принципом, для порівняння на колізії(збіг),можна сказати що результат чудовий. Збіглося лише 1 байтове значення 8 розряду, а саме 153 воно виділене жирним.

2.3 Процес генерації псевдовипадкового числа для 16 розрядів

Одним з популярних методів генерації псевдовипадкових чисел є лінійний зворотній зсувний регістр (Linear Feedback Shift Register, LFSR). Для генерації використовується регістр зсуву з лінійним зворотнім зв'язком з поліномом в 16 степені використовується для генерації псевдовипадкових чисел. Цей регістр складається з 16 бітів (bit_15, bit_14, ..., bit_1, bit_0) і використовує лінійний зворотній зв'язок для зміщення бітів і генерації нового значення.

Ось поліном, який ми використовуєм для регістра зсуву з лінійним зворотнім зв'язком в 16 степені:

$$X^{16} + X^{14} + x^{13} + x^{11} + 1$$

Цей поліном вказує, що біти bit_15, bit_14, bit_13, bit_11 та константа 1 зв'язані між собою завдяки логічного елемента «виключне АБО». Вираз, який відповідає цьому поліному, має вигляд:

bit_0 = bit_15 виключне АБО bit_14 виключне АБО bit_13 виключне АБО bit_11

Логічний елемент "виключне АБО" (XOR) - це бінарна операція, яка приймає два вхідних біти і генерує вихідний біт відповідно до наступних правил:

– якщо вхідні біти однакові (обидва 0 або обидва 1), то вихідний біт буде 0.

– якщо вхідні біти різні (один 0 і один 1), то вихідний біт буде 1.

Символічно логічний елемент "виключне АБО" позначається знаком "⊕" або "+". В таблиці 2.2 наведено істиннісь логічних елементів.

2.2 – Таблиця істинності логічного елемента "виключне АБО"

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

Умовне позначення графічного елемента «виключне АБО» зображено на рис.2.1.

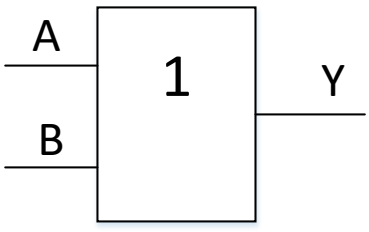


Рисунок 3.1 – Умовне граівчне позначення «виключне АБО»

Тепер розглянемо процес генерації псевдовипадкових чисел з регістром зсуву LFSR з поліномом в 16 степені:

– Ініціалізуємо регістр зсуву початковими значеннями бітів (можуть бути випадковими або заданими певним чином).

– Здійснюємо зсув усіх бітів регістра зліва направо. Значення біту bit_0 замінюється новим значенням, яке обчислюється за допомогою виключне АБО – операцій, використовуючи вказаний поліном і поточні значення інших бітів.

– Виходом генератора є значення біта bit_0.

Цей процес повторюється для кожного нового числа, яке потрібно згенерувати. Псевдовипадкові числа генеруються шляхом послідовного отримання значень з виходу регістра зсуву. Структуру процесу генерації псевдовипадкового числа для 16 розрядів зображена на рис. 2.2.

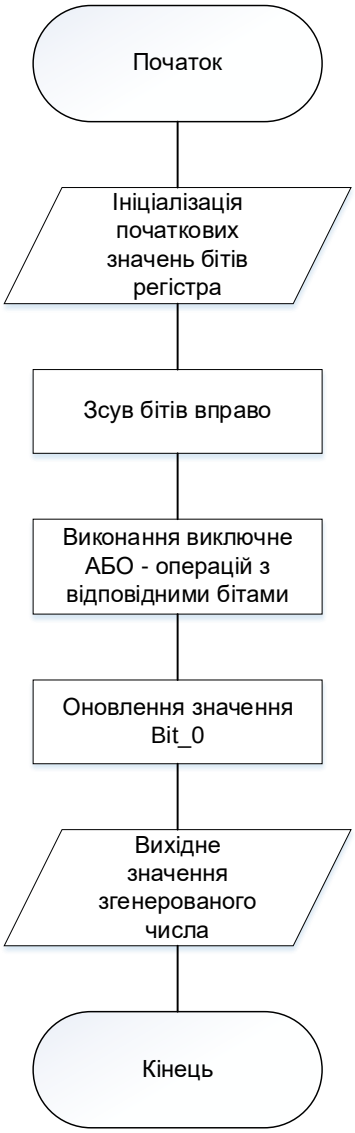


Рисунок 2.2 – Структура процесу генерації псевдовипадкового числа

Також для порівнння можна взяти регістр зсуву з нелінійним зворотнім зв'язком – це цифровий пристрій, який здійснює зсув бітів у внутрішньому регістрі за допомогою керуючих сигналів. У порівнянні зі звичайним регістром зсуву з лінійним зворотнім зв'язком, де зворотній зв'язок виконується лінійно, у регістрі зсуву з нелінійним зворотнім зв'язком використовуються необхідні логічні функції для забезпечення нелінійності зворотного зв'язку.

Основною ідеєю є використання нелінійних логічних функцій, таких як логічний елемент "І", "АБО" або "ВИКЛЮЧНЕ АБО", для створення зворотного зв'язку, який залежить від поточного стану регістра зсуву.

Процес зсуву в регістрі зсуву з нелінійним зворотнім зв'язком може виглядати наступним чином:

– Уведення даних: Початкові дані або біти введені у вхідний регістр.

– Зсув: Керуючі сигнали виконують зсув бітів у внутрішньому регістрі. Це може бути однобітовий або багатобітовий зсув вправо або вліво.

– Нелінійний зворотний зв'язок: За допомогою нелінійних логічних функцій, таких як "І", "АБО" або "ВИКЛЮЧНЕ АБО", виконується обчислення для визначення нового значення біта зворотного зв'язку. Це значення може залежати від певних бітів внутрішнього регістра.

– Зворотній зв'язок: Отримане значення біта зворотного зв'язку використовується для зміни стану регістра зсуву. Це може бути реалізовано шляхом внесення змін до логічних елементів, таких як мультиплексори або комбінаційні логічні схеми, для заміни старих значень бітів на нові значення зворотного зв'язку.

– Виведення даних: Остаточні дані виводяться з вихідного регістра, які можуть бути використані для подальшої обробки або передачі в інші частини системи.

Однак даний РЗНЗЗ потребує більше апаратної реалізації, тому було прийнято рішення взяти РЗЛЗВ.

3 РОЗРОБЛЕННЯ СПЕЦІАЛІЗОВАНОГО ПРОЦЕСОРА ДЛЯ ГЕШУВАННЯ ДАНИХ

3.1 Розроблення алгоритму процесора байторієнтованого гешування даних

Для реалізації спеціалізованого процессора необхідно 19 регістрів, блок керування, блок логічних елементів(8) та суматор за mod 256.Структурна схема зображена на рис. 3.2.

Блок керування є важливою частиною багатьох цифрових систем, таких як мікропроцесори, мікроконтролери, програмовані логічні пристрої та інші. Він відповідає за керування роботою системи та виконання послідовності операцій згідно з заданими інструкціями або програмою.

Основні функції блоку керування включають:

– Декодування інструкцій: Блок керування отримує інструкції з пам'яті та декодує їх для визначення необхідних операцій, які потрібно виконати. Для цього він аналізує біти інструкції, визначає її тип, операції та адреси операндів.

