Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему:
«МЕТОД ТА ЗАСІБ КРИПТОАНАЛІЗУ ГЕШ-ФУНКЦІЙ»

Виконав: студент 2 курсу, групи БС-21м
спеціальності – 125 Кібербезпека
_____ В. В. Казміревський

Керівник: к. т. н, доцент каф. ЗІ
_____ Ю. В. Баришев
« 19 » грудня 2022 р.

Опонент: к. т. н, доцент каф. ПЗ
_____ Д. І. Катєльніков
« 19 » грудня 2022 р.

Допущено до захисту
Завідувач кафедри ЗІ
д. т. н., проф.
_____ В. А. Лужецький
« 19 » грудня 2022 р.

Вінниця ВНТУ – 2022 року

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 125 Кібербезпека
Освітньо-професійна програма – Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри ЗІ,
д.т.н., проф.
_____ В. А. Лужецький
«_15_» _Вересня_ 2022 року

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Казміревському Віталію Віталійовичу

1. Тема роботи: «Метод та засіб криптоаналізу геш-функцій»
керівник роботи: Баришев Юрій Володимирович, к. т. н., доцент кафедри ЗІ,
затверджені наказом ректора ВНТУ від 14 вересня 2022 року №203

2. Строк подання студентом роботи 19 грудня 2022 р.

3. Вихідні дані до роботи:

- позитивні тести;
- негативні тести;
- тести «чорної скриньки».

4. Зміст текстової частини: Вступ. 1. Аналіз та обґрунтування доцільності дослідження. 2. Метод тестування геш-функцій. 3. Засіб тестування геш-функцій. 4. Експериментальне дослідження засобу. 5. Економічна частина. Висновки. Перелік використаних джерел. Додатки.

5. Перелік ілюстративного матеріалу: Класифікація геш-функцій (плакат А4). Результати порівняльного аналізу атак на геш-функції (плакат А4). Метод криптоаналізу геш-функцій (плакат А4). Алгоритм диференційного криптоаналізу (плакат А4). Алгоритм дослідження статистичних властивостей геш-функцій (плакат А4). Структура засобу криптоаналізу геш-функцій (плакат А4). Алгоритм роботи засобу криптоаналізу геш-функцій (плакат А4). Результати експериментального дослідження стійкості та коректності реалізації геш-функцій (плакат А4).

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Баришев Ю. В., к. т. н., доц. каф. ЗІ		
2	Баришев Ю. В., к. т. н., доц. каф. ЗІ		
3	Баришев Ю. В., к. т. н., доц. каф. ЗІ		
4	Баришев Ю. В., к. т. н., доц. каф. ЗІ		
5	Лесько О. Й., к. е. н., доц., проф. каф. ЕПВМ		

7. Дата видачі завдання 1 вересня 2022 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Прим
1	Аналіз завдання. Вступ	01.09.2022 - 04.09.2022	
2	Розробка технічного завдання	05.09.2022 - 15.09.2022	
3	Аналіз джерел за напрямком магістерської кваліфікаційної роботи	16.09.2022 - 04.10.2022	
4	Розробка рішень	05.10.2022 - 08.10.2022	
5	Практична реалізація, моделювання, експериментування, результати	09.10.2022 - 14.11.2022	
6	Розробка розділу економічного обґрунтування доцільності розробки	15.11.2022 - 21.11.2022	
7	Аналіз виконання ТЗ, висновки	22.11.2022 - 24.11.2022	
8	Оформлення пояснювальної записки	25.11.2022 - 02.12.2022	
9	Попередній захист та доопрацювання МКР	03.12.2022 - 14.12.2022	
10	Представлення МКР до захисту, рецензування	15.12.2022 - 20.12.2022	
11	Захист МКР	21.12.2022 - 23.12.2022	

Студент _____ В. В. Казміревський

Керівник роботи _____ Ю. В. Баришев

АНОТАЦІЯ

УДК 004.056.2 + 004.42

Казміревський В. В. Метода та засіб криптоаналізу геш-функцій. Магістерська кваліфікаційна робота зі спеціальності 125 – кібербезпека, освітня програма – Безпека інформаційних і комунікаційних систем. Вінниця: ВНТУ, 2022. 81 с.

Укр. мовою. Бібліогр.: 30 назв; рис.: 20; табл. 15.

Магістерська кваліфікаційна робота присвячена розробці методу та засобу криптоаналізу геш-функцій з метою дослідження коректності та стійкості розробки криптографічних геш-функцій мовою Dart. В результаті аналізу методів дослідження функцій гешування обрано критерії перевірки: коректність реалізації, статистичні властивості та стійкість до диференційного криптоаналізу.

Розроблено метод та засіб криптоаналізу алгоритмів гушування. Розроблено архітектуру програмного засобу та алгоритми роботи його складових. Здійснено програмну реалізацію за допомогою фреймворку Flutter. Перевірено низку безключових геш-функцій, зокрема, і легких криптографічних геш-функцій. Виконано обґрунтування економічної доцільності використання розробленого засобу.

Ключові слова: геш-функція; криптоаналіз; прообраз; алгоритм; диференційний криптоаналіз, тестування.

ABSTRACT

V. Kazmireskyi. Method and tool for cryptanalysis of hash functions. Master's thesis for speciality 125 – cybersecurity. Vinnytsia: VNTU, 2022. – 81 p.

In Ukrainian language. Bibliographer: 30 titles; fig.: 20; tabl. 15.

The master's thesis is devoted to the development of a method and tool of hash functions cryptanalysis order to investigate the correctness and infeasibility of the development of cryptographic hash functions implemented by the computer language Dart. Due to performed analysis of hashing function research methods, the verification criterias are selected: the implementation correctness, the statistical properties, and the resistance to differential cryptanalysis.

A method and tool of hash functions cryptanalysis of are developed. The architecture of the software tool and its performance algorithms are developed. The software implementation is carried out using the Flutter framework. The set of the keyless hash functions are tested light-weight cryptographic hash functions in particular. The economic reasoning of developed software tool usage is proved.

Key words: hash function; cryptanalysis; prototype; algorithm; differential cryptanalysis, testing.

ЗМІСТ

ВСТУП

Сьогодні великої популярності набули легкі криптографічні геш-функції, зокрема для використання в мобільних пристроях. Дані геш-функції достатньо швидко замінюють один одного, з'являються нові алгоритми з різноманітними конструкціями побудови, які можуть суттєво відрізнятися один від одного. Також серед них немає певного фаворита, тому замовники тих чи інших програмних продуктів віддають перевагу використанню різних алгоритмів гешування у своїх продуктах. Тому, такі функції гешування потребують швидкого, якісного та універсального тестування.

Аналіз відповідних інформаційних джерел показав, що легкі криптографічні функції гешування, попри значну кількість тестів, є все ще не достатньо дослідженими, оскільки, швидко замінюють одне одного в першості використання у програмних продуктах. Часто не всі показники продуктивності легких криптографічних геш-функцій є протестованими. Також більшість із досліджень не надають результатів, що підтверджують криптографічні властивості даних алгоритмів гешування.

Тому, проведення криптоаналізу для відомих та нових криптоалгоритмів є дуже актуальним, так як вчасно можна сказати, що даний криптоалгоритм є нестійким, і що його потрібно вдосконалити або замінити новим.

Об'єктом дослідження є процес криптографічних перетворень.

Предметом дослідження є криптоаналіз геш-функцій.

Метою даної роботи є покращення безпеки програмних продуктів для мобільних пристроїв, які використовують геш-функції, за рахунок розробки методу та засобу криптоаналізу функцій гешування для дослідження показників їх стійкості та збільшення стійкості програмних засобів для мобільних пристроїв, що використовують криптографічні бібліотеки на мові Dart.

Розробка призначена для перевірки коректності та стійкості реалізації методів гешування в розроблюваних застосунках або фрагментах коду третіх сторін. Для досягнення мети необхідно розв'язати такі задачі:

- проаналізувати сучасні криптоаналітичні методи та найчастіше використовувані алгоритми гешування;
- проаналізувати наявні методи криптоаналітичних примітивів та сформулювати вимоги до методу та засобу криптоаналізу геш-функцій;
- виконати математичний опис процесу криптоаналізу геш-функцій;
- розробити метод криптоаналізу геш-функцій;
- розробити алгоритми, що реалізують запропонований метод;
- реалізувати засіб, який шляхом виконання набору тестів виконує криптоаналіз геш-функцій;
- виконати економічне обґрунтування доцільності використання запропонованого засобу.

Під час дослідження використовувались методи аналізу, синтезу, проблемного виконання, абстрактних автоматів, тестування чорної скрині, тестування сірої скрині, динамічного тестування.

Наукова новизна полягає в тому, що удосконалено криптоаналіз, який, на відміну від відомих, використовує код Грея, що дозволяє автоматизувати криптоаналіз.

Практична цінність роботи: засіб виконання автоматизованого диференційного криптоаналізу геш-функцій.

Результати магістерської кваліфікаційної роботи були представлені на 4 конференціях: L Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії, LI Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії, L Науково-технічна конференція факультету менеджменту та інформаційної безпеки, XLVII Науково-технічна конференція Інституту екологічної безпеки та моніторингу довкілля [1-4].

Магістерська кваліфікаційна робота виконувалася на замовлення ТОВ ElephantsLab (Додаток А), а її результати впроваджені на цьому підприємстві (Додаток Б).

1 АНАЛІЗ ТА ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ ДОСЛІДЖЕННЯ

1.1 Поняття геш-функцій

Геш-функція – це криптографічна (математична) функція, яка приймає повідомлення довільної довжини на вхід і створює вихід фіксованої (як правило, меншої) довжини, який зазвичай називають відбитком або дайджестом повідомлення [1, 2]. Також в науковій літературі можна зустріти інші назви, а саме: геш, геш-образ, геш-код, згортка, дайджест повідомлення, криптографічна контрольна сума, код автентичності повідомлення, код виявлення маніпуляцій.

Визначення функції гешування можна формалізувати таким чином: H називають геш-функцією, якщо вона відображає повідомлення довільної довжини n в образи скінченної довжини l:

$$H: \{0,1\}^n \rightarrow \{0,1\}^l \tag{1.1}$$

Оскільки, вхідне повідомлення геш-функції в переважному більше за повідомлення на виході, то можна стверджувати, що це функція багато-до-одного. З цієї причини завжди існують принаймні два повідомлення, які мають однаковий образ, який називають колізією [1-5]. На рисунку 1.1 зображено приклад виникнення колізії при гешуванні даних m_1 і m_2.

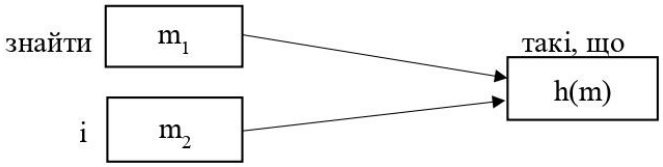


Рисунок 1.1 – Зображення прикладу колізії

Уникнути колізій неможливо, але для криптографічного геш-алгоритму очікується, що виявлення колізій повинно бути не обчислювальним.

Також, щоб геш-функція вважалася зламаною, потрібно мати можливість знайти перший або другий прообраз.

Суть знаходження першого прообразу полягає в тому, що зловмисник, знаючи геш-код $H(m)$ деякого повідомлення m, намагається обчислювально знайти вхідне повідомлення m, щоб $H(m)$було гешем m [1-5].

Знаходження другого прообразу полягає в тому, що зловмисник, знаючи геш-образ першого повідомлення m_1, намагається знайти друге повідомлення m_2, щоб їх геш-значення збігалися [1-7]:

$$H(m_1)= H(m_2) \tag{1.2}$$

Якщо такі прообрази легко знайти алгоритмічно або обчислювально, то розглянута геш-функція вважається поганою з криптографічної точки зору.

Отже, основною вимогою до алгоритмів криптографічного гешування є їх стійкість до колізій.

Далі в магістерській кваліфікаційній роботі проведено аналіз найвідоміших безпечних моделей конструкцій геш-функцій, а саме:

- конструкції Merkle-Damgard;
- ітеративної конструкції на основі конструкції Merkle-Damgard;
- губчатої конструкції.

Конструкція Merkle-Damgard є найбільш широко використовуваним методом геш-конструкції, який був розроблений R. Merkle та I. Damgard в 1989 році. Більшість функцій гешування побудовані саме на цій конструкції. Конструкція Merkle-Damgard в основному обробляється в три етапи [6]:

1) Крок доповнення. Мета доповнення полягає в тому, щоб зробити довжину повідомлення кратною довжині блоку повідомлення m. Найпоширеніша процедура доповнення така: біт «1», за яким слідує кількість «0» бітів. Кількість бітів «0», що додаються до повідомлення, вибирається так, щоб довжина повідомлення стала кратною довжині блоку m. Загалом максимальна довжина повідомлення, яке може бути оброблено геш-функцією, становить 264 - 1. Тому для доповнення довжини надається 64-бітний простір. Враховуючи доданий біт «1»,

щонайменше 65 бітів додаються до всіх повідомлень незалежно від довжини повідомлення.

2) Другим кроком є розділення доповненого повідомлення на m бітових блоків m_0 m_1 m_2 . . . m_{t-1}.
3) Після другого кроку значення ланцюжка знаходять ітеративно за допомогою фіксованого загальновідомого вектора ініціалізації.

На рисунку 1.2 зображено схему роботи конструкції Merkle-Damgard.

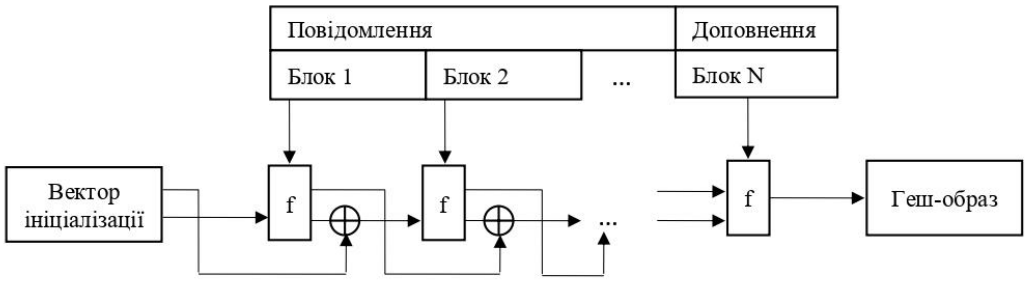


Рисунок 1.2 – Зображення схеми роботи конструкції Merkle-Damgard

Найважливішою властивістю структури Merkle-Damgard є стиснення колізій. Алгоритми MD4 і MD5 стали основою для наступних геш-функцій, що мають однакову конструкцію Merkle-Damgard, а саме: SHA-0, SHA-1, SHA-2, RIPEMD, RIPEMD-160 [6].

Отже, проаналізувавши структуру Merkle-Damgard можна вивести такі твердження:

- якщо функція ущільнення, яка використовується в ітерованій геш-функції, вразлива до певної атаки, то й геш-функція також буде вразлива до атак;
- якщо функція ущільнення, що використовується в алгоритмі гешування, є стійкою до колізій, то сама геш-функція є стійкою до колізій.

Однак із зростанням обчислювальної потужності та створенням нових криптоаналітичних інструментів геш-функції Merkle-Damgard стають слабшими та вразливішими до атак. Біхам і Данкельман, виправляючи недоліки в структурі Merkle-Damgard, створили нову конструкцію побудови геш-функцій. Так

з’явився ітерований метод побудови функцій гешування на основі конструкції Merkle-Damgard [5-6].

Ітерована конструкція використовує різні вектори ініціалізації для різних геш-значень, тому ущільнення не викличе жодних проблем. Даний метод побудови об’єднує побітове позначення розміру гешу після доповнення довжини. Це гарантує, що не існує двох повідомлень, які мають однакове геш-значення без ущільнення для різних розмірів гешу [5-6].

На відміну від стандартів SHA-1 і SHA-2, SHA-3 в основному покладається на структуру губки, де дані поглинаються та витискаються для генерації виходу гешу. Конструкція губки складається з двох фаз:

- поглинання;
- стискання.

Геш-фунції зайняли впевнене місце в розвитку IT технологій і зайняли доволі обширну та різноманітну область їх застосування. В першу чергу, геш-функції використовують для передачі даних комп’ютерними мережами. Прості геш-функції застосовують в протоколі TCP/IP для перевірки цілісності передачі пакетів. Також їх застосовують для виявлення апаратних помилок – так зване "надлишкове кодування". Якщо геш отриманого пакета даних збігається з відправленим разом з пакетом даних (так званою «контрольною сумою»), то це може означати, що втрати чи помилок при передачі пакета даних мережею не сталось. Якщо ж геші не збігаються, то можливо при передачі пакета відбулась втрата даних, у такому разі пакет пересилається знову. У даному випадку використовується проста геш-функція, тому що при передачі даних важлива швидкість. Геш-функції використовують для перевірки цілісності даних. За допомогою спеціально розробленого ПЗ обчислюють геш переданого або завантаженого файлу і порівнюють його з тим, що було наведено в першоджерелі файлу. Якщо геші збігаються – файл завантажено без помилок. Геш-функції використовують у криптографії. Для доступу до сайтів та сервісів за логіном і паролем часто використовують великі за складністю геш-функції. Паролі зберігають у вигляді гешу, адже у відкритому вигляді їх зберігати не доцільно.

Також гешування використовують у технології електронного цифрового підпису [1-6].

В тому числі необхідно виділити такі застосування функцій гешування:

- механізми контролю цілісності та справжності інформації та ресурсів;
- протоколи електронного підпису, електронного штампу та мітки часу);
- алгоритми генерація псевдовипадкових послідовностей;
- криптографічні протоколи узгодження та встановлення ключів;
- контроль цілісності баз даних, у тому числі без використання спільного секрету тощо.

Найбільш актуальним на сьогоднішній день є використання геш-функцій у блокчейні. Алгоритми гешування в системах типу блокчейн використовуються під час створення нового блоку і валідації транзакції, оскільки, згідно із протоколом та ідеєю, кожна нова ланка має містити інформацію про попередню. Функції гешування чудово підходять для вирішення цього завдання, оскільки вони досить ефективні і з точки зору інформації не займають багато фізичного місця.

Геш-функції сімейства SHA-2 є доволі потужними та підходять для їх застосування в мережі Bitcoin. SHA-2 використовуються в мережі Bitcoin двома основними способами, для майнингу та для створення адрес Bitcoin.

Окрім криптовалюти Bitcoin (BTC), на алгоритмі SHA-2 базуються ще 43 блокчен-проекти. Основні з яких є: Bitcoin Cash, Bitcoin SV, Litecoin та інші.

На сьогодні відомо, що SHA-2 схильні до атаки подовження довжини. Тому у технології блокчейн починають застосовувати SHA-3. SHA-3 краще відповідає потребам ринків капіталу завтрашнього дня, ніж те робив SHA-2. SHA-3 має потрібні характеристики для зміцнення довіри до P2P-мережі, яка не покладається на централізованих посередників для видачі дозволів та повноважень.

Отже, аналіз поняття геш-функцій показав, що основною метою таких функцій є відображення повідомлень довільної довжини в фіксовану за допомогою стискання, забезпечивши при цьому три основні вимоги:

- стійкість до колізій;
- стійкість до знаходження першого прообразу;
- стійкість до знаходження другого прообразу.

1.2 Аналіз геш-функцій

Найпопулярнішими алгоритмами гешування на сьогоднішній день є такі: SHA-2, SHA-3, RIPEMD-160. Дані геш-алгоритми безпосередньо використовуються в технології блокчейн.

SHA-2 – це наступник SHA-1, схвалений і рекомендований NIST. SHA-2 являє собою сімейство з шести алгоритмів з різними розмірами дайджесту: SHA-224 , SHA-256, SHA-384 , SHA-512 , SHA-512/224, SHA-512/256 [7].

Найбільш істотна різниця між варіантами полягає в тому, що деякі призначені для 32-бітних регістрів, а деякі призначені для 64-бітних регістрів. З точки зору продуктивності це єдина різниця, яка має значення. Геш-функції сімейства SHA-2 побудовані на основі структури Merkle-Damgard [7].

SHA-3, також відомий як Keccak (його оригінальна назва до того, як його було обрано переможцем конкурсу NIST SHA-3), є абсолютно новим алгоритмом хешування, який не має нічого спільного з SHA-1 і SHA-2 [8].

SHA-3 базується на новому криптографічному підході під назвою «губчаста конструкція», який використовує Keccak . По суті, дані «вбираються» в губку, потім «видавлюється» результат, так само як губка вбирає і відпускає воду. SHA-3 – це сімейство з чотирьох алгоритмів із різними геш-функціями, а також дві розширювані функції виводу, які можна використовувати для гешування домену, рандомізованого гешування, шифрування потоку та генерації MAC-адрес: SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE-128 і SHAKE-256 [8-11].

RIPEMD-160 являє собою 160-бітну криптографічну геш-функцію. RIPEMD160 розроблена Хансом Доббертіном, Антоном Босселарсом і Бартом Пренілом. RIPEMD-160 являє собою поліпшений варіант RIPEMD з 160-бітовим гешем на виході. Особливості функції засновані на попередніх геш-функціях

MD4, MD5 і RIPEMD. Розмір геш-результату та змінної ланцюжка для RIPEMD-160 збільшено до 160 біт (п’ять 32-бітних слів), кількість раундів збільшено з трьох до п’яти, а два рядки стали більш різними (не лише змінюються константи, а також булеві функції та порядок слів повідомлення) [12, 13].

Вище згадані геш-функцій використовують в різних цілях, зокрема, і для контролю цілісності інформації, наприклад, експериментальних даних про дослідження фізичних властивостей води [4].

Також великої популярності набули легкі криптографічні геш-функції, зокрема для використання в мобільних пристроях. Однак, на відміну від вище згаданих алгоритмів гешування, дані геш-функції дуже швидко замінюють один одного, з’являються нові алгоритми, які можуть суттєво відрізнятися один від одного. Такі функції гешування потребують якісного тестування.

Було проведено багато досліджень щодо прогресу досліджень безпеки Інтернету речей (IoT), до яких відносять і мобільні пристрої, опублікованих в останні роки. Дослідники в основному зосереджені на рішеннях проблем безпеки IoT. Проблеми безпеки загалом представлені як компоненти кожного дослідження та розглядаються як загальні поняття, а безпека та конфіденційність часто розглядаються разом як одне поняття. На жаль, немає достатньо детальних та актуальних досліджень, що описують глибокі проблеми безпеки мобільних пристроїв, пов’язані з легкими криптографічним геш-функціями [14].

Дизайн легких криптографічних геш-функцій також побудований на трьох популярних конструкціях: конструкції Merkle-Damgard, губчатій конструкції та на конструкції на основі блочного шифру. Деякі алгоритми використовують змішану конструкцію, наприклад Merkle-Damgard або губчата, як основну конструкцію та інші конструкції (наприклад, на основі блочного шифру) як будівельні блоки для розробки функцій стиснення або перестановок [14].

До легких криптографічних геш-функцій відносять алгоритми сімейства ARMADILLO побудованих на основі конструкції Merkle-Damgard з різними варіантами довжини вихідного гешу, а саме: 80 біт, 128 біт, 160 біт, 192 біт та 256 біт.

Першою геш-функцією на основі легкої губчастої конструкції була QUARK. Алгоритм був розроблений на основі потокового шифру Grain і блокового шифру KATAN. Для перестановки бітів використовувалися два регістри зсуву з нелінійним зворотним зв'язком і регістр зсуву з лінійним зворотним зв'язком.

Ще одним прикладом легкої функції, побудованій на губчастій конструкції, є геш-функція SPONGENT. SPONGENT було розроблено як сімейство геш-функцій із 88, 128, 160, 224, 256-бітним геш-значенням. Автори даного алгоритму стверджують, що алгоритм стійкий до атак, спрямованих на геш-функції.

Також до алгоритмів, побудованих на губчатій конструкції, відносять такі: PHOTON, GLUON, SPN-Hash, SipHash, LHash, Neeva-hash, Hash-One, Gimli-Hash, sLiSCP-hash, XOODYAK, ASCON-HASH, KNOT-Hash, DryGascon-Hash, PHOTON-Beetle-Hash, HVH та багато інших.

DM-PRESENT є легкою геш-функцію, яка використовує блоковий шифр PRESENT і конструкцію DaviesMeyer. Є два типи DM-PRESENT геш-функції: DM-PRESENT-80 і DM-PRESENT-128.

TWISH розроблено на базі алгоритму TWINE-128 блокового шифрування і використовує конструкцію DM. TWISH – це геш-функція довжини одного блоку, яка приймає 128-бітне вхідне повідомлення та повертає 64-бітове геш-значення. Введення повідомлення в схемі DM діє як ключ [14].

Отже, легких криптографічних геш-функцій є дуже багато з різноманітними конструкціями побудови. А їх аналіз та огляд відповідних літературних джерел показав, що легкі криптографічні функції гешування, попри значну кількість тестів, є все ще не достатньо дослідженими, оскільки, дуже швидко замінюють одне одного. Часто не всі показники продуктивності легких криптографічних геш-функцій є протестованими. Також більшість із досліджень не надають підтверджуючих результатів криптографічних властивостей даних алгоритмів гешування.

1.3 Атаки на алгоритми гешування

Атаки на геш-функції розділяють на дві такі групи:

- атаки, що базуються на вразливостях алгоритму перетворень;
- атаки, що є не залежними від алгоритму гешування.

Атаки, які не залежать від алгоритму:

- атака "грубою силою",
- атака методом "дня народження",
- повний перебір ключів,
- райдужні таблиці,
- атака "зустріч посередині",
- атака подовженням довжини,
- атака на основі мультиколізій.

Аналітичні атаки:

- атака з корекцією блоку,
- атака з фіксованою точкою,
- атака на базовий алгоритм шифрування,
- лінійний криптоаналіз,
- диференційний аналіз.

Дані атаки ґрунтуються на недоліках структури функції гешування та алгоритмів її перетворень.

Метод грубої сили (brute-force attack) передбачає перебір всіх можливих варіантів гешованих даних до знаходження необхідного значення. Застосовують атаки на геш-функції методом грубої сили для досягнення таких цілей:

- пошук колізії;
- пошук прообразів;
- визначення секретного ключа (для ключових алгоритмів гешування).

На даний момент, розроблені атаки на прообраз є практично не придатними. Оскільки, якщо таку атаку буде можливо застосувати на практиці,

то це спричинить сильний вплив на протоколи Інтернет мережі. Тобто, атака є затратною, оскільки потребує занадто багато часу та ресурсів.

Для алгоритмів гешування, у яких довжина геш-образу є невеликою, загальним методом пошуку колізій є атака "парадокс дня народження". Дана атака є достатньо розповсюдженою і була описана Ювалом в 1979 році. Парадокс днів народження твердить, що ймовірність збігу дат днів народження хоча б у двох членів групи з 23 і більше осіб, перевищує 0,5.

«Зустріч посередині» вважають однією з можливих варіантів атак, що використовують парадокс «днів народження», хоча й її метою не є пошук колізій. Дана атака порівнює збіги проміжних величин, тому вона є ефективнішою за атаку «парадокс днів народжень».

Також для атаки пошуку прообразу використовують коригуючі блоки. Суть даної атаки в тому, що зловмисник робить заміну всіх блоків вихідного повідомлення M з геш-значенням H на будь-які інші блоки, які йому необхідно. Але окрім одного або декількох останніх блоків, які зловмисник замінює таким чином, щоб значення гешу повідомлення також було рівним H.

Атака за допомогою нерухомих точок (fixed points). Виконується тоді, коли проміжні обчислення гешу повідомлення не змінюють його поточного значення. В такому випадку дану атаку реалізовують за допомогою додавання будь-якої кількості блоків зі значенням M_i, по тій причині, що вони ніяким чином не впливають на результати обробки.

Також є алгоритми гешування, які дають можливість виконувати маніпуляції на рівні блоків гешованого повідомлення без зміни геш-образу. До таких маніпуляцій відносять:

- видалення;
- вставка;
- заміна;
- перестановка блоків.

Більшість такого роду атак перетинаються з атаками на основі коригуючих блоків. Коли змінення гешу, через певні модифікації над деяким блоком,

коригується одним з наступних блоків. Значення такого блоку обчислюють певним способом, щоб відновити належним чином геш-значення.

Більшість алгоритмів гешування реалізовані на основі блокових шифрів. З цієї причини, деякі слабкі місця, притаманні блоковим шифрам, можна використати на таких функціях гешування. Тому для атаки на геш-функції, побудованих на основі блокових шифрів, можливо застосувати метод бумеранга. Таку атаку, а саме посилену атаку методом бумеранга, можливо практично застосувати для злому алгоритму гешування. Дана атака відноситься до аналітичних атак та є диференційною. Ідея атаки полягає у використанні ретельно підібраного глобального диференційного шляху, який використовується в класичних атаках диференційного криптоаналізу [9-10]. Таку атаку було успішно застосовано до геш-функції SHA-1.

Значну роль для дослідження та аналізу функцій гешування відіграють їх конструкції та їх властивості. Оскільки конструкції алгоритмів гешування також піддаються атакам [7, 11]. Складність реалізації атак на конструкції алгоритмів гешування не залежить від методу реалізації геш-функції, а тому їх можна застосувати для будь-якої геш-функцій тієї чи іншої конструкції. Найпоширенішими для побудови алгоритмів гешування є конструкції Merkle-Damgard та деякі її модифікації. Але й інші конструкції, які використовують для побудови геш-функцій, відмінних від конструкції Merkle-Damgard, також є вразливими [12-14]. До таких атак відносять [7, 15-18]:

- сталу атаку збільшення довжини повідомлення, яка полягає у дописуванні до повідомлення блоків даних до тих пір, поки геш-значення початкового та доповненого повідомлень не збігаються;
- сталу атаку Жу, що використовує мультиколізії.

1.4 Диференційний криптоаналіз геш-функцій

Диференціальний криптоаналіз – це атака за допомогою вибраного відкритого тексту, у якій для визначення значення бітів ключа використовується велика кількість пар відкритий текст-зашифрований текст. Статистична

інформація про ключ виводиться з блоків зашифрованого тексту, отриманих шляхом шифрування пар блоків відкритого тексту з певною побітовою різницею A' під цільовим ключем. Коефіцієнт роботи атаки критично залежить від найбільшої ймовірності $P(B' \vee A')$ з різницею B' на деякому фіксованому проміжному етапі криптографічної функції, наприклад, на вході останнього раунду. У першому наближенні ймовірності $P(B' \vee A')$ для DES вважаються незалежними від конкретного значення ключа [19-21].

Ключова інформація витягується з вихідних пар у такий спосіб:

- для кожної пари передбачається, що проміжна різниця дорівнює B';
- абсолютні значення виходів і (передбачувана) проміжна різниця B' накладають обмеження на кількість l ключових бітів останнього раунду.

Атака є успішною, якщо правильне значення підключа пропонується значно частіше, ніж будь-яке інше значення. Пари з проміжною різницею, що не дорівнює B', називаються неправильними парами. Значення підключів, запропоновані цими парами, загалом неправильні. Правильні пари з проміжною різницею, що дорівнює B', не лише припускають правильне значення підключа, але часто також являються рядом неправильних значень підключа. Для DES неправильні пропозиції можна вважати рівномірно розподіленими серед можливих ключових значень, якщо значення $P(B' \vee A')$ значно більше, ніж $P(C' \vee A')$ для будь-якого $C' \neq B'$ [20-23].

За даних умов є сенс розрахувати співвідношення між кількістю пропозицій правильного значення та середньою кількістю пропозицій на запис, тобто відношення сигнал/шум. Якщо розмір таблиці дорівнює 2^l, а середня кількість запропонованих підключів на пару дорівнює γ, це співвідношення дорівнює $P(B' \vee A')2^l / \gamma$. Відношення сигнал/шум сильно впливає на кількість правих пар, необхідних для однозначної ідентифікації правильного значення підключа [21-25].

Для співвідношення 1-2 достатньо близько 20-40 правильних пар. Для більших співвідношень потрібно лише кілька правильних пар, а для

співвідношень, які набагато менші за 1, необхідна кількість правих пар може зробити практичну атаку нездійсненною.

Великі ймовірності $P(B' \vee A')$ локалізуються побудовою так званих характеристик. Характеристика m-раунду становить m+1 кортеж розбіжних шаблонів: (X'_0, X'_1, …, X'_m). Імовірність цієї характеристики – це ймовірність того, що початковий відмінний шаблон X_0 поширюється на розбіжні шаблони X'_1, X'_2, …, X'_m відповідно після 1, 2, …, m раундів. У припущенні, що ймовірність поширення від X'_{i-1} до X'_i не залежить від поширення від X'_0 до X'_{i-1}, ця ймовірність визначається як [25]:

$$\prod_i P(X'_i \vee X'_{i-1}) \,, \tag{1.5}$$

де $P(X'_i \vee X'_{i-1})$ ймовірність того, що відмінний шаблон X'_{i-1} на вході циклічного перетворення породжує X'_i на його виході. Отже, багатораундова характеристика розкладається на ряд одноraундових характеристик (X'_{i-1}, X'_i) з імовірністю $P(X'_i \vee X'_{i-1})$.

1.5 Постановка задачі

На теперішній час в мобільних пристроях є популярним використання легких криптографічних геш-функцій, які дуже швидко замінюють один одного.

Аналіз джерел інформації з досліджуваного питання показав, що існує проблема недостатньої кількості методів аналізу таких геш-функцій. Більшість методів заточені під блокові шифри. Використання таких методів для тестування малоресурсних геш-функцій не завжди є доцільним, оскільки геш-функцію варто розглядати не як те, в основі чого лежить якийсь криптографічний примітив, а як окремий повноцінний метод захисту інформації, який потребує специфічного тестування. Тобто, в більшості випадків можна зустріти, що для криптоаналізу геш-функцій використовуюся ті ж самі методи аналізу, що й для блокових шифрів. Актуальність порушення даного питання підкріплюється і відсутністю достатньої кількості детальних та актуальних досліджень щодо глибоко аналізу

безпеки мобільних пристроїв, пов'язаних з легкими криптографічним геш-функціями.