– Управління послідовністю операцій: Блок керування забезпечує виконання операцій в правильній послідовності. Він визначає, яка інструкція повинна бути виконана наступною, переходить до відповідної адреси пам'яті або виконує інші керуючі сигнали, щоб визначити наступну операцію.

– Керування режимами роботи: Блок керування відповідає за вибір режиму роботи системи, такого як режим роботи з пам'яттю, введенням-виведенням, перериваннями тощо. Він встановлює необхідні налаштування, щоб система працювала в потрібному режимі.

– Управління сигналами таймера та годинника: Блок керування може включати годинниковий модуль або таймер для синхронізації роботи системи. Він керує генерацією сигналів тактової частоти, пульсів таймера та інших сигналів, необхідних для синхронного виконання операцій.

– Управління станами та реєстрами: Блок керування відповідає за керування станами системи та реєстрами. Він здійснює запис та зчитування з реєстрів, змінює значення флагів стану, здійснює перехід до певного стану системи.

Регістр зсуву є одним з базових елементів цифрової логіки і використовується для зсуву бітів від одного розряду до іншого. Він складається з послідовності D-тригерів, які забезпечують збереження і переміщення даних.

Давайте опишемо основні характеристики та деталі регістра зсуву:

Кількість тригерів: Регістр зсуву може містити будь-яку кількість тригерів, від одного до декількох десятків, так як в нашому випадку регістри 8-розрядні нам необхідно 8 D-тригерів.

Тригери: Кожен тригер у регістрі зсуву зазвичай є D-тригером, який зберігає бітове значення. Вхід D-тригера приймає вхідні дані, а також вхідний сигнал керування (зазвичай тактовий сигнал), який вказує на момент зсуву даних.

Керування зсувом: Зсув даних в регістрі зсуву відбувається при настанні певної події, зазвичай при зміні тактового сигналу (наприклад, спадаючий фронт або зростаючий фронт). При зміні тактового сигналу, вхідні дані передаються через послідовність тригерів, здійснюючи зсув від одного тригера до наступного.

Режим роботи: Регістр зсуву в працює в послідовному режимі (вхідні дані передаються через тригери послідовно, з одного до наступного).

Зворотній зв'язок: У деяких випадках регістр зсуву може мати зворотній зв'язок, що дозволяє звести до мінімуму зовнішні сигнали для зсуву. Зворотній зв'язок може бути реалізований шляхом підключення виходів тригерів до входів інших тригерів, створюючи циклічну послідовність.

Загалом, регістр зсуву є важливим елементом в цифрових системах і використовується для збереження, зсуву та обробки даних. Його точна реалізація може варіюватись залежно від конкретного дизайну та вимог до системи.

Алгоритм процесора байторієнтованого гешування:

– Спочатку задаються значення нуля регістрам 1-16.

– Після чого за допомогою Рг17(РЗЛЗЗ) відбуваєтсья генерація 16 розрядів, результат записується в Рг18.

– На Рг19 подається вхідне повідомлення, відповідно коли обробляються всі дані процес завершується.

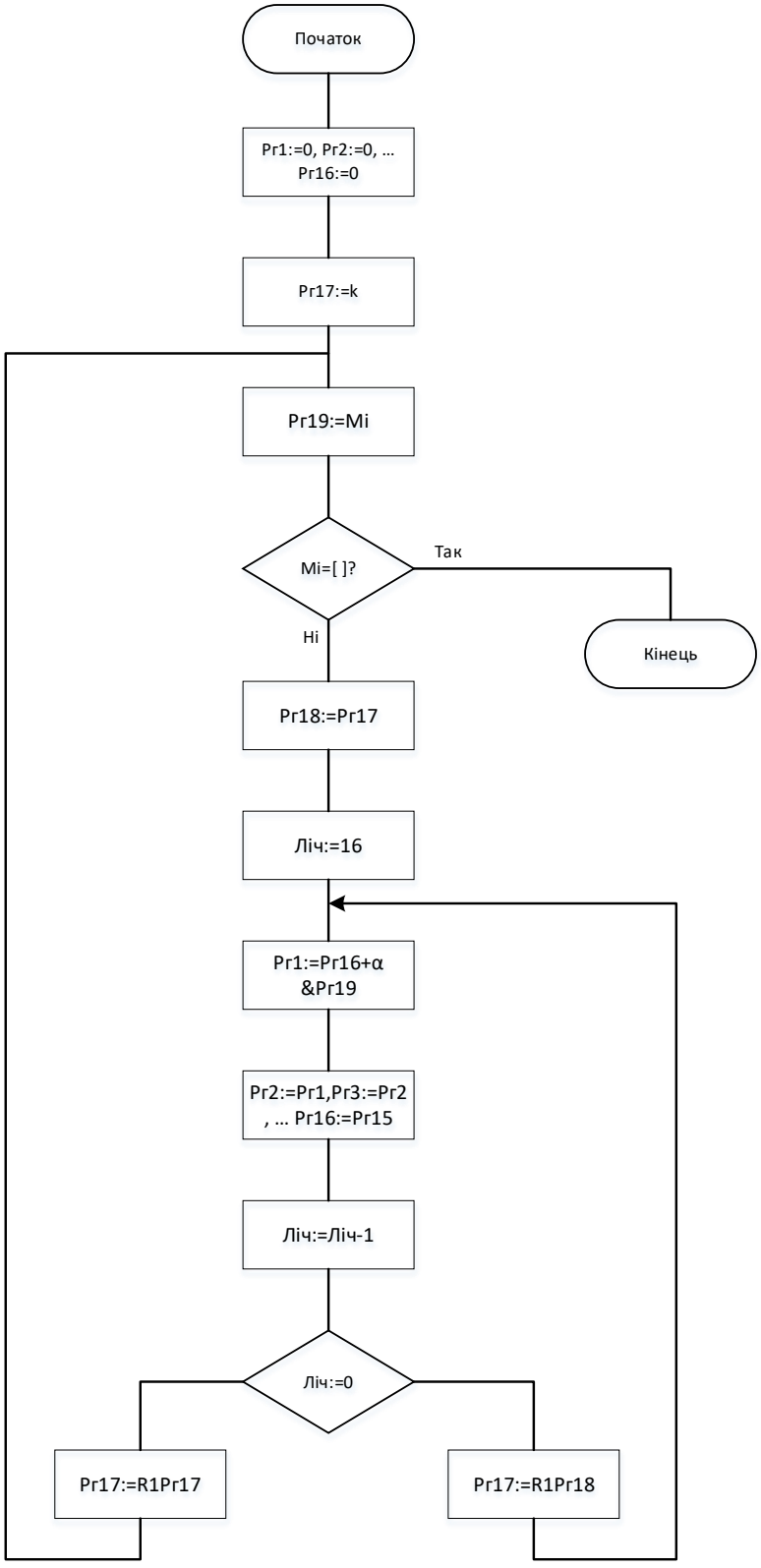


Рисунок 3.1 – Алгоритм процесора байторієнтованого гешування даних

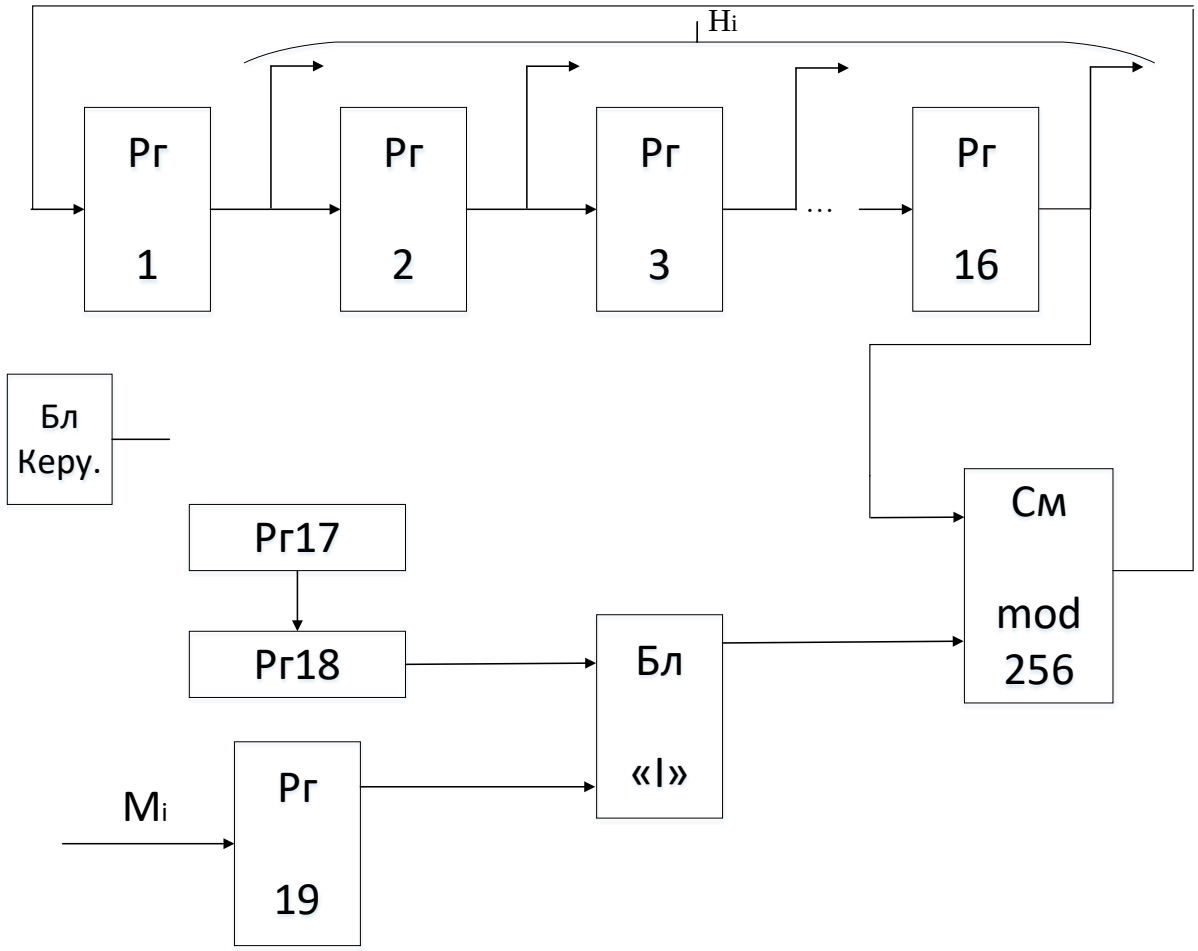


Рисунок 3.2 – Структура процесора байторієнтованого гешування даних

D – тригер (від англійського delay – затримка) має два входи: D – інформаційний та С-тактовий (синхронізуючий): D- тригер синхронний. А це значить, що інформація, яка надходить на вхід D, запам'ятовується лише при надходженні синхронізуючого імпульсу на вхід С, тобто із затримкою на час надходження останнього. Тому D-тригер ще називають тригером затримки. R – це запис значення «0», реалізацію регістрів зсуву 1-16,19 наведено на рис.3.3. Реалізацію Рг18 наведено на рис. 3.4.

Основні характеристики однорозрядного Д-тригера включають:

Вхідний сигнал даних (D): Це вхідний сигнал, який встановлює значення, яке потрібно зберегти в тригері. Він може приймати значення 0 або 1.

Вихідний сигнал (Q): Це вихідний сигнал, який містить збережене значення в тригері. Якщо вхідний сигнал даних (D) змінюється, то вихідний сигнал (Q) також змінюється відповідно.

Інверсний вихідний сигнал (/Q): Це вихідний сигнал, який є інверсією вихідного сигналу (Q). Якщо вихідний сигнал (Q) дорівнює 0, то інверсний вихідний сигнал (/Q) дорівнює 1, і навпаки.

Затримка (Propagation Delay): Це час, який потрібен для затримки вхідного сигналу даних (D) до появи зміни на вихідних сигналах (Q і /Q). Затримка може бути дуже мала, що дозволяє використовувати Д-тригери для синхронізації сигналів в цифрових схемах.

Режим роботи: Однорозрядні Д-тригери можуть працювати в режимі запису (запис значення в тригер), режимі збереження (збереження значення без зміни) або режимі скидання (скидання значення в тригері до 0).