Майже кожного дня з'являються нові геш-функції для мобільних пристроїв, які конкурують між собою. Тому серед таких алгоритмів гешування немає фаворитів, і підібрати якийсь алгоритм для вирішення тієї чи іншої задачі доволі складно, адже криптоаналізу таких геш-функцій приділяється мало уваги через їх велику кількість та динамічний розвиток. Тому такі геш-функції потребують швидкого та якісного дослідження.

Саме тому актуально розробити метод тестування для легких криптографічних геш-функцій та засіб для автоматичного тестування.

1.6 Висновки до розділу

У першому розділі магістерської кваліфікаційної роботи були розглянуті та проаналізовані особливості дослідження та сучасні вимоги до легких криптографічних геш-функцій. Розглянуто та проаналізовано поняття геш-функцій та атаки на них.

Отже, аналіз поняття геш-фунцій показав, що криптографічна функція гешування повинна бути стійкою до:

- пошуку колізій;
- знаходження першого прообразу;
- знаходження другого прообразу.

Дослідження конструкцій побудови геш-функцій виявило, що алгоритми гешування можуть бути як і дуже схожими між собою, так і суттєво відрізнятися.

Аналіз джерел показав, що: функції гешування, які використовуються в мобільних пристроях, доволі швидко замінюють один одного. І дослідження таких функцій в мобільних пристроях в основному розглядаються в контексті загальних понять.

Тому варто розробити метод та засіб для криптоаналізу алгоритмів гешування.

2 МЕТОД ТЕСТУВАННЯ ГЕШ-ФУНКЦІЙ

2.1 Математичний опис процесу тестування геш-функцій

При теоретичному дослідженні та описі процесу тестування криптографічних геш-функцій варто подати деякий формальний опис. Поставлена в попередньому розділі задача потребує формалізації у вигляді математичного опису для визначення подальших етапів розробки МКР.

Тестування криптографічних геш-функцій – це процес перевірки заявлених до функції гешування вимог і коректності її реалізації, який здійснюють шляхом аналізування її поведінку в штучно створених ситуаціях, які описуються в необмеженому (окрім часу та ресурсів) наборі тестів, обраних певним чином.

Для тестування геш-функцій обрано такі критерії оцінювання:

- відповідність вимогам, якими керувалися розробники функції,
- відповідність виходів для всіх можливих вхідних даних.

Також додатковими вимогами до тестування функції гешування можуть бути її практичність та її виконання за прийнятний час.

Тому для математичного опису процесу тестування геш-функцій пропонується використовувати кібернетичний підхід, який передбачає визначення значень вхідних сигналів T, вихідних сигналів R, внутрішніх станів S та виконуваних перетворень F.

Як показав аналіз геш-функцій в першому розділі, що важливо аналізувати адекватність реалізації програмного коду бібліотек певних функцій гешування, тому входом T математичного опису процесу тестування буде множина трьох функцій:

- *initialize*(·) – ініціалізація певної геш-функції,
- *process*(·) – процес гешування,
- *finalize*(·) – отримання результату гешування.

Вихідні результати будуть бінарними, true або false, які вказуватимуть на те, чи пройшла геш-функція тест чи ні. Також можливим результатом є число, яке потрібно визначити чи лежить воно в певному допустимому діапазоні. Отже,

на виході третє значення не дано. Тому на виході математичного опису буде тільки множина булевих значень R ={$output_i$}, $output_i$ є {true, false}.

Операцією буде множина нескінченної кількості тестів {$f_i(\cdot)$}, де i = 1, 2, ..., ∞.

Множина внутрішніх станів S є порожньою множиною, оскільки для тестування непотрібно додаткових станів, а кожен стан має на меті перевірку відповідності конкретному показнику стійкості.

Множина вхідних станів представлена множиною тестів.

В результаті виконання математичного опису процесу тестування геш-функцій отримано вираз, який дорівнює:

$$\{\{inialize, process, finalize\}_i\} \cdot \{bool_i\} \cdot \emptyset \cdot \{tests_i\} \qquad (2.1)$$

Отже, з математичного опису процесу тестування функцій гешування видно, що:

- тестування є дискретним,
- тест не залежить від інших тестів,
- тестування однієї геш-функції не впливає на тестування інших функцій.

Тому, тести мають бути універсальними, щоб їх можна було застосувати для будь-якої функції гешування.

2.2 Вимоги до тестування геш-функцій

Для того, щоб коректно розробити метод та засіб для тестування геш-функцій, необхідно висунути певні вимоги, зокрема, враховуючи результати математичного опису процесу тестування функцій гешування.

Перша вимога, яка випливає з першого розділу МКР, це масштабованість. Оскільки, під час розробки, варто подбати про те, щоб засіб задовольняв подальші вимоги та працював належним чином незалежно від його розміру та обсягу.

Наступними вимогами є функціональність та універсальність засобу тестування. Засіб повинен мати змогу тестувати будь-яку геш-функцію, яку захоче користувач, навіть ту, яку він розробив самостійно. А також підтримувати легку та зрозумілу інтеграцію нових тестів для адекватної взаємодії з уже розробленими тестами. Загалом, не менш важливо, забезпечити достатню належну множину функцій даного засобу для розв'язання ним поставлених задач та цілей, поставлених користувачами, що також дасть можливість покращити систему управління персоналом підприємства [3].

Засіб для тестування повинен бути здатний зберігати свій рівень якості функціонування встановлених умовах. Зокрема, це забезпечить практичність та зручність використання для користувача. Та дозволить провести якісне вичерпне тестування геш-функції. Для цього потрібно забезпечити такі характеристики:

- відсутність помилок. Тестування має уникати відмови через внутрішні дефекти засобу. Тому дані дефекти необхідно мінімізувати до нуля;
- стійкість до дефектів. У разі виникнення непередбачуваного дефекту або порушень специфікованого інтерфейсу, засіб повинен підтримувати встановлений рівень функціонування;
- відновлюваність. Здатність відновлювати функціонування на заданому рівні;

Також варто дотриматися вимоги універсальності, тобто забезпечити ефективне співвідношення між рівнем якості функціонування і об'ємом використовуваних ресурсів. Це забезпечить стабільність роботи засобу, здатність мінімізувати неочікувані ефекти модифікацій, зручність для його тестування, оскільки, тестування є невід'ємною частиною кожної розробки засобу, та змінюваність, щоб користувач міг самостійно додавати свої власні тести.

Отже, вимоги до тестування геш-функцій є такими:

- масштабованість;
- функціональність ;
- універсальність.

2.3 Обґрунтування вибору наборів тестів

Аналіз джерел показав, що для перевірки на коректність реалізації геш-функцій існують такі набори тестів:

- NIST SP 800-22 [25];
- DIEHARD [26];
- ENT [27];
- KAT [28].

Оскільки, дані набори тестів призначені для тестування випадкових бітових послідовностей, то можна заздалегідь зробити висновок, що не всі із запропонованих тестів підійдуть для перевірки на коректність реалізації геш-функцій. Також, не всі тести із даних наборів тестів зможуть забезпечити універсальність засобу тестування геш-функцій.

Отже, необхідно проаналізувати дані набори тестів, оскільки вони більше «заточені» під псевдовипадкові послідовності, а не геш-функції, відповідно до критеріїв в пункті 2.2.

Пакет NIST SP 800-22 включає в себе 15 тестів.

Тести 1-4 із пакету NIST SP 800-22 визначають, чи є число одиниць та нулів та їх коливання у послідовності приблизно таким, яке можна чекати для справді випадкової послідовності. Це також є важливим для геш-функцій. Також, дані тести відповідають вимогам до тестування геш-функцій масштабованості, універсальності та функціональності. Тому дані тести підходять для реалізації в засобі для тестування геш-функцій.

Тести 5-15 підходять більше для тестування блокових та потокових шифрів. Також деякі із цих тестів використовують m-бітове вікно для пошуку конкретного m-бітового шаблону. Якщо шаблон не знайдено, вікно ковзає на одну бітову позицію. Якщо шаблон знайдено, вікно повертається до біта після знайденого шаблону, і пошук відновлюється. Тому тести 5-15 із пакету NIST SP 800-22 не підходять для реалізації в засобі для тестування геш-функцій.

Тести Diehard складаються з 12 статистичних тестів, які застосовуються незалежно від потоку 32-бітових цілих чисел. Ці цілі числа передаються через тести нижче, і кожен повертає значення P, яке має бути рівномірно розподілене між 0 і 1. Тести слід повторювати кілька разів з різними цілими наборами, щоб продемонструвати надійність результатів.

Величина P – ймовірність того, що ідеальний генератор згенерував послідовність менш випадкову, ніж досліджуваний. Тоді якщо P більше α, то досліджувана послідовність вважається випадковою і навпаки в іншому випадку. Кроки статистичного тестування зображені у вигляді таблиці (табл. 2.1).

Таблиця 2.1 – Кроки тестування

Процес	Ремарка
1. Постановка гіпотези	Послідовність є випадковою
2. Обчислення статистики послідовності	Тестування на рівні бітів
3. Обчислення P	$P \in [0; 1]$
4. Порівняння P з α	α в межах [0,001; 0,01]. Якщо $P < \alpha$, то тест пройдено

Для «якісних» послідовностей ймовірність такої події α вкрай мала (~ 0,001).

Тести 5 та 11 з пакету Diehard задовольняють вимоги з пункту 2.2 та можуть бути практично реалізованими у засобі для тестування геш-функцій. Тест 5 полягає у обчисленні одиниць (Count the 1's) – рахуються поодинокі біти в кожному з наступних або обраних байт. Ці лічильники перетворюються в «букви», і рахуються випадки п'ятибуквенних «слів». Тест 11 це тест послідовностей (Runs Test) – генерується довга послідовність на [0,1). Обчислюються висхідні і низхідні послідовності. Числа повинні задовольняти деякому розподілу.

ENT тести мають інший характер, ніж будь-який з інших описаних пакетів тестування. Пакет описує кілька основних тестів:

- обчислення ентропії;
- chi square;

- розрахунок середнього арифметичного;
- Монте-Карло для обчислення пі;
- послідовний коефіцієнт кореляції.

Значення тесту 4 з пакету тестів ENT не є показовим, тому даний тест можна ігнорувати. Тести 1-3, 5 більше орієнтовані на псевдовипадкові числа, а не геш-функції, відповідно до критеріїв в пункті 2.2. Таким чином, тести 1-5 з пакету тестів ENT є менш корисними, ніж обрані тести з пакету NIST SP 800-22.

Також існує ще один пакет тестів Known-Answer-Tests (KAT) для аналізу геш-функцій. KAT містить в собі три тести.

Тест 1. Тест коротких повідомлень. Мета тесту полягає в тому, щоб перевірити, що геш-функція, яка тестується, має можливість генерувати дайджести для повідомлень довільної довжини.

Тест 2. Тест довгих повідомлень. Тест призначений для перевірки здатності алгоритму обробляти довгі повідомлення належним чином.

Тест 3. Тест екстримально довгих повідомлень. Мета тесту перевірити чи не вироджується значення геш-функції в нулі або одиниці.

Отже, з опису тестів KAT випливає те, що вони підходять для тестування функцій гешування, оскільки це дозволить перевірити здатність алгоритму адекватно гешувати короткі, довгі та екстримально довгі повідомлення, тому їх доцільно реалізувати у засобі для тестування геш-функцій.

Також для тестування геш-функцій доцільно застосувати перевірку за допомогою тестових векторів. Перевірка полягає в тому, щоб перевірити значення виходу алгоритму гешування з еталонним.

Таким чином, було описано мету кожного з тестів із пакетів тестів NIST SP 800-22, DIEHARD, ENT та KAT. Визначено, які із тестів можливо застосувати для перевірки на коректність реалізації геш-функцій.

Хоча можна використовувати вибрані тести із досліджених пакетів, але їх недостатньо, оскільки вони орієнтовані не на геш-функції, тому варто розробити власний метод тестування геш-функцій.

2.4 Узагальнений опис методу тестування геш-функцій

Методом тестування геш-функцій планується досягти високоточного тестування, тому потрібно розробити алгоритм для його дослідження.

Алгоритм методу тестування геш-функцій буде складатися з 4 кроків в такій послідовності:

Крок 1. Виконати процеси *initialize*(·), *process*(·) та *finalize*(·). На початку виконання методу тестування необхідно ініціалізувати геш-функцію процедурою *initialize*(·). Для коректного тестування геш-функції методу тестування повинна бути доступна функція *process*(·). За відсутності такого методу геш-функціюї необхідно створити фасад, що матиме метод *process*(·). За допомогою *finalize*(·) отримати результат для подальшого його тестування. Виконання цих трьох процесів є обов'язковим, оскільки без них буде не можливе подальше тестування.

Крок 2. Виконати тестування за допомогою тестових векторів коректності реалізації від розробників функції гешування. Необхідно також провести тестування функції гешування за допомогою векторів, які надають розробники. Це потрібно для того, щоб перевірити чи не припустилися розробники помилки як у самій геш-функції, так і в тестових векторах.

Крок 3. Здійснити тестування за допомогою тестів з пакетів NIST SP 800-22, DIEHARD, ENT та KAT, обраних в підрозділі 2.3. Після тестування за допомогою векторів, які пропонують розробники, з метою виявлення помилок, потрібно протестувати статистичні характеристики геш-функції за допомогою обраних тестів із досліджуваних стандартів на коректність їх реалізації.

Крок 4. Здійснити диференційний криптоаналіз. Потрібно дослідити за допомогою операції XOR, як зміни якихось бітів, а не вхідних, впливає на той чи інший біт або біти.

Відповідно до реалізації цього методу буде присвячено наступний розділ.

2.5 Висновки з розділу

У другому розділі магістерської кваліфікаційної роботи було проаналізовано метод тестування геш-функцій. Математично описано процес тестування функцій гешування. Визначено вимоги до тестування. Обґрунтовано вибір тестів з наборів тестів NIST SP 800-22, DIEHARD, ENT та KAT.

Отже, проведення математичного опису процесу тестування показало, що тестування є дискретним, тести не залежать один від одного, тестування однієї геш-функції не впливає на тестування іншої. Таким чином, тести повинні бути універсальними.

Аналіз вимог до тестування визначив, що при реалізації методу та засобу слід дотриматися вимог масштабованості, функціональності, універсальності, надійності, зручності у використанні, ефективності та супроводжуваності.

Розглянути та проаналізовано тести з пакетів NIST SP 800-22, DIEHARD, ENT та KAT. Їх аналіз показав, що деякі із запронованих тестів можна використати для тестування коректності реалізації геш-функцій. Але даних тестів недостатньо, оскільки їх кількість замала і вони не націлені на функції гешування.

Тому варто розробити метод та засіб для криптоаналізу алгоритмів гешування.

3 ЗАСІБ ТЕСТУВАННЯ ГЕШ-ФУНКЦІЙ

3.1 Узагальнений алгоритм тестування

Алгоритм засобу тестування геш-функцій складається з таких етапів:

- перевірка та виконання процесів *initialize*, *process* та *finalize;*
- перевірка, якщо таких процесів немає, то подальше тестування не виконується. Необхідно виконати обгортку *initialize*, *process* та *finalize* для функції гешування;
- тестування за допомогою тестових векторів, що надають розробники функції, для виявлення можливих помилок;
- перевірка на те, чи не було виявлено помилок під час тестування за допомогою тестових векторів. Якщо такі помилки виявлено, то тестування закінчується негативно;
- тестування за допомогою обраних тестів з пакетів NIST SP 800-22, DIEHARD, ENT та KAT у підрозділі 2.3 для дослідження та аналізу статистичних властивостей геш-функції;
- перевірка на те, чи геш-функція пройшла тестування за допомогою обраних тестів з даних пакетів. Якщо тести успішно пройдено, то алгоритм переходить до наступного етапу тестування, якщо ні, то тестування закінчується з негативним результатом;
- тестування методом диференційного криптоаналізу;
- перевірка на те, чи диференційний криптоаналіз не виявив недоліки геш-функції. Якщо всі етапи тестування успішно пройдені, то користувач отримає відповідне позитивне повідомлення. Якщо ж один із тестів не пройдено, то – відповідне повідомлення про негативне закінчення тестування.

Узагальнений алгоритм засобу тестування криптографічних геш-функцій представлено на рисунку 3.1.