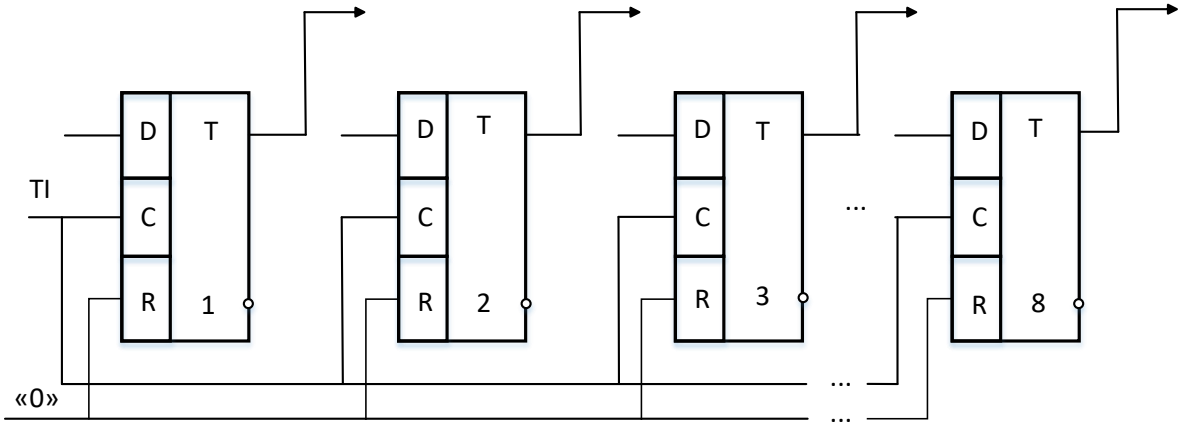


Рисунок 3.3 – Структура Рг1 – Рг16, Рг19

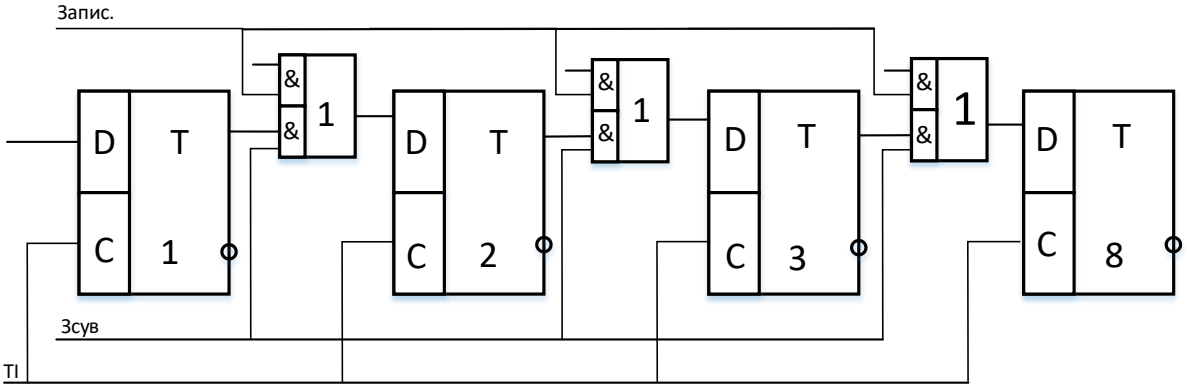


Рисунок 3.4 – Структура Рг18

Блок з восьми логічними елементами "І" (AND) складається з восьми вентилів "І", які працюють паралельно і мають спільний вихід. Цей блок виконує операцію логічного "І" між вхідними сигналами, що означає, що вихід буде активним (1) лише тоді, коли всі вхідні сигнали будуть активними (1).

Схематично блок 8 логічних елементів "І" зображено на рис. 3.5.

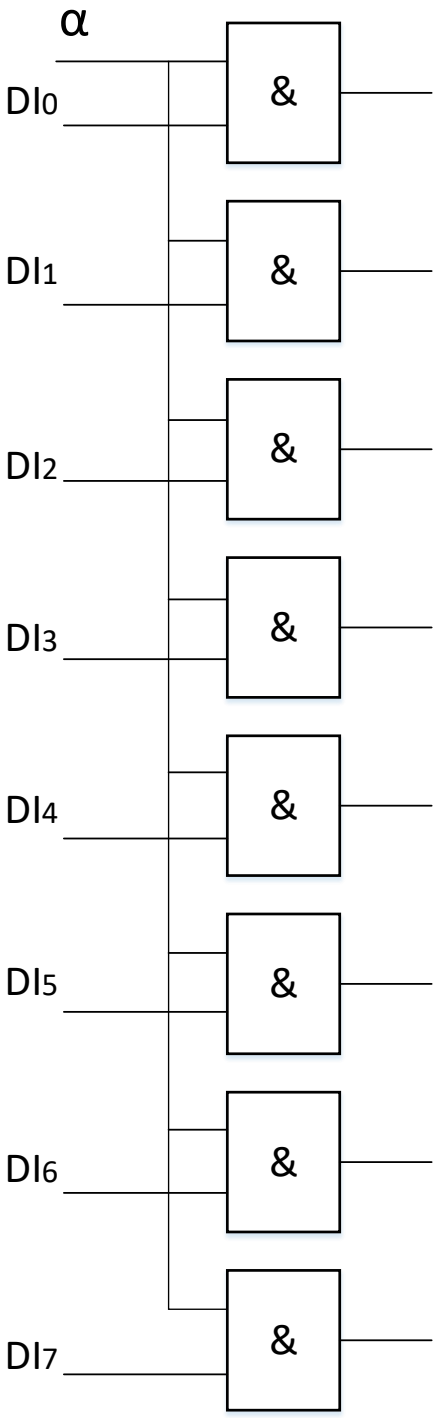


Рисунок 3.5 – Структура блоку логічних елементів

3.2 Реалізація суматора за mod 256

Суматором називається комбінаційний логічний пристрій, призначений для виконання операції арифметичного додавання чисел, представлених у вигляді двійкових кодів. Суматори є одним із основних вузлів арифметико-логічного пристрою. Термін суматор охоплює широкий спектр пристроїв, починаючи з простих логічних схем, до складних цифрових вузлів. Загальним для всіх цих пристроїв є арифметичне додавання чисел, представлених в двійковій формі. [13]

Для вирішеня нашої задачі було обрано 8-розрядний комбінаційний паралельний суматор – логічний пристрій, який використовується для додавання двох 8-бітних чисел. Він має 8 входів для першого числа (A7-A0), 8 входів для другого числа (B7-B0) і 9 виходів для відображення суми (S7-S0) та переносу (C).

Опис роботи 8-розрядного комбінаційного паралельного суматора:

– додавання бітів: кожен біт Ai додається до відповідного біту Bi. Результат додавання зберігається на відповідному виході суматора (Si). Таким чином, Si представляє суму двох бітів (Ai + Bi) з урахуванням можливого переносу від попередніх розрядів.

– перенос: якщо сума двох бітів перевищує 1, генерується перенос (Ci+1). Цей перенос передається до наступного розряду для додавання його до наступних бітів.

– розряд переносу: останній виходний розряд (C8) представляє старший біт переносу. Якщо в результаті додавання останніх бітів виникне перенос, він буде відображений на виході C8.

– використання виходів: результат додавання зберігається на виходах суматора (S7-S0) і може бути використаний для подальшої обробки або відображення. Розряд переносу (C8) також може бути використаний для додавання до наступного розряду при необхідності.

Описаний комбінаційний паралельний суматор є широко використовуваним пристроєм в цифрових логічних схемах та комп'ютерних системах для виконання операції додавання двох 8-бітних чисел. Він дозволяє швидку та ефективну обробку (рис. 3.6) [13]

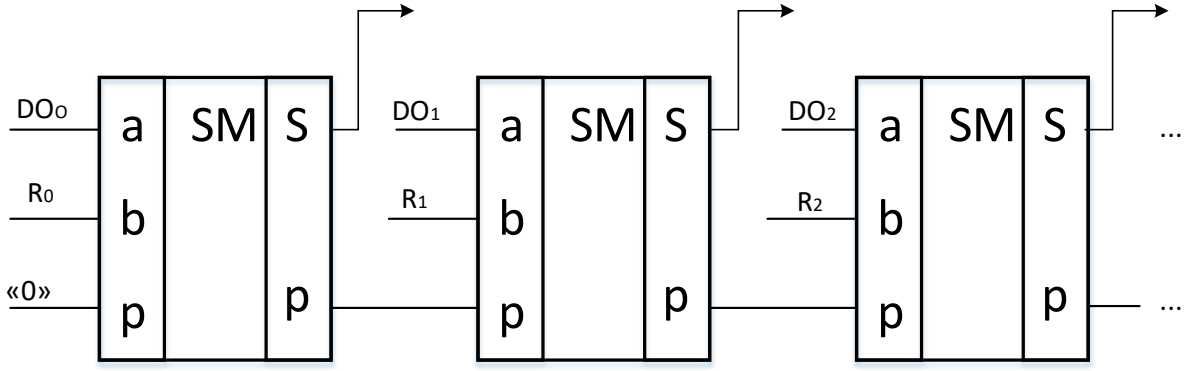


Рисунок 3.6 – Структура 8-розрядного комбінаційного паралельного суматора

Функціонування однорозрядного суматора визначається системою ФАЛ. Технічна реалізація даної ФАЛ може бути виконана на ЛЕ будь-якого типу. Розглянемо, наприклад, побудову однорозрядного суматора з використанням схем двійкових напівсуматорів. Очевидно, що для цієї мети необхідно два напівсуматори і елемент АБО (рис 3.7). Умовне позначення суматора зображено на рис. 3.8.

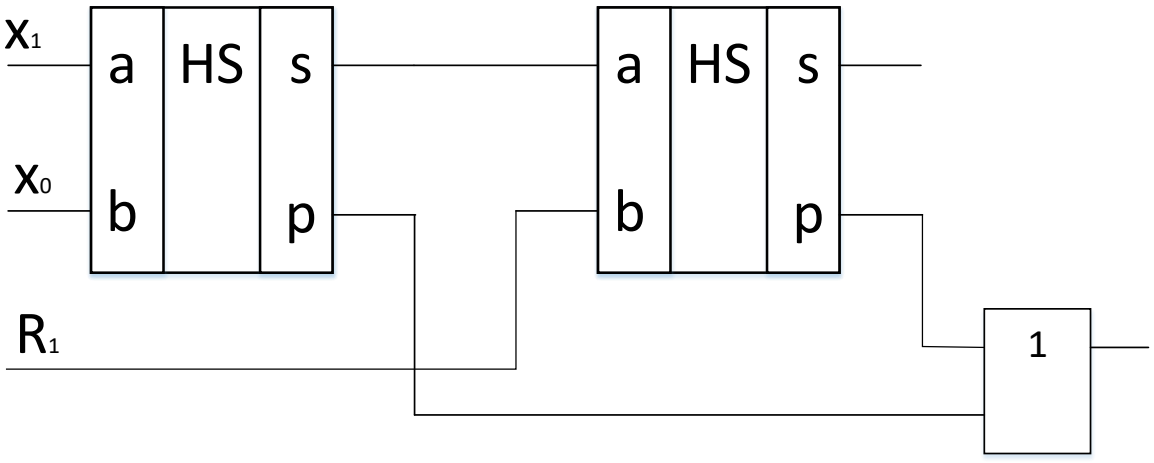


Рисунок 3.7 – Структура однорозрядного суматора

Час підсумовування в наведеній схемі також визначається часом виконання операції «виключне АБО».

Формування сигналу перенесення в старший розряд виконується швидше. Для цього необхідний час.

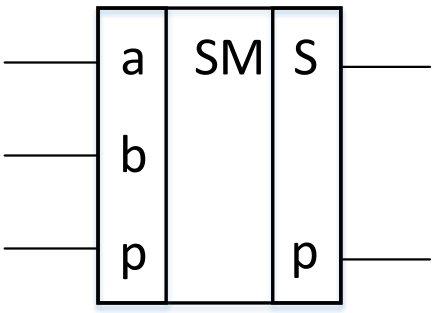


Рисунок 3.8 – Умовне графічне позначення однорозрядного суматора

Напівсуматор – це функціональний вузол з двома входами, на які подаються два однорозрядні числа A і B, та двома виходами: на одному формується результат додавання за модулем два чисел A і B, а на іншому виході – сигнал перенесення у наступний (старший розряд). На умовному зображенні логічна функція напівсуматора позначається буквами HS (від англ. half adder, sum). Структура та його умовне позначення наведено на рис. 3.9.

Найпростішою є реалізація напівсуматора за допомогою двох елементів: «виключального АБО» для отримання суми двох однорозрядних двійкових чисел та логічного елемента 2I для отримання сигналу перенесення.

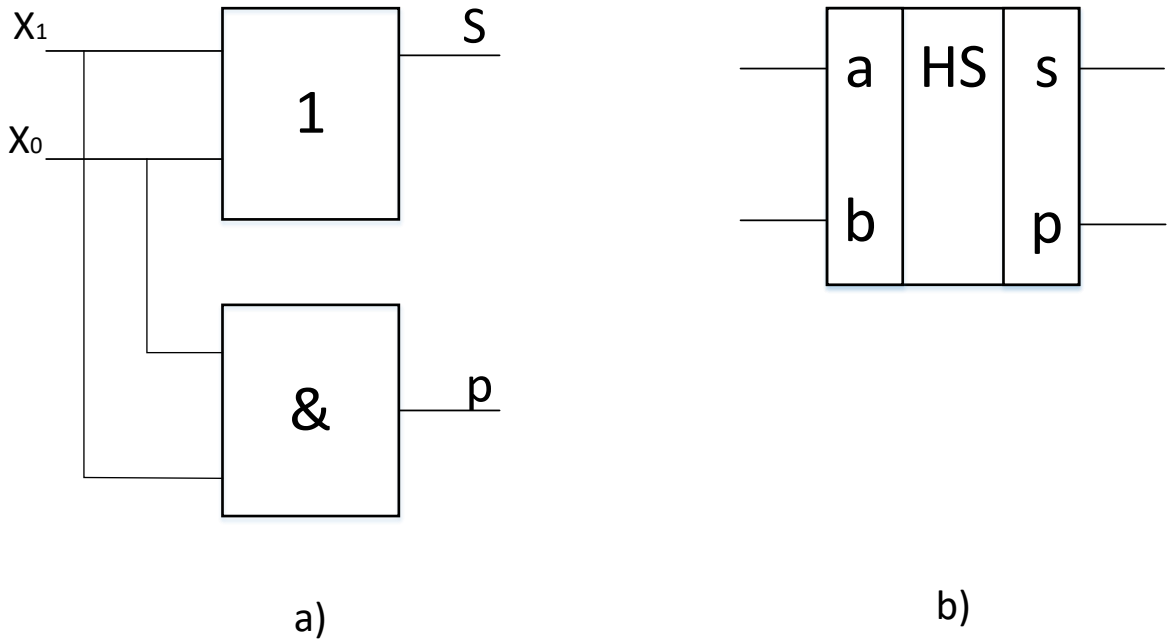


Рисунок 3.8 – Напівсуматор (a) та його умовне позначення (b)

3.3 Оцінка апаратної складності процесора байторієнтованого гешування даних

Оцінка апаратної складності малоресурсного гешування відноситься до визначення обчислювальних вимог для виконання геш-функції на обмежених ресурсах, таких як пристрої з обмеженим обчислювальним потужністю, енергоспоживанням або пам'яттю. Малоресурсне гешування має на меті створення ефективних геш-функцій, які можуть бути реалізовані на таких обмежених пристроях.

Оцінка апаратної складності малоресурсного гешування включає наступні аспекти:

– Обчислювальна складність: Враховується кількість операцій, які необхідно виконати для обчислення геш-функції. Це може охоплювати операції, такі як побітові операції, арифметичні операції, циклічні зсуви, логічні операції та інші.

– Вимоги до пам'яті: Враховується обсяг пам'яті, необхідний для зберігання проміжних даних та таблиць, які використовуються під час обчислення геш-функції. Малоресурсні геш-функції намагаються зменшити вимоги до пам'яті із збереженням адекватної безпеки.