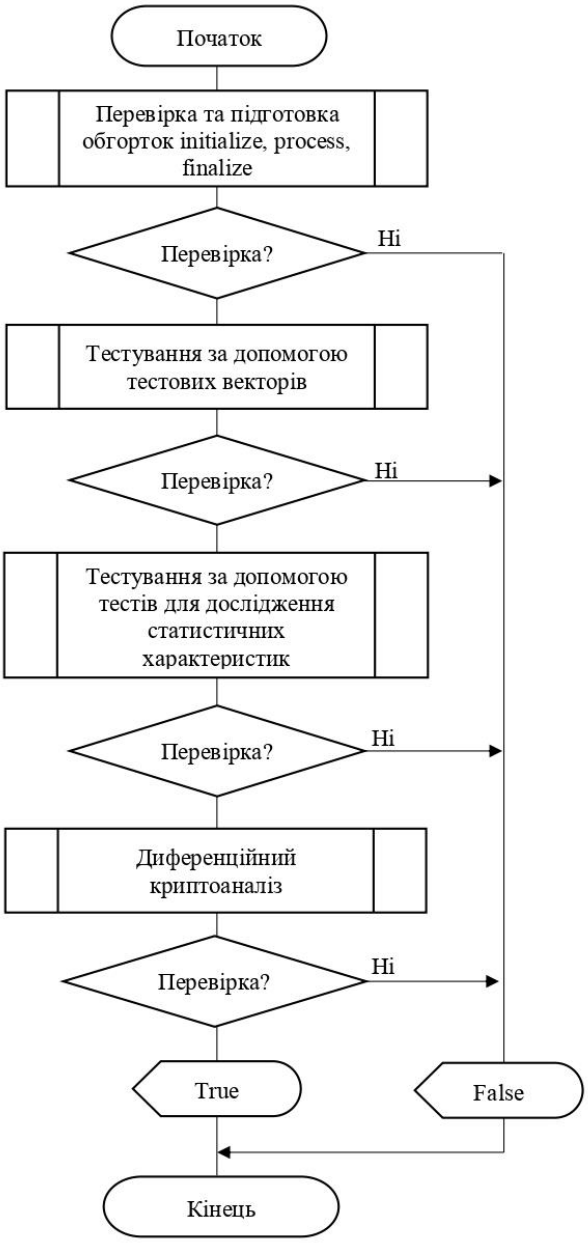


Рисунок 3.1 – Узагальнений алгоритм тестування

Перед безпосередньою реалізацією засобу тестування функцій гешування необхідно розробити алгоритми роботи зазначених процесів тестування.

3.2 Тестування коректності реалізації

Тестування коректності реалізації геш-функції відповідно до тестових векторів, які надають розробники тієї чи іншої геш-функції. Реалізація даного тесту потребує розробки окремої бібліотеки для більшої простоти та зручності використання засобу для тестування, де користувач буде описувати класи вхідних повідомлень та еталонних вихідних геш-значень для певного алгоритму гешування, який йому необхідно протестувати.

Алгоритм тестування коректності реалізації функцій гешування представлено на рисунку 3.2.

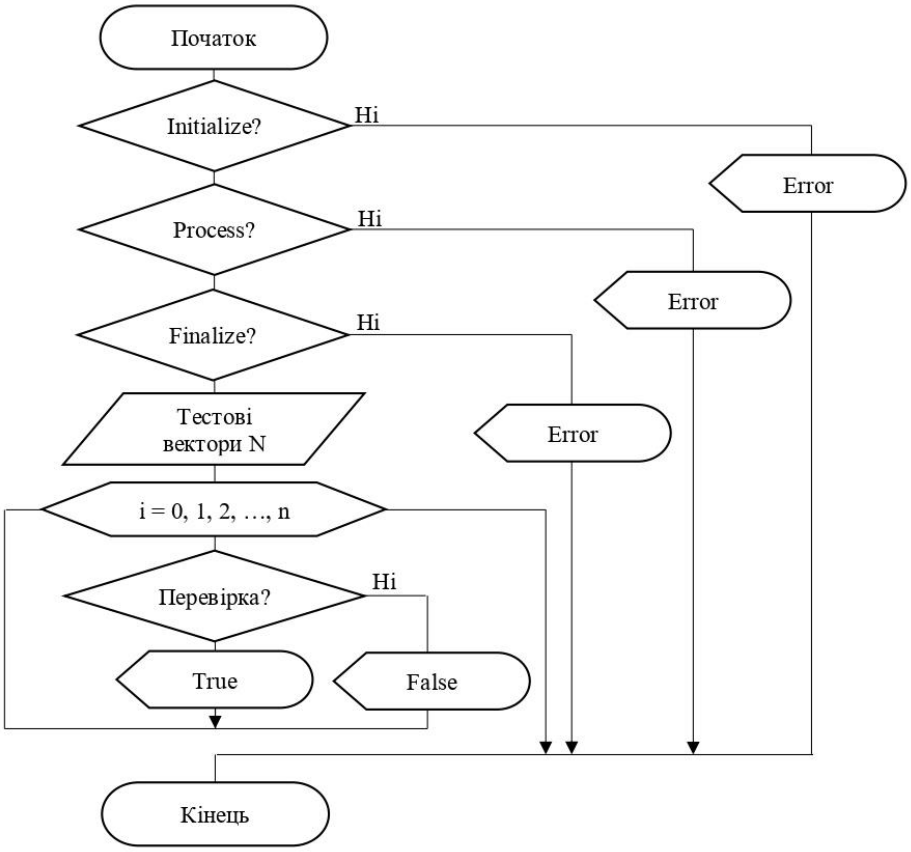


Рисунок 3.2 – Алгоритм тестування коректності реалізації геш-функції

Першим кроком відбувається виконання процесів *initialize*, *process* та *finalize*. Відповідно, якщо даної обгортки для геш-функції немає, то буде виведено відповідне повідомлення про помилку. Далі вводяться певні тестові вектори для перевірки та проводяться тест по кожному із них.

Якщо отримані та еталонні геш-значення збігаються, то це означає, що тестування успішно пройдено. Тому засіб для тестування геш-функцій може переходити до наступного етапу.

3.3 Тестування статистичних властивостей

Алгоритм процесу тестування статистичних властиостей геш-функцій складається з таких етапів:

- процес тестування за допомогою вибраних тестів з пакету NIST SP 800-22;
- перевірка на те, чи пройшов алгоритм гешування дані тести. Якщо даний процес тестування пройдено успішно, то алгоритм перейде до наступного етапу тестування за допомогою тестів з іншого набору тестів;
- процес тестування за допомогою вибраних тестів з набору Diehard;
- відповідна перевірка, аналогічна попередній перевірці;
- процес тестування за допомогою вибраних тестів з набору KAT;
- перевірка на те, чи дані тести не виявили недоліки геш-функції. Якщо всі етапи тестування успішно пройдені, то користувач отримає відповідне позитивне повідомлення. Якщо ж один із тестів не пройдено, то – відповідне повідомлення про негативне закінчення тестування.

Алгоритм процесу тестування статистичних властиостей геш-функцій представлено на рисунку 3.3.

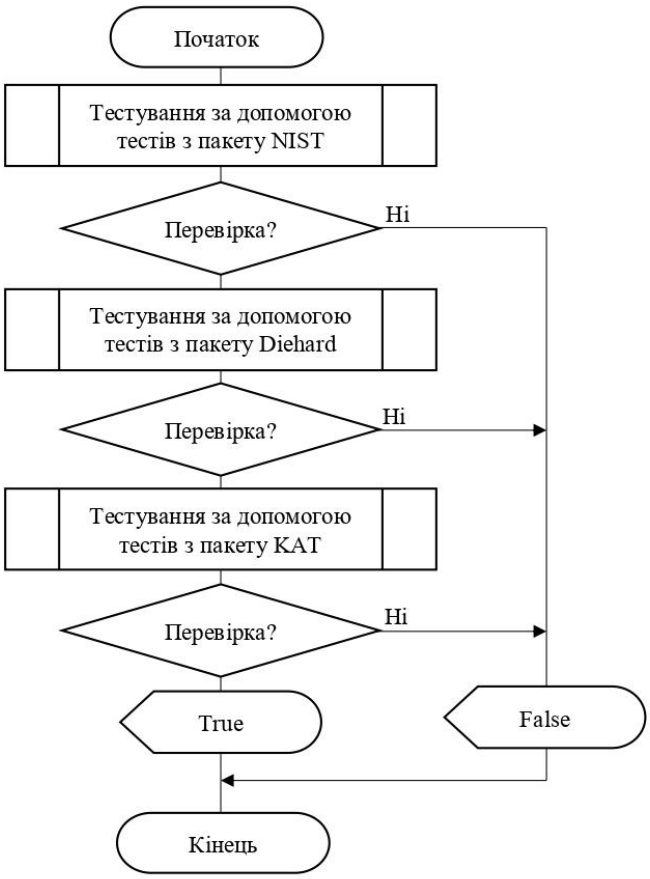


Рисунок 3.3 – Алгоритм тестування статистичних властивостей геш-функції

Далі буде розглянути алгоритми роботи декількох тестів із наборів NIST SP 800-22, DIEHARD та KAT.

Тест перевірки частоти в межах блоку. Алгоритм роботи даного тесту представлений на рисунку 3.4. Основними кроками алгоритму є обчислення одиниць в кожному блоці M та перевірка результату для його інтерпритації у відповідний вигляд. Алгоритми роботи даних кроків наведено в додатку В на рисунку В.1 та В.2 відповідно. Результатом тесту буде *true* або *false* відповідного до того, чи пройшла геш-функція даний тест чи ні.

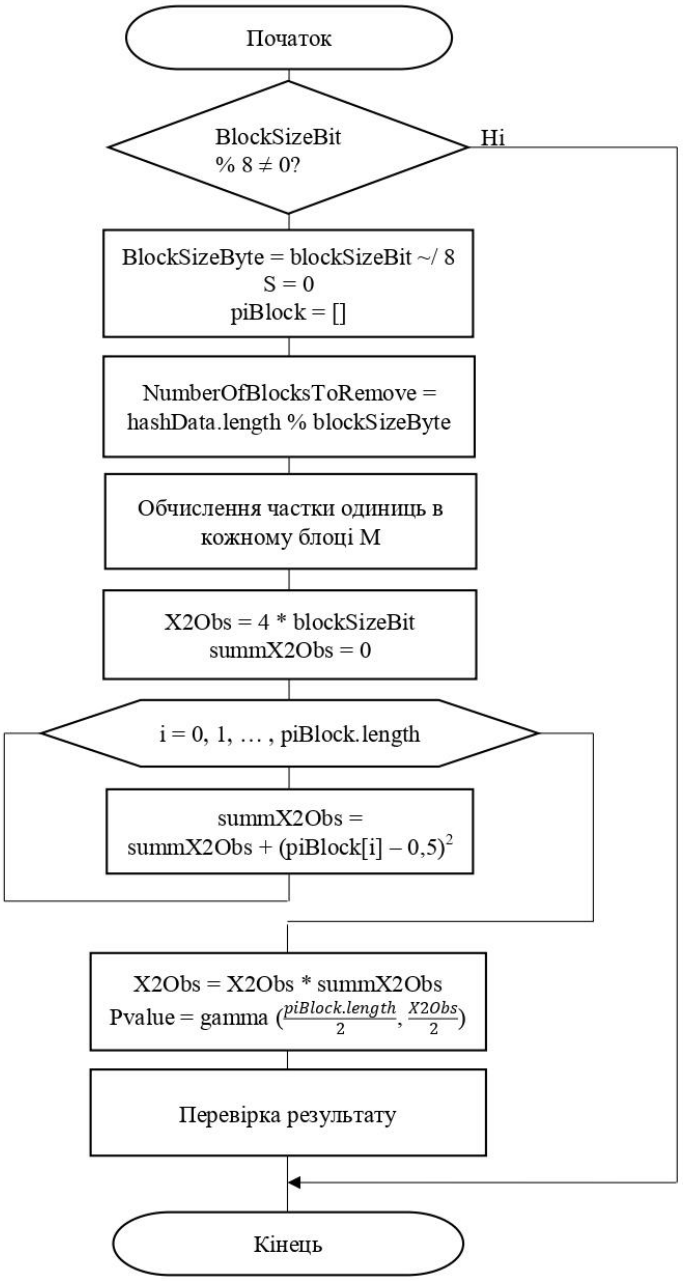


Рисунок 3.4 – Алгоритм роботи тесту перевірки частоти в межах блоку

Далі розроблено алгоритм роботи тесту запусків. Проходження монобітного тесту є основною вимогою для даного тесту, оскільки, якщо геш-фукція не пройшла монобітного тесту, то й і тест запусків не виконується. Якщо

ж монобітний тест успішно пройдено, то далі необхідно обчислити пропорцію π одиниць в вхідній послідовності за формулою $\pi = \sum_j digest_j/n$.

Відповідно, після перевірки монобітного тесту відбувається обчислення статистики та перевірка результату для його інтерпритації у відповідний вигляд. Алгоритми роботи обчислення статистики та перевірка результату наведені на рисунку В.3 та на рисунку В.2 відповідно. Алгоритм роботи монобітного тесту наведено в додатку В на рисунку В.4. Монобітний тест містить в собі процеси перевірки розподілу даних та перевірки результату, які зображені в додатку В на рисунку В.5 та В.2 відповідно.

Алгоритм роботи тесту запусків представлений на рисунку 3.5.

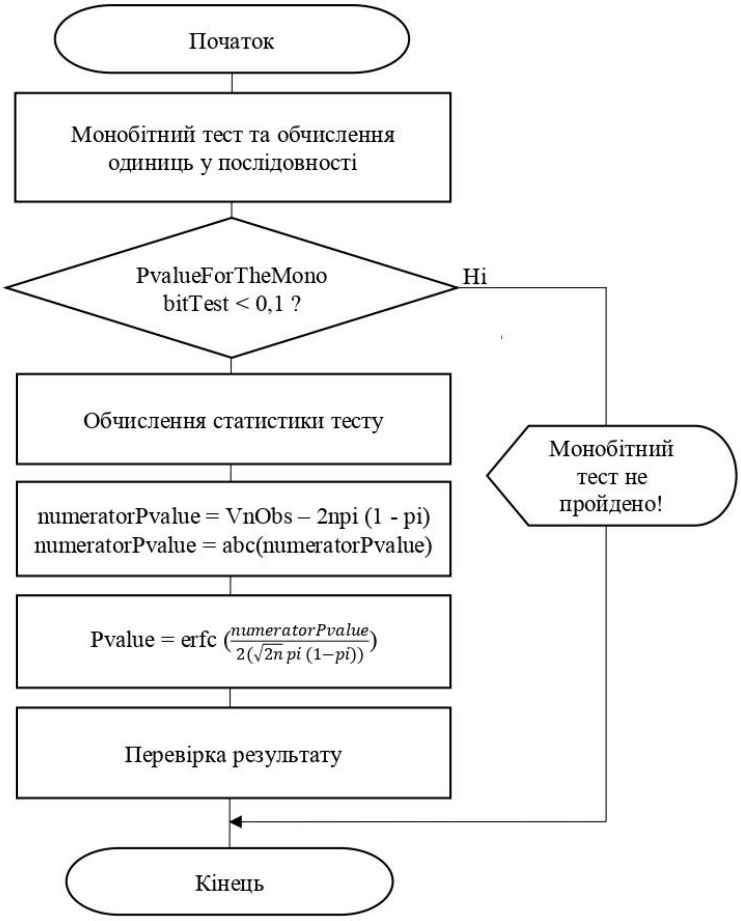


Рисунок 3.5 – Алгоритм роботи тесту запусків

Тест на найдовший пробіг у блоці. Першим кроком є розбиття вхідної послідовності на М-біт. Довжина М-біт визначається таким чином:

- якщо $128 \leq n < 6272$, то М = 8 біт;
- якщо $6272 \leq n < 750000$, то М = 128 біт;
- якщо $750000 \leq n$, то М = 10^4 біт.

Далі за допомогою таблиці 3.1 обраховується v_i – частота найдовшого пробігу одиниць у кожному блоці [23].

Таблиця 3.1 – Відліки для розрахунку частоти v_i

v_i	М = 8	М = 128	М = 10^4
v_0	≤ 1	≤ 4	≤ 10
v_1	= 2	= 5	= 11
v_2	= 3	= 6	= 12
v_3	≥ 4	= 7	= 13
v_4		= 8	= 14
v_5		≥ 9	= 15
v_6			≥ 16

Алгоритм роботи тесту на найдовший пробіг у блоці містить в собі такі процеси:

- визначення довжини кожного блоку. Алгоритм роботи даного процесу наведено в додатку В на рисунку В.6;
- обчислення найдовшої послідовності одиниць. Алгоритм роботи даного процесу наведено в додатку В на рисунку В.7;
- обчислення параметру v_i. Загальний алгоритм роботи даного процесу наведено в додатку В на рисунку В.8. Обчислення v_i в загальному алгоритмі розгалужуються в залежності від розміру блоку, тестом пропонується три варіанти розміру блоку 8, 128 та 10^4. Алгоритми роботи обчислення v_i для всіх трьох варіантів наведено в додатку Б на рисунках В.9, В.10 та В.11 відповідно;
- перевірка результату. Алгоритм роботи даного процесу наведено в додатку В на рисунку В.2.

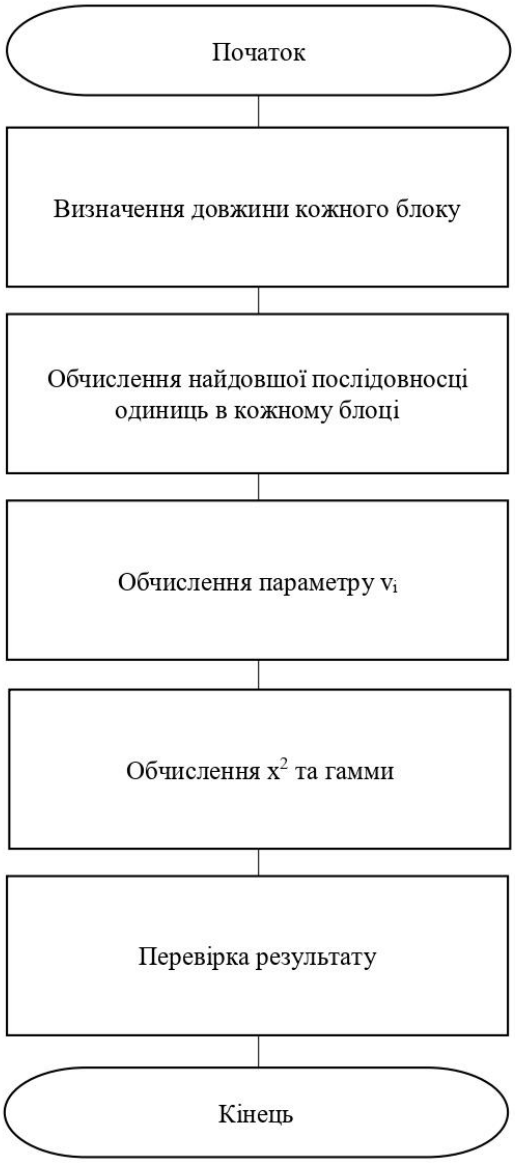


Рисунок 3.6 – Алгоритм роботи тесту на найдовший пробіг у блоці

Отже, було розроблено алгоритм тестування статистичних властивостей геш-функцій та алгоритми роботи кількох основних тестів для дослідження їх статистичних характеристик.

В наступному підрозділі буде розроблено алгоритм автоматизації диференційного криптоаналізу геш-функцій.

3.4 Алгоритм автоматизації диференційного криптоаналізу

Для автоматизації диференційного криптоаналізу геш-функцій пропонується такий алгоритм:

- користувачу пропонується вибрати те, яким чином формуватимуться вхідні дані, відповідно після цього він може самостійно ввести дані для гешування або згенерувати їх автоматично;
- далі користувач повинен вести порядок біта або бітів, які будуть змінюватися. Отримане значення буде використовуватися в подальшій роботі алгоритму за необхідності, в залежності від методу перебору данних;
- наступним кроком відбувається ініціалізація методу для перебору бітів. Саме цей крок і автоматизує диференційний криптоаналіз, оскільки на даному етапі можна підставити будь-який метод для перебору бітів;
- далі користувач повинен ввести порядок біта або бітів вихідного значення, які будуть досліджуватися, якщо це дозволяє метод перебору бітів. Зокрема алгоритм може збирати статистику по усіх бітах вихідного значення;
- наступним етапом виконуються два цикли по бітам для зміни та по бітам для дослідження. Зокрема відбувається гешування та обчислення статистики по кожному бітові. Після чого викликається метод для зміни бітів і цикл повторюється заново;
- останнім кроком є отримання результату.

Алгоритм роботи автоматизованого диференційного криптоаналізу зображено на рисунку 3.7.

Для методу перебору бітів пропонується використовувати коди Грея. Алгоритм роботи кодів Грея зображено на рисунку 3.8.

Коди Грея є таким порядком двійкової системи числення, що два послідовних значення відрізняються лише одним бітом.

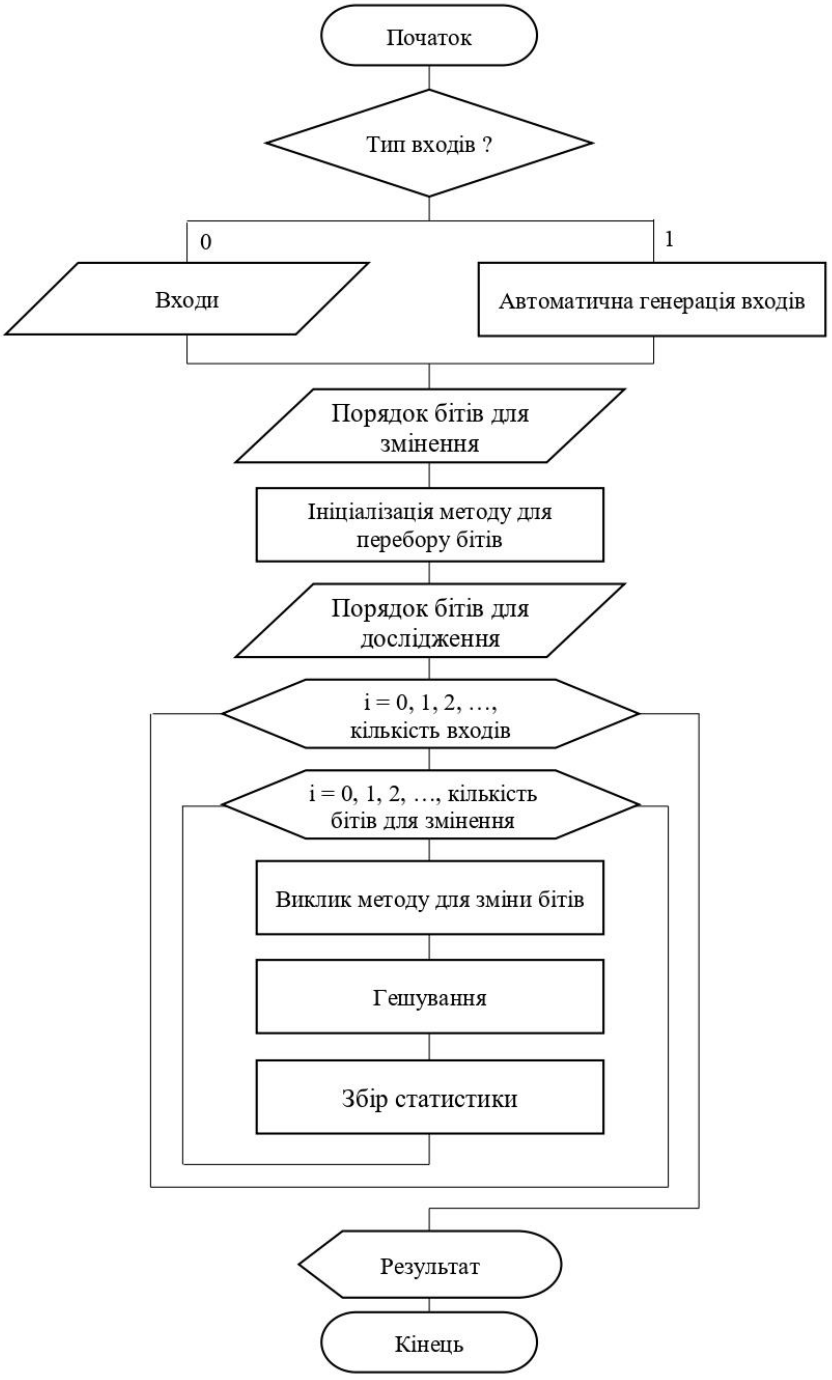


Рисунок 3.7 – Алгоритм роботи автоматизованого диференційного криптоаналізу

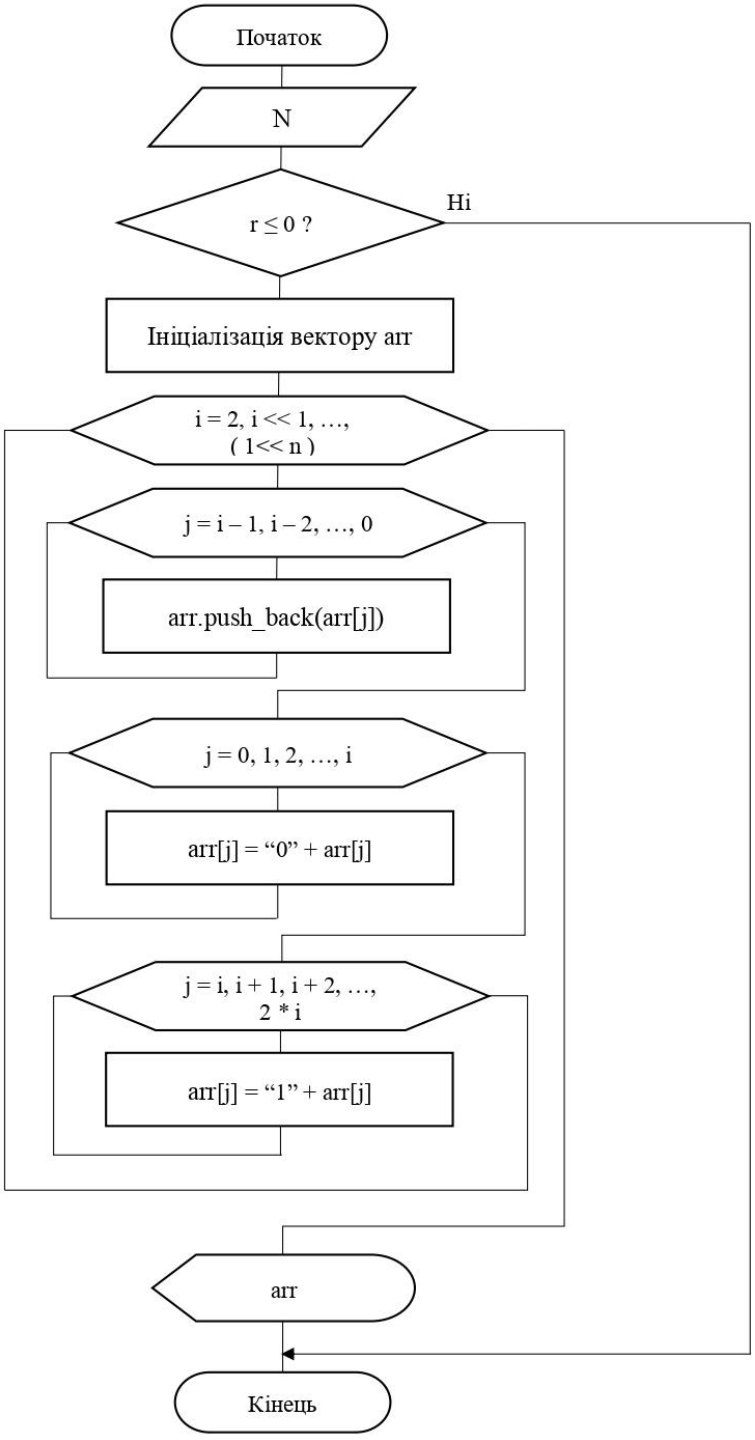


Рисунок 3.8 – Алгоритм роботи кодів Грея

Зокрема, коди Грея широко використовують для запобігання помилковим виходам від електромеханічних перемикачів і для полегшення виправлення помилок у цифрових комунікаціях. Тому варто було б застосувати коди Грея і для дослідження геш-функцій.

Зокрема, пропонується використовувати також Грея подібні коди для диференційного криптоаналізу алгоритмів гешування, таких як:

- збалансований код Грея;
- довгострокові коди Грея;
- монотонні коди Грея;
- одноколінний код Грея;
- код Беккета-Грея.

Наступним пропонується такий метод перебору вхідних бітів, а саме: ініціалізуються вхідні дані довжиною L, яку зазначе користувач, та заповнюються нулями. Далі кожен біт змінюється циклічно на одиницю. Тобто, наприклад, якщо користувач, задасть довжину вхідної послідовності 5, то вхідні дані змінюватимуться таким чином:

- 00000;
- 10000;
- 01000;
- 00100;
- 00010;
- 00001.

Далі кожен сформований вихід із цих вхідних даний подається на функцію автоматизованого диференційного криптоаналізу, де відбувається обчислення статистики. Алгоритм роботи ще одного запропонованого методу перебору вхідних бітів, а саме, почергової зміни кожного розряду нульової послідовності на одиницю, зображено на рисунку 3.9.

Також, пропонується розробити ще один подібний тест для диференційного криптоаналізу, який відрізнятиметься від попереднього тим, що на вхід подаватимуться не нулі, а одиниці, і зміна, відповідно, відбуватиметься

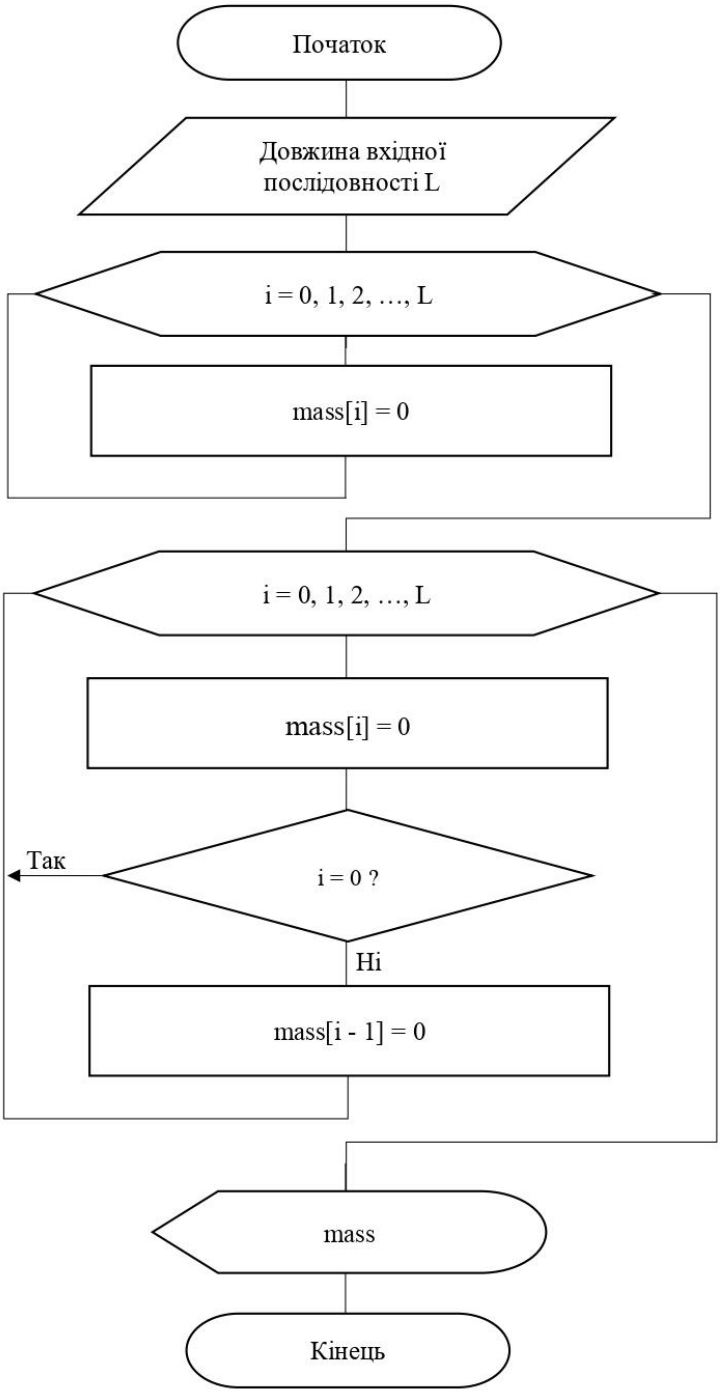


Рисунок 3.9 – Алгоритм роботи почергової зміни кожного розряду нульової послідовності на одиницю

на «0». Тобто, якщо користувач, задасть довжину вхідної послідовності 5, то вхідні дані змінюватимуться таким чином: 11111, 01111, 10111, 11011, 11101, 11110.

Наступними методами для перебору вхідних бітів пропонуються методи зміни k-ї кількості бітів нульової або одниничної послідовності на одиниці або нулі відповідно. Алгоритм роботи почергової зміни k-ї кількості бітів нульової послідовності на одиниці зображено на рисунку 3.10. Таким чином, користувач сам зазначає порядок бітів вхідної послідовності, які будуть змінюватися.

Розроблені вище алгоритми можливих методів перебору вхідних даних були запропоновані як варіанти для диференційного криптоаналізу геш-функцій для отримання більш якісних результатів дослідження коректності реалізації алгоритмів гешування та їх стійкості у розділі 4.

Тому, для вичерпної автоматизації алгоритму диференційного криптоаналізу геш-функцій, пропонується розробити такий метод для перебору вхідних даних, щоб користувач самостійно вказував необхідні йому параметри для дослідження того чи іншого алгоритму гешування, а саме:

- довжину вхідної послідовності N;
- порядок бітів, які необхідно змінювати;
- та, на яке значення необхідно змінити вибраний порядок бітів.

Алгоритм роботи даного методу зображено на рисунку 3.11.

Отже, після розробки засобу тестування геш-функцій варто підбити підсумки з розділу, що й буде зроблено в наступному підрозділі.

3.5 Висновки з розділу

У третьому розділі магістерської кваліфікаційної роботи було проаналізовано засіб тестування геш-функцій. Розроблено узагальнений алгоритм тестування геш-функцій. Описано алгоритм тестування коректності реалізації алгоритмів гешування за допомогою тестових векторів, що надають розробники, з метою виявлення можливих помилок на ранньому етапі. Описано алгоритм тестування статистичних властивостей геш-функцій.