– Енергоефективність: Враховується енергоспоживання, пов'язане з виконанням геш-функції. Малоресурсні геш-функції розробляються з метою зниження енергетичних витрат, що є критичним для пристроїв з обмеженою енергією.

– Оптимізація обчислювальних примітивів: Використання спеціалізованих обчислювальних примітивів, таких як швидкі арифметичні операції, оптимізовані логічні операції та циклічні зсуви, може допомогти покращити апаратну ефективність геш-функцій.

Оцінку апаратної складності процесора байторієнтованого гешування даних, здійснено за допомогою формули (3.1).

$$S = S_T + S_{CM} + S_{\Lambda E}. \quad (3.1)$$

Де: S_T – складність тригерів;

S_{CM} – складність суматора;

$S_{\Lambda E}$ – складність логічних елементів.

Складність логічних елементів наведено в таблиці 3.1.

Для аналізу нашого прикладу необхідна складність таких елементів :

– T – 5,33 у.о.;

– I – 1,33 у. о..

$S_T = (16 \cdot 8 + 16 + 16 + 8) \cdot 5{,}33$ у. о.

$S_{\text{CM}} = (8 \cdot 9) \cdot 1{,}33$ у.о.

$S_{\Lambda\text{E}} = 8 \cdot 1{,}33$ у. о.

$S = 895 + 95 + 10 = 1000$ у.о.

Таблиця 3.1 – Таблиця складності логічних елементів

Standart cell	GE
NOT	0,67
NAND	1
NOR	1
AND	1,33
OR	1,33
MUX	2,33
XOR(2)	2,67
XOR(3)	4,67
D Flip-flop	5,33
Scan D Flip-flop enable	6
Scan D Flip-flop	8,67
Complex Scan D Flip	23,33

Вимогою апаратної реалізації для малоресурсної криптографії являється складність від 100 до 2000 у.о. Ми розробили алгоритм та засіб засіб що його реалізує, зменьшивши апаратну складність майже вдічі ніж відомі алгоритми гешуванняна основі 128 біт H-PRESENT та SQUASH. Приклади алгоритмів зображено в таблиці 3.2.

Таблиця 3.2 – Таблиця апаратної складності алгоритмів гешування

Алгоритм	Nb	GE
AES128-based-DM-AES	28	> 4,400
H-AES		>9,800
Luffa-224/256		10,157 25,833
Luffa-384		13,168 34,401
Luffa-512		16,720 40,715
MAME	256	8,100
MD4	128	7,350
MD5	128	8,400
C-PRESENT	192	8,048 4,600
DM-PRESENT-80	64	1,600 2,213
DM-PRESENT-128	128 64	1,886 2,530
H-PRESENT-128	128	2,330 4,253
QUARK		1,379 2,296
D-QUARK	160	1,702
T-QUARK	224	2,296
U-QUARK	128	1,379
SHA-1	160	6,812 8,120 54,133

Продовження таблиці 3.2

SHA-224/256[3]	256	10,868 11,484 22,025
SHA-384/512		43,330 23,146
SQUASH	32	6,303
	64	6,328
	128	2,646
Cubehash8/1	512	7630

ВИСНОВОК

Під час виконання дипломної роботи було проведено аналіз підходів до побудови геш-функцій малоресурсної криптографії, він показав, що може бути забезпечений використанням відомих блокових шифрів, або шляхом побудови геш-функцій з нуля. Проаналізувавши відоми алгоритми гешування такі як SHA-1, SHA-256, Blake, Keccak, DM-PRESENT та H-PRESENT можна сказати Б що недоліком цих підходів є відносно великі апаратні витрати та час на виконання програми.

В роботі розроблено алгоритм байторієнтованого гешування даних та засіб що його реалізує який має майже двічі меньшу апаратну складність ніж відомі апаратні засоби для гешування.

Таким чином усі сформульовані задачі дослідження розв’язано і досягнута мета дослідження.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Poschmann A. Lightweight Cryptography – Cryptographic Engineering for a Pervasive World. Ph.D. Thesis, Ruhr University Bochum, 2009.

2. D. Coppersmith, S. Pilpel, C.H. Meyer, S.M. Matyas, M.M. Hyden, J. Oseas, B. Brachtl, and M. Schilling. Data Authentication Using Modification Detection Codes Based on a Public One Way Encryption Function. U.S. Patent No. 4,908,861, March 13 1990(49).

3. National Institute of Standards and Technology. FIPS 46-3: Data Encryption Standard (DES). Available via http://csrc.nist.gov, October 1999. (159)

4. Black, P. Rogaway, and T. Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In M. Yung, editor, Advances in Cryptology — CRYPTO 2002, volume 2442 of Lecture Notes in Computer Science, pages 320–335. Springer- Verlag, 2002.(30)

5. B. Preneel. Analysis and Design of Cryptographic Hash Functions. PhD thesis, Katholieke Universiteit Leuven, 1993. (187)

6. L. Brown, J. Pieprzyk, and J. Seberry. LOKI – A Cryptographic Primitive for Authen- tication and Secrecy Applications. In J. Pieprzyk and J. Seberry, editors, Advances in Cryptology — AUSCRYPT 1990, volume 453 of Lecture Notes in Computer Science, pages 229–236. Springer-Verlag, 1990. 38

7. S. Hirose. Provably Secure Double-Block-Length Hash Functions in a Black-Box Model. In C. Park and S. Chee, editors, ICISC 2004, volume 3506, pages 330–342. Springer-Verlag, 2004.

8. S. Hirose. Some Plausible Constructions of Double-Block-Length Hash Functions. In M.J.B. Robshaw, editor, Fast Software Encryption 2006 – FSE 2006, volume 4047 of Lecture Notes in Computer Science, pages 210–225, 2006.

9. P. Steinberger. The Collision Intractability of MDC-2 in the Ideal-Cipher Model. In M. Naor, editor, Advances in Cryptology — EUROCRYPT 2007, volume 4515 of Lecture Notes in Computer Science, pages 34–51. Springer-Verlag, 2007.

10. Щур Н.О. Основи криптології / Н.О. Щур, О.А. Покотило // 2021 – С 88-90.

11. BLAKE2 [Електронний ресурс] – Режим доступу до ресурсу:https://www.blake2.net/.

12. U.S. Department of Commerce, National Institute of Standards and Technology. Secure Hash Standard: Federal Information Processing Standards Publication 180-4. CreateSpace Independent Publishing Platform, pages 10 – 20, 2016.

13. Вузли обчислювальних пристроїв [Електронний ресурс] – Режим доступу до ресурсу:https://itcj.sethost.net/pdf/epivt_2_l_29.pdf.

54

Додаток А

ПРОТОКОЛ ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Засіб байторієнтованого гешування даних
Автор роботи: Давимока Микола Вадимович
Тип роботи: бакалаврська дипломна робота
Підрозділ кафедра захисту інформації ФІТКІ

Показники звіту подібності Unicheck

Оригінальність – 77,8%. Схожість – 22,2%.