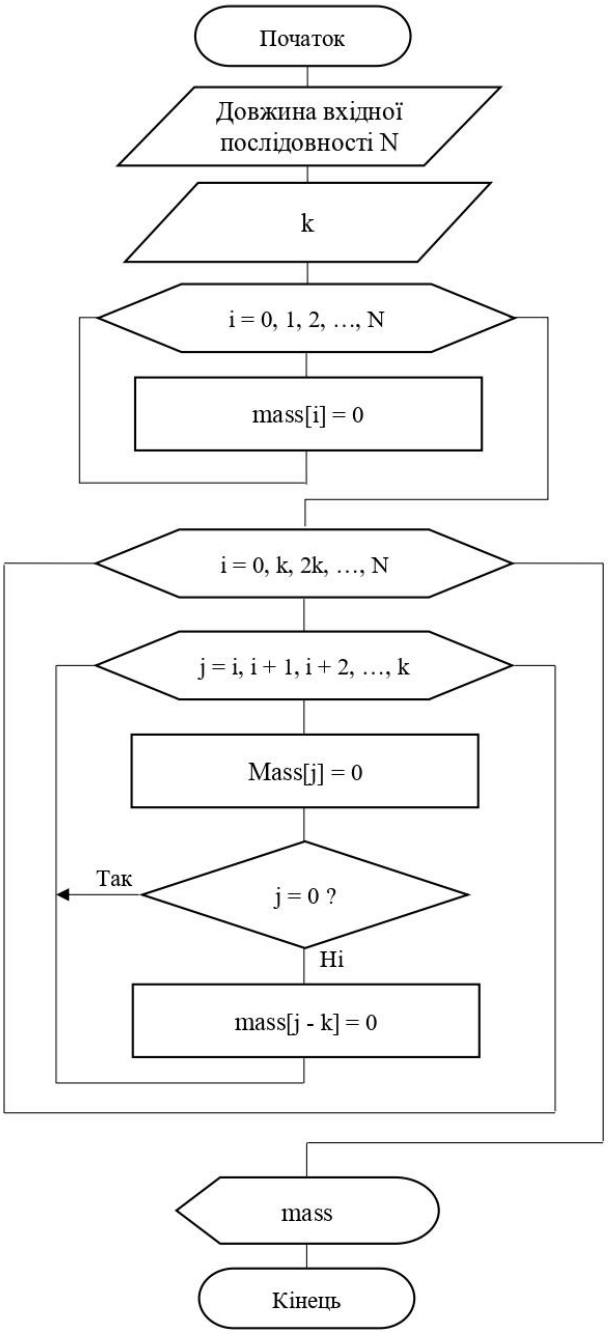


Рисунок 3.10 – Алгоритм роботи почергової зміни k-ї кількості бітів нульової послідовності на одиниці

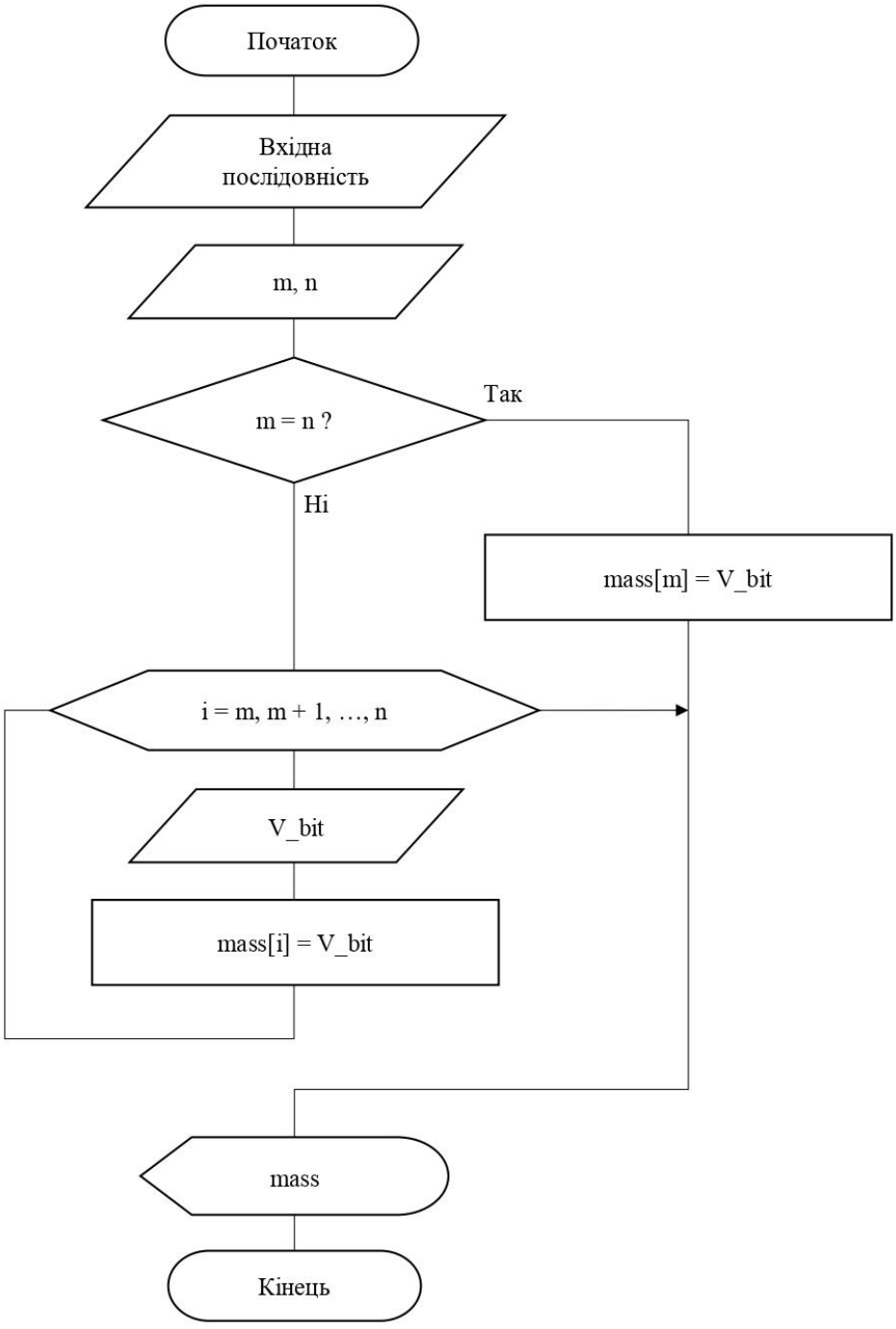


Рисунок 3.11 – Алгоритм роботи зміни певного біта або бітів вхідної послідовності на «1» або «0»

Розроблено алгоритм автоматизації диференційного криптоаналізу функцій гешування.

Отже, розробка узагальненого алгоритму тестування геш-функцій показала, що засіб проводитиме тестування в чотири основних етапи:

- виконання процесів *initialize*, *process* та *finalize*;
- тестування за допомогою тестових векторів, що надають розробники геш-функції;
- тестування функцій гешування за допомогою обраних тестів з пакетів NIST SP 800-22, DIEHARD, ENT та KAT;
- тестування алгоритмів гешування за допомогою диференційного криптоаналізу.

Розробка алгоритму тестування коректності реалізації геш-функцій виявила, що для більшої простоти та зручності використання засобу для тестування необхідно розробити окрему бібліотеку, де користувач буде описувати класи вхідних повідомлень та еталонних вихідних геш-значень для певного алгоритму гешування, який йому необхідно протестувати.

Зокрема, під час розробки алгоритму тестування статистичних властивостей геш-функцій, було описано алгоритми роботи тестів для дослідження цих статистичних властивостей.

Розробка алгоритму автоматизації диференційного криптоаналізу функцій гешування показала, що даний процес автоматизовано в достатній мірі.

Далі необхідно провести експериментальне дослідження засобу криптоаналізу геш-функцій.

4 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЗАСОБУ

4.1 Дослідження коректності засобу для тестування геш-функцій

Розроблений засіб для тестування алгоритмів гешування має цілком зрозуміле візуальне представлення результатів тестів. Під час роботи тесту на екран подаються такі дані:

- назва геш-функції, яка в даний момент часу тестується;
- назва тесту, який виконується;
- час виконання тестів, що, у свою чергу, дає змогу користувачу порівняти швидкість роботи тих чи інших алгоритмів гешування відносно часу, затраченого на виконання певної множини тестів.

Даний процес зображений на рисунку 4.1.

```
C:\Users\vital\AndroidStudioProjects\crypto_dart_tests>flutter test test
00:03 +6: SHA-256 testing:  Runs Test
```

Рисунок 4.1 – Вигляд програми під час виконання тесту

Якщо ж геш-функція пройшла усі тести, то буде виведено відповідне повідомлення (рис. 4.2).

```
C:\Users\vital\AndroidStudioProjects\crypto_dart_tests>flutter test test
00:07 +52: All tests passed!
```

Рисунок 4.2 – Вигляд програми після успішного проходження тестів

Якщо ж алгоритм гешування не проходить якийсь із тестів, тоді на екран виводиться (рис. 4.3):

- назва геш-функції;
- назва тесту, який не пройшов;
- очікуваний результат;
- отриманий результат;
- значення параметра P-value для більшої інформативності;
- повідомлення про те, де в коді знаходиться не пройдений тест.

```
C:\Users\vital\AndroidStudioProjects\crypto_dart_tests>flutter test test
00:29 +48 -1: scrypt testing:  Monobit test [E]
  Expected: <true>
    Actual: <false>
  The actual P-value value is 0.0017500065824132586

  package:test_api                                    expect
  package:flutter_test/src/widget_tester.dart 429:3   expect
  test\hashTests\hashTests.dart 21:9                  run.<fn>.<fn>

00:29 +51 -1: Some tests failed.
```

Рисунок 4.3 – Вигляд програми після не успішного проходження тестів

Далі для тестування коректності розроблених тестів пропонується на вхід тесту тестових векторів алгоритму SHA-256 подати алгоритм SHA-512. Очікується, що геш-функція SHA-512 не пройде тест тестових векторів алгоритму SHA-256 (рис 4.4).

```
00:11 +1 -1: SHA-512 testing:  Test Vectors [E]
  Expected: <true>
    Actual: <false>
  The actual digest value is 'empty'[207, 131, 225, 53, 126, 239, 184, 189, 241, 84, 40, 80, 214, 109, 128, 7, 214, 32, 228, 5, 11, 87, 21, 220,
08, 209, 60, 93, 133, 242, 176, 255, 131, 24, 210, 135, 126, 236, 47, 99, 185, 49, 189, 71, 65, 122, 129, 165, 56, 50, 122, 249, 39, 218, 62]
 152, 252, 28, 20, 154, 251, 244, 200, 153, 111, 185, 36, 39, 174, 65, 228, 100, 155, 147, 76, 164, 149, 153, 27, 120, 82, 184, 85]

  package:test_api                                    expect
  package:flutter_test/src/widget_tester.dart 429:3   expect
  test\hashTests\hashTests.dart 141:9                 run.<fn>.<fn>
```

Рисунок 4.4 – Вигляд результату тесту тестових векторів алгоритму SHA-256 для SHA-512

Отже, очікуваний результат підтвердився, SHA-512 не пройшов тест тестових векторів алгоритму SHA-256.

Далі на вхід тестів подається геш-функція власної розробки, яка на своєму виході дає тільки одні нулі. Очікується, що така геш-функція не пройде монобітний тест (рис. 4.5), а отже, і жоден із інших тестів, окрім тесту за допомогою тестових векторів, адже, якщо геш-функція не проходить монобітний тест, то втрачається смисл проходження наступних тестів.

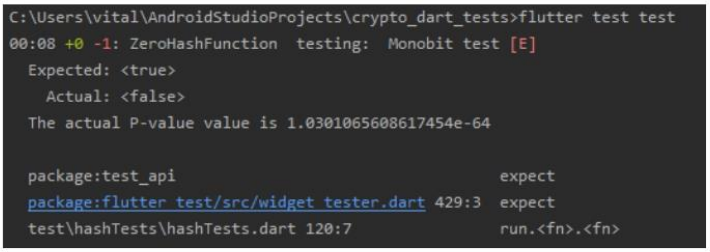


Рисунок 4.5 – Вигляд результату монобітного тесту для нульової геш-функції

Отже, очікуваний результат підтвердився. Далі пропонується подати на вхід тестів геш-функції MD2 та MD5. Очікується, що дані алгоритми гешування покажуть низькі показники стійкості (рис. 4.6).

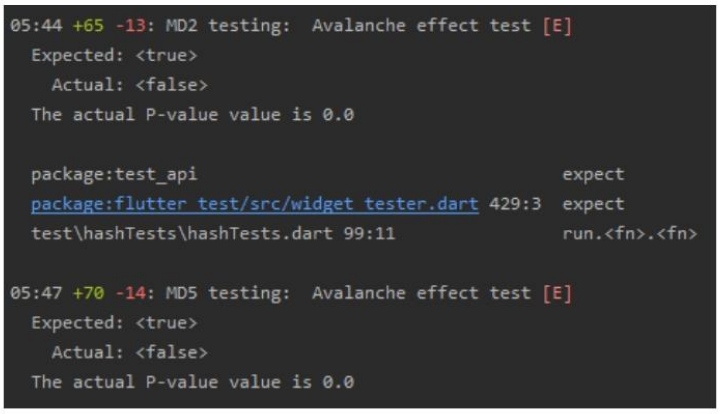


Рисунок 4.6 – Вигляд результату тестування алгоритмів MD2 та MD5

Як видно з рисунку 4.6, очікуваний результат підтвердився. Наступним кроком буде подання на вхід тестів алгоритму гешування Keccak-512. Очікуваним результатом є те, що даний алгоритм пройде усі тести (рис 4.7).

Рисунок 4.7 – Вигляд результату тестування алгоритмів Keccak-512

Отже, тестування коректності розроблених тестів показало, що тести працюють коректно, підтвердилися очікувані результати та виводилась вся необхідна інформація. Під час роботи з програмним застосунком помилок, які передбачалися і опрацьовувались, не було виявлено.

4.2 Дослідження коректності реалізації алгоритмів гешування

Для дослідження коректності реалізації алгоритмів гешування необхідно для кожного запропонованого алгоритму «прогнати» усі тести за допоиогою засобу для тестування. Результати дослідження геш-функцій наведені у таблиці 4.1. Для дослідження пропонується взяти наступні алгоритми гешування: геш-функції сімейства SHA-2, SHA-3, легкі геш-функції сімейства ARMADILO-80, які побудовані на конструкції Merkle-Damgard, H-PRESENT-128, TWISH-128, які побудовані на основі блокових шифрів, PHOTON-80, PHOTON-128, PHOTON-160, PHOTON-224, PHOTON-256, QUARK-136, QUARK-176, QUARK-256, які побудовані на губчастій конструкції, та власна розроблена геш-функція, яка генеруватиме звичайну послідовність 0 та 1.

Таблиця 4.1 – Результати тестування геш-функцій

Геш-функція \ Тест	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
SHA-224	5	5	5	5	5	0	5	5	5	3	3	3	3	5	5	3	2	3
SHA-256	3	5	5	5	5	0	5	5	5	3	4	3	4	5	5	3	4	3
SHA-384	5	5	5	5	5	0	5	5	5	4	4	4	4	5	5	4	4	4
SHA-512	5	5	5	5	5	0	5	5	5	5	5	5	5	5	5	5	5	5
SHA3-224	5	5	5	5	5	0	5	5	5	4	3	2	3	5	5	3	4	4
SHA3-256	5	5	5	5	5	0	5	5	5	4	4	3	4	5	5	4	5	5
SHA3-384	5	5	5	5	5	0	5	5	5	5	5	5	5	5	5	5	4	4
SHA3-512	5	5	5	5	5	1	5	5	5	5	5	5	5	5	5	5	5	5
ARMADILO-80	4	4	4	4	4	0	5	5	5	0	0	0	0	1	0	0	0	0
ARMADILO-128	4	4	4	4	4	0	5	5	5	0	2	2	1	3	1	1	0	0
ARMADILO-160	4	4	4	4	4	0	5	5	5	0	0	1	1	3	0	1	0	0
ARMADILO-192	5	5	5	5	5	0	5	5	5	0	0	2	2	4	0	0	0	0
ARMADILO-256	5	5	5	5	5	0	5	5	5	2	1	3	2	5	2	2	3	3
H-PRESENT-128	4	4	4	4	4	0	5	5	5	1	1	1	3	4	2	2	1	1
TWISH-128	4	4	4	4	4	0	5	5	5	1	1	1	3	4	2	2	1	1

Продовження таблиці 4.1

Геш-функція \ Тест	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
PHOTON-80	4	4	4	4	4	0	5	5	5	0	0	0	0	2	0	0	0	0
PHOTON-128	4	4	4	4	4	0	5	5	5	0	2	2	2	3	1	2	0	0
PHOTON-160	4	4	4	4	4	0	5	5	5	0	0	2	2	3	0	1	0	0
PHOTON-224	4	4	4	4	4	0	5	5	5	2	3	3	2	5	2	1	3	3
PHOTON-256	5	5	5	5	5	0	5	5	5	2	2	3	2	5	3	2	3	5
QUARK-136	4	4	4	4	4	0	5	5	5	0	1	1	1	2	0	1	0	0
QUARK-176	4	4	4	4	4	0	5	5	5	2	1	2	2	4	2	2	3	3
QUARK-256	5	5	5	5	5	0	5	5	5	2	1	3	2	5	2	2	5	5
Тестова геш-функція	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0

Відповідно до таблиці 4.1 для кожної геш-функції тести запускалися 5 разів. Отже, таблиця 4.1 відображає скільки разів із п’яти той чи інший алгоритм гешування пройшов певний тест. Позначення тестів 1, 2, 3, …, 18 відповідають таким тестам:

- 1 – Monobit test;
- 2 – Frequency test withing a 8-bit Block;
- 3 – Frequency test withing a 16-bit Block;
- 4 – Runs test;
- 5 – Test for the Longes Run of Ones in a Block;
- 6 – Avalanche effect test;
- 7 – Vector tests для кожного алгоритму відповідно;
- 8-18 – тести диференційного криптоаналізу.

Оскільки, розроблено 18 тестів та проведено 5 повтерень, за 5 успішно повтерених тестів алгоритм може набрати максимально 5 балів за 1 тест, алгоритм гешування може набрати максимально 90 балів. У таблиці 4.2 наведено підсумок результатів дослідження коректності реалізації та стійкості геш-функцій.

Таблиця 4.2 – Підсумок результатів тестування геш-функцій

Геш-функція	Загальна кількість балів
1. SHA-224	70
2. SHA-256	67
3. SHA-384	78
4. SHA-512	85
5. SHA3-224	71
6. SHA3-256	79
7. SHA3-384	83
8. SHA3-512	86
9. ARMADILO-80	36
10. ARMADILO-128	45
11. ARMADILO-160	41
12. ARMADILO-192	48
13. ARMADILO-256	62
14. H-PRESENT-128	51
15. TWISH-128	51
16. PHOTON-80	37
17. PHOTON-128	47
18. PHOTON-160	43
19. PHOTON-224	59
20. PHOTON-256	67
21. QUARK-136	41
22. QUARK-176	56
23. QUARK-256	62
24. Власна геш-функція	5

Отримані результати у таблиці 4.2 можна інтерпретувати таким чином:

- 50 і менше балів – геш-функція має проблеми з реалізацією та стійкістю;
- 51-70 балів – геш-функція доволі стійка та може мати дещо некоректну реалізацію;
- 70 і більше балів – геш-функція стійка та коректно реалізована.

Для наочності результатів побудовано графік порівняння алгоритмів гешування, по осі X – бали, по осі Y – порядковий номер геш-функції (рис. 4.1).

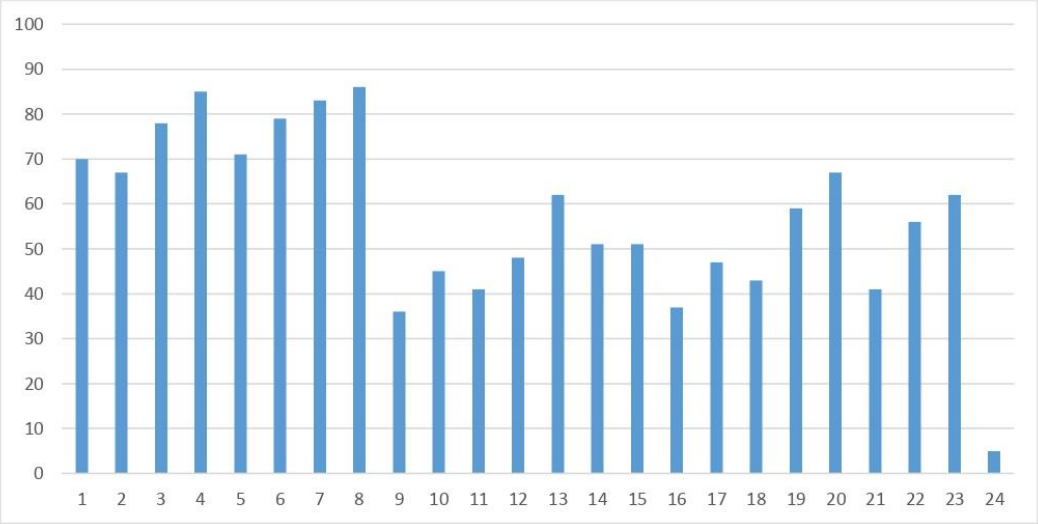


Рисунок 4.8 – Зображення графіку порівняння коректності реалізації та стійкості алгоритмів гешування

Дослідження засобу тестування коректності та стійкості алгоритмів гешування показало, що найстійкішими алгоритмами гешування є SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, SHA3-512.

Функції SHA2-224, SHA3-256, ARMADILO-256, H-PRESENT-128, TWISH-128, PHOTON-224, PHOTON-256, QUARK-176, QUARK-256 є достатньо стійкими, але з високою ймовірність мають проблеми із реалізацією.

Функція гешування власної розробки показала найменший результат, вона пройшла тільки тест на вектори, що показує те, що розроблений засіб працює коректно.

Інші ж алгоритми гешування є нестійкими або мають суттєві проблеми із коректністю реалізації.

5 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

Магістерська кваліфікаційна робота з розробки та дослідження на тему «Метод та засіб криптоаналізу геш-функцій» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки. Цей напрямок є пріоритетним, оскільки результатами розробки можуть користуватися інші споживачі, отримуючи при цьому певний економічний ефект. Але для цього потрібно знайти потенційного інвестора, який би взявся за реалізацію цього проекту і переконати його в економічній доцільності такого кроку.

Для наведеного випадку потрібно виконати такі етапи робіт:

- провести комерційний аудит науково-технічної розробки, тобто встановити її науково-технічний рівень та комерційний потенціал;
- розраховати витрати на здійснення науково-технічної розробки;
- розраховати економічну ефективність науково-технічної розробки у випадку її впровадження і комерціалізації потенційним інвестором і провести обґрунтування економічної доцільності комерціалізації потенційним інвестором.

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Метод та засіб криптоаналізу геш-функцій» є оцінювання науково-

технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в табл. 5.1 [29].

Таблиця 5.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
	Технічні та споживчі властивості продукту значно гірші, ніж в	Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в	Технічні та споживчі властивості продукту значно кращі, ніж в
	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
	Активна конкуренція великих компаній на	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Продовження таблиці 5.1

Бали (за 5-ти бальною шкалою)					
0		1	2	3	4
Практична здійсненність					
	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
0	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
1	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
2	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь- які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно звести до таблиці 5.2.

Таблиця 5.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	4	4	4
2. Ринкові переваги (наявність аналогів)	2	3	2
3. Ринкові переваги (ціна продукту)	2	2	1
4. Ринкові переваги (технічні властивості)	1	2	2
5. Ринкові переваги (експлуатаційні витрати)	2	2	2
6. Ринкові перспективи (розмір ринку)	3	4	3
7. Ринкові перспективи (конкуренція)	3	2	3
8. Практична здійсненність (наявність фахівців)	5	5	5
9. Практична здійсненність (наявність фінансів)	3	4	3
10. Практична здійсненність (необхідність нових матеріалів)	4	5	5
11. Практична здійсненність (термін реалізації)	4	4	4
12. Практична здійсненність (розробка документів)	4	3	4
Сума балів	37	40	38
Середньоарифметична сума балів $СБ_c$	38,3		

За результатами розрахунків, наведених в таблиці 5.2, можна зробити висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки, використовуючи при цьому рекомендації, наведені в табл. 5.3 [29].

Таблиця 5.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ , розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Метод та засіб криптоаналізу геш-функцій» становить 38,3 бала, що, відповідно до таблиці 4.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

5.2 Розрахунок узагальненого коефіцієнта якості розробки

Окрім комерційного аудиту розробки доцільно також розглянути технічний рівень якості розробки, розглянувши її основні технічні показники. Ці показники по-різному впливають на загальну якість проєктної розробки.

Узагальнений коефіцієнт якості ($B_н$) для нового технічного рішення розраховується за формулою [30]:

$$B_н = \sum_{i=1}^{k} \alpha_i \cdot \beta_i \,, \tag{5.1}$$

де k – кількість найбільш важливих технічних показників, які впливають на якість нового технічного рішення;

α_i – коефіцієнт, який враховує питому вагу *i-го* технічного показника в загальній якості розробки. Коефіцієнт α_i визначається експертним шляхом і при цьому має виконуватись умова $\sum_{i=1}^{k} \alpha_i = 1$;

β_i – відносне значення *i-го* технічного показника якості нової розробки.

Відносні значення β_i для різних випадків розраховуються за такими формулами:

- для показників, зростання яких вказує на підвищення в лінійній залежності якості нової розробки:

$$\beta_i = \frac{I_{ні}}{I_{аі}} \,, \tag{5.2}$$

де $I_{ні}$ та $I_{на}$ – чисельні значення конкретного *i*-го технічного показника якості відповідно для нової розробки та аналога;

- для показників, зростання яких вказує на погіршення в лінійній залежності якості нової розробки:

$$\beta_i = \frac{I_{аі}}{I_{ні}} \,; \tag{5.3}$$

Використовуючи наведені залежності можна проаналізувати та порівняти техніко-економічні характеристики аналогу та розробки на основі отриманих наявних та проектних показників, а результати порівняння звести до таблиці 4.4.

Таблиця 5.4 – Порівняння основних параметрів розробки та аналога.

Показники (параметри)	Одиниця вимірю-вання	Аналог	Проектований пристрій	Відношення параметрів нової розробки до аналога	Питома вага показника
функціональність	бал	7	9	1,29	0,25
універсальність	%	60	84	1,4	0,15
зручність	бал	6	8	1,33	0,1
точність	%	77	85	1,1	0,3
надійність	%	75	93	1,24	0,2

Узагальнений коефіцієнт якості ($B_н$) для нового технічного рішення складе:

$$B_H = \sum_{i=1}^{k} \alpha_i \cdot \beta_i = 1{,}29 \cdot 0{,}25 + 1{,}4 \cdot 0{,}15 + 1{,}33 \cdot 0{,}1 + 1{,}1 \cdot 0{,}3 + 1{,}24 \cdot 0{,}2 = 1{,}24.$$

Отже, за технічними параметрами, згідно з узагальненим коефіцієнтом якості розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 1,24 рази.

5.3 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов'язані з проведенням науково-дослідної роботи на тему «Метод та засіб криптоаналізу геш-функцій», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групують за відповідними статтями.

5.3.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників. Витрати на основну заробітну плату дослідників ($З_о$) розраховують у відповідності до посадових окладів працівників, за формулою [29]:

$$З_о = \sum_{i=1}^{k} \frac{M_{mi} \cdot t_i}{T_p}, \quad (5.4)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{mi} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дн.;

T_p – середнє число робочих днів в місяці, T_p=22 дні.

$$З_о = 17450{,}00 \cdot 22 / 22 = 17450{,}00 \text{ грн.}$$

Проведені розрахунки зведено до таблиці 5.5.

Таблиця 5.5 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	17450,00	793,18	22	17450,00
Ст. науковий співробітник	17100,00	777,27	15	11659,09
Інженер-програміст	17150,00	779,55	22	17150,00
Лаборант	6750,00	306,82	12	3681,82
Всього				49940,91

Основна заробітна плата робітників. Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему «Метод та засіб криптоаналізу геш-функцій» розраховують за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i, \quad (5.5)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}}, \quad (5.6)$$

де M_M – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), *Мм*=6700,00 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду;

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок і підприємств до законодавчо встановленого розміру мін. заробітної плати.

T_p – середнє число робочих днів в місяці, приблизно T_p = 22 дн;

$t_{зм}$ – тривалість зміни, год.

$$C_1 = 6700{,}00 \cdot 1{,}10 \cdot 1{,}65 / (22 \cdot 8) = 69{,}09 \text{ грн.}$$

$$З_{р1} = 69{,}09 \cdot 10{,}00 = 690{,}94 \text{ грн.}$$

Величину витрат на основну заробітну плату робітників наведено у таблиці 5.6.

Таблиця 5.6 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Встановлення допоміжного обладнання	10,00	2	1,10	69,09	690,94
Інсталяція програмного забезпечення	6,00	3	1,35	84,80	508,78
Встановлення цифрових обчислювальних систем	5,00	5	1,70	106,78	533,91
Відлагодження програмних модулів криптоаналізу	6,00	4	1,50	94,22	565,31
Підготовка цифрової експериментальної моделі криптоаналізу	9,00	4	1,50	94,22	847,97
Формування бази даних дослідження	15,00	2	1,10	69,09	1036,41
Всього					4183,31

Додаткова заробітна плата дослідників та робітників. Додаткову заробітну плату розраховують як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (5.7)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приблизно 10%.

$$З_{дод} = (49940,91 + 4183,31) \cdot 10 / 100\% = 5412,42 \text{ грн.}$$

5.3.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (5.8)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

$$Зн = (49940,91 + 4183,31 + 5412,42) \cdot 22 / 100\% = 13098,06 \text{ грн.}$$

5.3.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за темою «Метод та засіб криптоаналізу геш-функцій».

Витрати на матеріали ($М$), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$М = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{вj}, \quad (5.9)$$

де $Н_j$ – норма витрат матеріалу j-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу j-го найменування, грн/кг;

K_j – коефіцієнт транспортних витрат, (K_j = *1,1 ... 1,15*);

B_j – маса відходів *j*-го найменування, кг;

$Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг.

$$M_1 = 3,0 \cdot 239,00 \cdot 1,1 - 0 \cdot 0 = 788,70 \text{ грн.}$$

Проведені розрахунки зведено до таблиці 5.7.

Таблиця 5.7 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Папір канцелярський офісний (А4)	239,00	3,0	0	0	788,70
Папір для заміток (А5)	136,00	4,0	0	0	598,40
Начиння канцелярське	201,00	3,0	0	0	663,30
Органайзер офісний	211,00	3,0	0	0	696,30
Картридж для принтера	1068,00	1,0	0	0	1174,80
Диск оптичний (CD-R)	20,50	3,0	0	0	67,65
Диск оптичний (CD-RW)	22,00	1,0	0	0	24,20
FLASH-пам'ять (16 ГБ)	107,00	1,0	0	0	117,70
FLASH-пам'ять (64 ГБ)	329,00	1,0	0	0	361,90
Всього					4492,95

5.3.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі ($K_в$), які використовують при проведенні НДР на тему «Метод та засіб криптоаналізу геш-функцій», розраховують, згідно з їхньою номенклатурою, за формулою:

$$K_в = \sum_{j=1}^{n} H_j \cdot Ц_j \cdot K_j \qquad (5.10)$$

де H_j – кількість комплектуючих *j*-го виду, шт.;

$Ц_j$ – покупна ціна комплектуючих *j*-го виду, грн;

K_j – коефіцієнт транспортних витрат, (K_j = *1,1 ... 1,15*).

$$K_в = 1 \cdot 6750,00 \cdot 1,1 = 7425,00 \text{ грн.}$$

Проведені розрахунки зведено до таблиці 5.8.

Таблиця 5.8 – Витрати на комплектуючі

Найменування комплектуючих	Кількість, шт.	Ціна за штуку, грн	Сума, грн
Оперативна пам'ять: - оперативна пам'ять 32 ГБ - тип оперативної пам'яті DDR4	1	6750,00	7425,00
Графічний адаптер: - тип відеокарти Дискретна - відеокарта nVidia Quadro M1200M - об'єм відеопам'яті 4 ГБ	1	5480,00	6028,00
Всього			13453,00

5.3.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення.

Балансову вартість спецустаткування розраховують за формулою:

$$B_{спец} = \sum_{i=1}^{k} Ц_i \cdot C_{пр.i} \cdot K_i \ , \qquad (5.11)$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, марки, грн;

$C_{пр.i}$ – кількість одиниць устаткування відповідного найменування;

K_i – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, (K_i = *1,10…1,12);*

k – кількість найменувань устаткування.

$$B_{спец} = 41370{,}00 \cdot 1 \cdot 1{,}1 = 45507{,}00 \text{ грн.}$$

Отримані результати зведено до таблиці 4.9.

Таблиця 5.9 – Витрати на придбання спецустаткування по кожному виду

Найменування устаткування	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Ноутбук Dell precision 5520 Процесор: - Intel Core i7-7820HQ (3,9 ГГц) - кількість ядер: 4 ядра	1	41370,00	45507,00
Всього			45507,00

5.3.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховують за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (5.12)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, (K_i = *1,10…1,12);*

k – кількість найменувань програмних засобів.

$$B_{прг} = 0{,}00 \cdot 1 \cdot 1{,}1 = 0{,}00 \text{ грн.}$$

Отримані результати зведено до таблиці 4.10.

Таблиця 5.10 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Прикладне програмне забезпечення розробки	1	0,00	0,00
Середовище розробки: - Android Studio	1	0,00	0,00
Всього			0,00

5.3.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховують з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12} \ , \qquad (5.13)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$Т_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$$А_{обл} = (42400,00 \cdot 1) / (2 \cdot 12) = 1766,67 \text{ грн.}$$

Проведені розрахунки зведено до таблиці 5.11.