Аналіз звіту подібності (відмітити потрібне):

√ 1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

☐ 2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

☐ 3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ (підпис) Каплун В. А. (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ (підпис) Давимока М.В. (прізвище, ініціали)

Керівник роботи _____ (підпис) Лужецький В.А. (прізвище, ініціали)

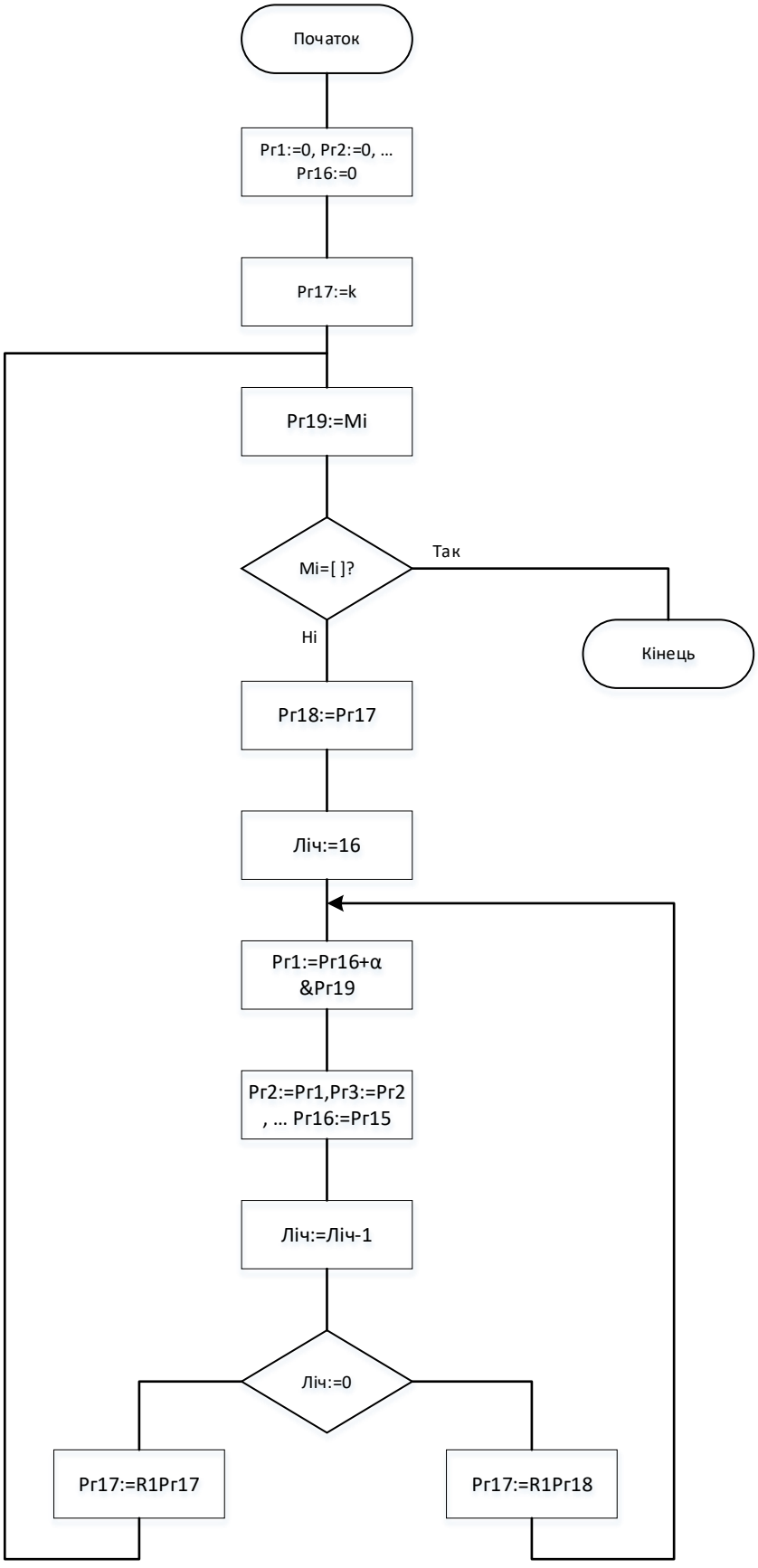
ІЛЮСТРАТИВНА ЧАСТИНА

Засіб байторієнтованого гешування даних

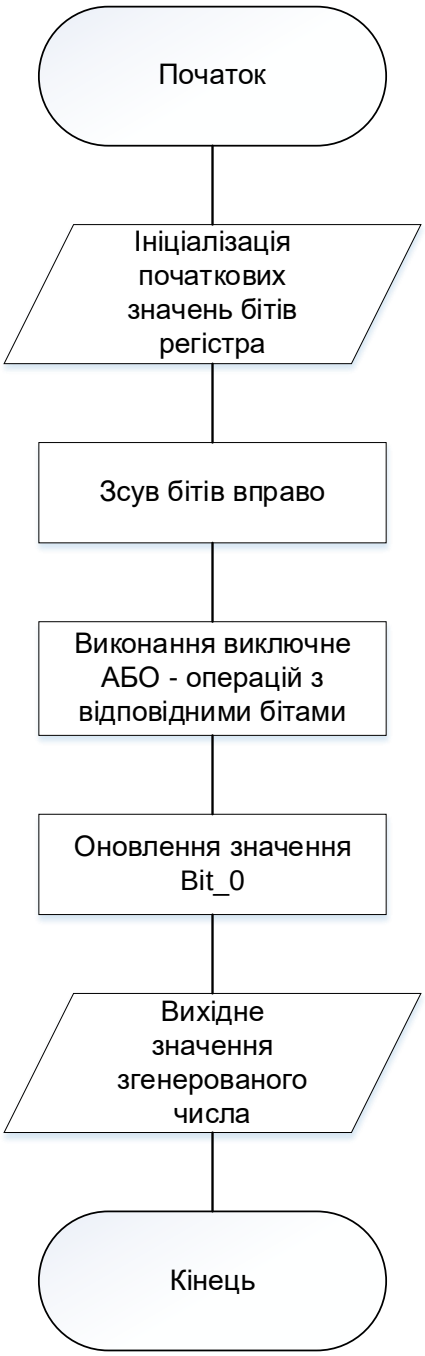
Виконав: студент 2 курсу групи 1БС-21мс
спеціальності 125 Кібербезпека
_____ Давимока М.В
19 червня 2023 р.

Керівник: зав. кафедри ЗІ д.т.н., проф
_____ Лужецький В.А.
19 червня 2023 р.

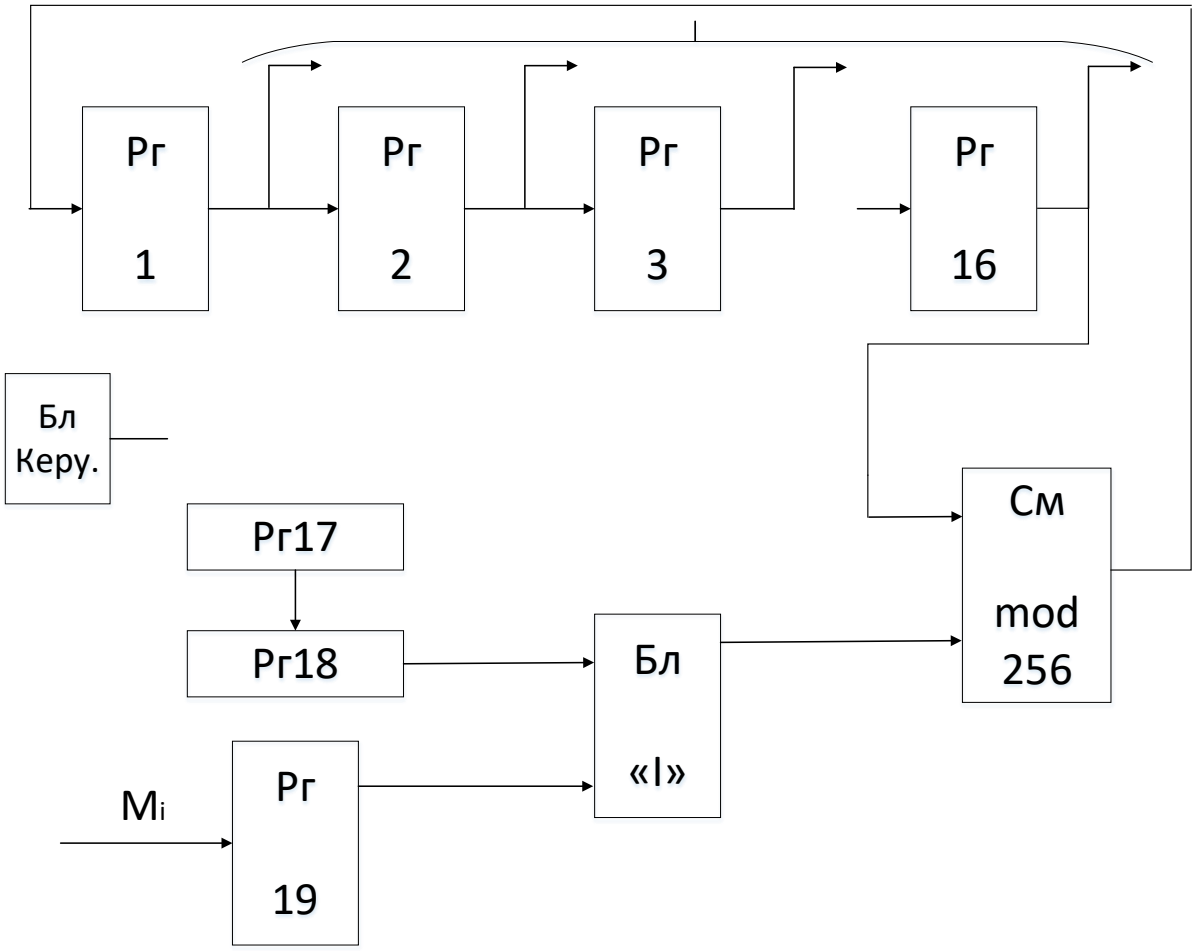
АЛГОРИТМ ПРОЦЕСОРА БАЙТОРІЄНТОВАНОГО ГЕШУВАННЯ ДАНИХ



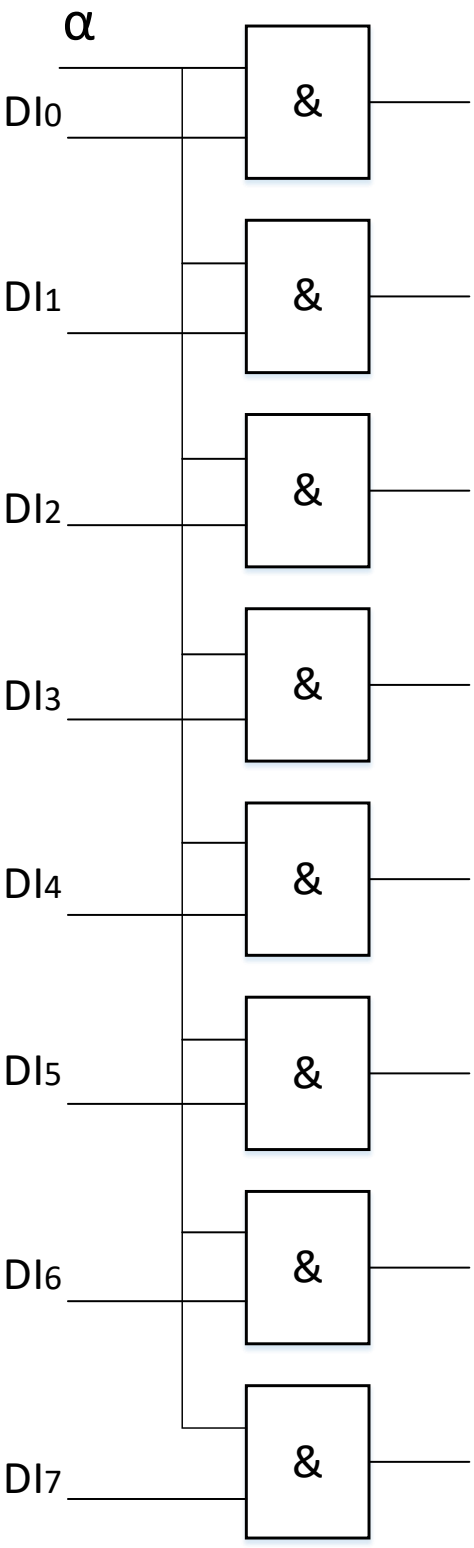
СТРУКТУРА ПРОЦЕСУ ГЕНЕРАЦІЇ ПСЕВДОВИПАДКОВОГО ЧИСЛА



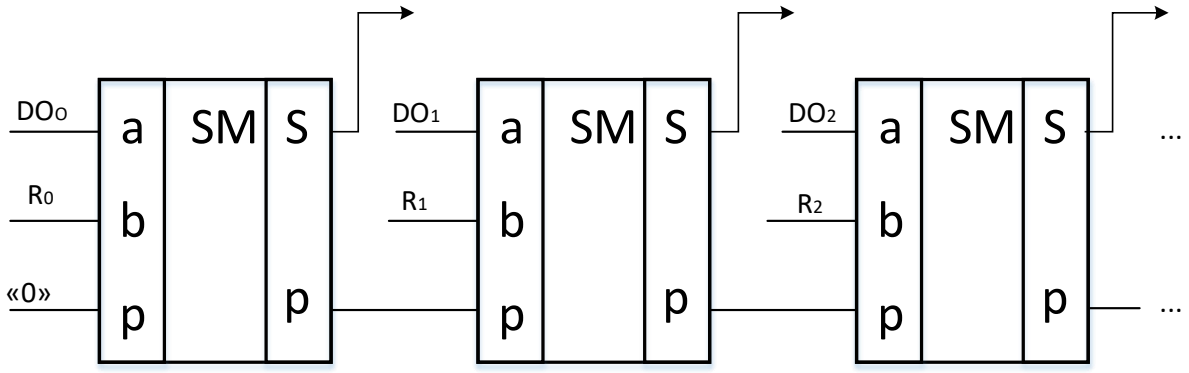
СТРУКТУРА ПРОЦЕСОРА БАЙТОРІЄНТОВАНОГО ГЕШУВАННЯ ДАНИХ



СТРУКТУРА БЛОКУ ЛОГІЧНИХ ЕЛЕМЕНТІВ



СТРУКТУРА 8-РОЗРЯДНОГО КОМБІНАЦІЙНОГО ПАРАЛЕЛЬНОГО СУМАТОРА



СТРУКТУРА РЕГІСТРІВ 1-16