Таблиця 5.11 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Програмно-обчислювальний комплекс	42400,00	2	1	1766,67
Місце оператора спеціалізоване	6890,00	5	1	114,83
Офісна оргтехніка	9420,00	4	1	196,25
Дослідницька лабораторія	678000,00	20	1	2825,00
Всього				4902,75

5.3.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію ($В_е$) розраховують за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вті}}{\eta_i}, \qquad (5.14)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_е$ = 6,20 грн;

$K_{вті}$ – коефіцієнт, що враховує використання потужності, $K_{вті} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0,41 \cdot 210,0 \cdot 6,20 \cdot 0,95 / 0,97 = 533,82$ *грн.*

Проведені розрахунки зведено до таблиці 4.12.

Таблиця 5.12 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Програмно-обчислювальний комплекс	0,41	210,0	533,82
Місце оператора спеціалізоване	0,12	200,0	148,80
Офісна оргтехніка	0,45	14,0	39,06
Ноутбук Dell precision 5520	0,05	210,0	65,10
Всього			786,78

5.3.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи на тему «Метод та засіб криптоаналізу геш-функцій» належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» відсутні.

5.3.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховують як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{сп} = (З_о + З_р) \cdot \frac{Н_{сп}}{100\%}, \qquad (5.15)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 30%.

$B_{сп} = (49940,91 + 4183,31) \cdot 30 / 100\% = 16237,27$ *грн.*

5.3.11 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховують як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.16)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», $Н_{ів}$ = 60%.

$$I_в = (49940,91 + 4183,31) \cdot 60 / 100\% = 32474,53 \text{ грн.}$$

5.3.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховують як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.17)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 120%.

$$В_{нзв} = (49940,91 + 4183,31) \cdot 120 / 100\% = 64949,07 \text{ грн.}$$

Витрати на проведення науково-дослідної роботи на тему «Метод та засіб криптоаналізу геш-функцій» розраховують як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \quad (5.18)$$

$$B_{заг} = 49940,91 + 4183,31 + 5412,42 + 13098,06163 + 4492,95 + 13453,00 + 45507,00 + 9751,50 + 4902,75 + 786,78 + 0,00 + 16237,27 + 32474,53 + 64949,07 = 265189,55 \text{ грн.}$$

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховують за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (5.19)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,95.

$$ЗВ = 265189,55 / 0,95 = 279146,90 \text{ грн.}$$

5.4 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження проведені за темою «Метод та засіб криптоаналізу геш-функцій» передбачають комерціалізацію протягом 4-х років реалізації на ринку. В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

ΔN – збільшення кількості споживачів продукту, у періоди часу, що аналізуються, від покращення його певних характеристик;

Показник	1-й рік	2-й рік	3-й рік	4-й рік
Збільшення кількості споживачів, осіб	400	600	1000	700

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, 2800 осіб;

$Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки, 8750,00 грн;

$\pm\Delta Ц_о$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, 630,63 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 4-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховують за формулою [29]:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (5.20)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2022 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту). ρ =30%;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2022 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$$\Delta П_1 = (630,63 \cdot 2800,00 + 9380,63 \cdot 400) \cdot 0,83 \cdot 0,3 \cdot (1\text{-}0,18/100\%) =$$

1126665,24 грн.

Збільшення чистого прибутку 2-го року:

$$\Delta П_2 = (630,63 \cdot 2800,00 + 9380,63 \cdot 1000) \cdot 0,83 \cdot 0,3 \cdot (1\text{-}0,18/100\%) =$$

2275866,85 грн.

Збільшення чистого прибутку 3-го року:

$$\Delta П_3 = (630,63 \cdot 2800,00 + 9380,63 \cdot 2000) \cdot 0,83 \cdot 0,3 \cdot (1\text{-}0,18/100\%) =$$

4191202,86 грн.

Збільшення чистого прибутку 4-го року:

$$\Delta П_4 = (630,63 \cdot 2800,00 + 9380,63 \cdot 2700) \cdot 0,83 \cdot 0,3 \cdot (1\text{-}0,18/100\%) =$$

5531938,07 грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \tag{5.21}$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів;

τ – ставка дисконтування, τ=0,24;

t – період часу (в роках).

$ПП = 1126665,24/(1 + 0,24)^1 + 2275866,85/(1 + 0,24)^2 + 4191202,86/(1 + 0,24)^3 + 5531938,07/(1 + 0,24)^4 = 908601,00 + 1480142,33 + 2198232,51 + 2339863,75 = 6926839,59$ грн.

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \tag{5.22}$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, $k_{інв}$=2,4;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, 279146,90 грн.

$$PV = k_{інв} \cdot ЗВ = 2,4 \cdot 279146,90 = 669952,55 \text{ грн.}$$

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \quad (5.23)$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 6926839,59 грн;

PV – теперішня вартість початкових інвестицій, 669952,55 грн.

$$E_{абс} = ПП - PV = 6926839,59 - 669952,55 = 6256887,04 \text{ грн.}$$

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (5.24)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 6256887,04 грн;

PV – теперішня вартість початкових інвестицій, 669952,55 грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 4 роки.

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+6256887,04/669952,55)^{1/4} = 0,79.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \quad (5.25)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2022 році в Україні d =0,12;

f – показник, що характеризує ризикованість вкладення інвестицій, становитиме 0,35.

$\tau_{мін}$ = 0,12+0,35 = 0,47 < 0,79 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у

впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Метод та засіб криптоаналізу геш-функцій» доцільно.

Період окупності інвестицій $T_{ок}$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \tag{5.26}$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 0,79 = 1,26 р.

$T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

5.5 Висновки до розділу

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Метод та засіб криптоаналізу геш-функцій» становить 38,3 бала, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

При оцінюванні за технічними параметрами, згідно з узагальненим коефіцієнтом якості розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 1,24 рази.

Також термін окупності становить 1,26 року, що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже, можна зробити висновок, що проведення науково-дослідної роботи за темою «Метод та засіб криптоаналізу геш-функцій» є доцільним.

ВИСНОВКИ

В магістерській кваліфікаційній роботі розроблено метод та засіб для криптоаналізу геш-функцій з метою перевірки коректності їх реалізації в розроблюваних застосунках або фрагментах коду третіх сторін та стійкості. Досліджувана проблема є актуальною, оскільки більшість криптографічних бібліотек, що реалізують алгоритми гешування, є безкоштовними, що, у свою чергу, абсолютно не дає ніяких гарантій щодо їх стійкості або коректності реалізації.

В результаті виконання роботи було проведено аналіз джерел, присвячених темі магістерської кваліфікаційної роботи.

В першу чергу було проведено дослідження геш-функцій з погляду «сірої скриньки» та проведено аналіз їх особливостей, що дало змогу визначити основні вимоги до їх криптографічної стійкості. Зокрема, проведено аналіз атак на алгоритми гешування, який показав, що існує різноманітна кількість атак на геш-функції, які, у свою чергу, можна поділити за двома ознаками:

- атаки, які використовують особливості криптографічних перетворень;
- атаки, які не залежать від використовуваних методів криптографічних перетворень.

Після було проаналізовано такий інструмент дослідження стійкості геш-функцій як диференційний криптоаналіз та визначено, що більшість методів диференційного криптоаналізу, адаптованих до блокових шифрів. Використання таких методів для тестування малоресурсних геш-функцій не завжди є доцільним, оскільки геш-функцію варто розглядати не як те, в основі чого лежить якийсь криптографічний примітив, а як окремий повноцінний метод захисту інформації, який потребує специфічного тестування. Але, оскільки алгоритми гешування мають широке застосування, тому доцільніше використовувати змішаний підхід тестування, який полягає, зокрема, не тільки в їх диференційному криптоаналізі, а й в перевірці статистичних властивостей та

корекності реалізації. На основі цього було виконано постановку задач магістерської кваліфікаційної роботи.

Далі було розроблено метод тестування геш-функцій. Зроблено математичний опис процесу тестування, результатом якого є твердження, що тести мають бути універсальними, щоб їх можна було застосувати для будь-якої функції гешування.

Аналіз вимог до криптоаналізу геш-функцій дозволив обгрунтувати вибір наборів тестування із пакетів NIST SP 800-22, DIEHARD, ENT. Аналіз показав, що деякі тести із пакетів NIST SP 800-22, DIEHARD та KAT підходять для дослідження статистичних властивостей геш-функцій. Тести з набору ENT виявилися не такими показовими та менш корисними, ніж обрані тести з пакету NIST SP 800-22.

Далі було розроблено засіб тестування геш-функцій. Завдяки можливості налаштування, розроблений засіб дозволяє користувачеві автоматизувати процес диференційного критоаналізу.

Наступним кроком було експериментальне дослідження засобу. Дослідження корекності реалізації геш-функцій показало, що за допомогою засобу можна дослідити коректність реалізації будь-якого безключового алгоритму гешування. Дослідження коректності розроблених тестів дало змогу провести аналіз роботи засобу тестування функцій гешування. Аналіз показав, що розроблений засіб та реалізовані в ньому тести працюють коректно.

Проведення розрахунків в економічної частини магістерської кваліфікаційної роботи дозволило довести економічну доцільність реалізації засобу криптоаналізу геш-функцій.

Подальший розвиток дослідження планується продовжити у напрямку придатності метода до криптоаналізу ключових геш-функцій та відповідну інтеграцію інструментів лінійного криптоаналізу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Казміревський В. В. Аналіз методів криптоаналізу геш-функцій: матеріали L науково-технічної конференції факультету інформаційних технологій та комп'ютерної інженерії, м. Вінниця, 2021. 4 с. URL: https://conferences. vntu.edu.ua/index.php/all-fitki/all-fitki-2021/paper/view/12543 (дата звернен-ня: 03.09.2022).

2. Казміревський В. В. Дослідження стійкості до лінійного та диференційного криптоаналізу функцій гешування: матеріали LI науково-технічної конференції факультету інформаційних технологій та комп'ютерної інженерії, м. Вінниця, 2022. 4 с. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2022/paper/view/ 15844/13341 (дата звернення: 03.09.2022).

3. Казміревський В. В. Способи покращення системи управління персоналом підприємства. Мотивація персоналу, м. Вінниця, 2022. 4 с. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2021/paper/ view/ 12543 (дата звернення: 03.09.2022).

4. Казміревський В. В. Властивості води та її аномальність, м. Вінниця, 2022. 4 с. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2021/ paper/view/ 12543 (дата звернення: 03.09.2022).

5. Bart Preneel. Analysis and Design of Cryptographic Hash Functions. 2003. 338 p. URL: https://www.esat.kuleuven.be/cosic/publications/thesis-2.pdf (дата звернення: 03.09.2022).

6. Kudin A., Kovalenko B. Differential analysis of hash functions and block ciphers: generalized approach. Ukrainian Scientific Journal of Information Security. 2015. pp. 159-164. URL: http://jrnl.nau.edu.ua/index.php/ Infosecurity/article/view/8734 (дата звернення: 03.09.2022).

7. Лужецький В. А., Баришев Ю. В. Конструкції гешування стійкі до мультиколізій. Наукові праці ВНТУ. – 2010. №1. – 8 с. URL: https://praci.vntu.edu.ua/index.php/praci/article/view/191/189. (дата звернен-ня: 03.09.2022).

8. NIST computer security resource center: Hash Functions SHA-3 Project. 2020. URL: https://csrc.nist.gov/projects/hash-functions/sha-3-project (дата звернення: 03.09.2022).

9. Patrick Nohe. Re-Hashed: The Difference Between SHA-1, SHA-2 and SHA-256 Hash Algorithms. Hashedout by the ssl store. 2018. 11 p. URL: https://www.thesslstore.com/blog/difference-sha-1-sha-2-sha256-hash-algorithms/ (дата звернення: 03.09.2022).

10. Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, Ronny Van Keer. The sponge and duplex constructions. URL: https://keccak.team/sponge_ duplex.html (дата звернення: 03.09.2022).

11. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Sponge Functions. Presented at ECRYPT Hash Workshop. 2007. 22 p.. URL: http://sponge.noekeon.org/ SpongeFunctions.pdf (дата звернення: 03.09.2022).

12. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Keccak specifications. SHA-3 Algorithm Submission. 2008. 14 p.. URL: https://keccak.team/files/ Keccak-submission-3.pdf (дата звернення: 03.09.2022).

13. Hans Dobbertin, Antoon Bosselaers, Bart Preneel. RIPEMD-160: A Strengthened Version of RIPEMD // Katholieke Universiteit Leuven, ESAT-COSIC K. Belgium. pp. 71-82. URL: https://core.ac.uk/download/pdf/ 191296638.pdf (дата звернення: 03.09.2022).

14. Harshvardhan Tiwari. Merkle-Damgård Construction Method and Alternatives: A Review. Centre for Incubation, Innovation, Research and Consultancy (CIIRC) Jyothy Institute of Technology. 2017. pp. 284-304. URL: https://sites.google.com/site/sashadkeem1502/home/mobilni-os (дата звернення: 03.09.2022).

15. Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Elaine Barker, Stefan Leigh, Mark Levenson, Mark Vangel, David Banks, Alan Heckert, James Dray, San Vo. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. NIST Special Publication 800-22 Revision 1a. 2010. 131

p.. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/ SP/nistspecialpublication800-22r1a.pdf (дата звернення: 03.09.2022).

16. Gaetan Leurent. Improved Differential-Linear Cryptanalysis of 7-round Chaskey with Partitioning. Advances in Cryptology – EUROCRYPT. 2016. pp. 344-371. URL: https://www.iacr.org/archive/eurocrypt2016/96650217/ 96650217.pdf (дата звернення: 03.09.2022)

17. Susila Windarta, Kalamullah Ramli, Bernardi Pranggono, Bernardi Pranggono. Lightweight Cryptographic Hash Functions: Design Trends, Comparative Study, and Future Directions. IEEE Access. 2022. pp. 82272-82294. URL: https://www.researchgate.net/publication/362502388_Lightweight_Cryptographic_Hash_Functions_Design_Trends_Comparative_Study_and_Future_Directions (дата звернення: 03.09.2022).

18. Howard M. Heys. A Tutorial on Linear and Differential Cryptanalysis. Cryptologia. 2001. pp. 362-395. URL: https://www.researchgate.net/ publication/2362831_A_Tutorial_on_Linear_and_Differential_Cryptanalysis (дата звернення: 03.09.2022).

19. Deniz Toz. Cryptanalysis of Hash Functions. KU Leuven – Faculty of Engineering Science Kasteelpark Arenberg 10. 2013. 132 p. URL: https://www.esat.kuleuven.be/cosic/publications/thesis-223.pdf (дата звернення: 03.09.2022).

20. Matsui M. Linear Cryptoanalysis Method for DES Cipher. In Helleseth. 2001. pp. 386–397. URL: https://link.springer.com/content/pdf/10.1007/3-540-48285-7_33.pdf (дата звернення: 03.09.2022).

21. C. Krishna Kumar, Dr. C. Suyambulingom. Cryptographic of high Security Hash Functions. International Journal of Engineering Research & Technology (IJERT). 2012. 7 p. URL: https://www.ijert.org/research/cryptographic-of-high-security-hash-functions-IJERTV1IS3074.pdf (дата звернення: 03.09.2022).

22. Dmitry Khovratovich. Cryptanalysis of Hash Functions with Structures. SAC 2009: Selected Areas in Cryptography. 2009. pp. 108-125. URL:

https://link.springer.com/chapter/10.1007/978-3-642-05445-7_7 (дата звернення: 03.09.2022).

23. Florian Mendel. Analysis of Cryptographic Hash Function. Graz University of Technology. 2012. 141 p. URL: https://diglib.tugraz.at/download.php?id=576 a7a85b2dc5&location=browse (дата звернення: 03.09.2022).

24. Thomsen S. S. Cryptographic Hash Functions. Technical University of Denmark. 2009. 174 p. URL: https://backend.orbit.dtu.dk/ws/files/5025771/ sst_thesis_v1.0.pdf (дата звернення: 03.09.2022).

25. Bert Preneel, Rene Govaerts, Jogs Vandewalle. Differential cryptanalysis of hash functions based on block ciphers. Proceedings of the 1st ACM conference on Computer and communications security. 1993. pp. 183–188. URL: https://dl.acm.org/doi/pdf/10.1145/168588.168611 (дата звернення: 03.09.2022).

26. AndrewRukhin, JuanSoto, JamesNechvatal, Miles Smid, ElaineBarker, Stefan Leigh, MarkLevenson, Mark Vangel, DavidBanks, AlanHeckert, JamesDray, SanVo. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. revision 1a. Inst. Stand. Technol. Spec. Publ. 800-22rev1a, 2010. 131 p. URL: https://nvlpubs.nist.gov/nistpubs/ Legacy/SP/nistspecialpublication800-22r1a. pdf (дата звернення: 03.09.2022).

27. Robert G. Brown. Dieharder. A Random Number Test Suite. Duke University Physics Department Durham, NC 27708-0305 Copyright Robert G. Brown, 2021. URL: https://webhome.phy.duke.edu/~rgb/General/dieharder.php (звернення: 03.09.2022).

28. Description of Known Answer Test (KAT) and Monte Carlo Test (MCT) for SHA-3 Candidate Algorithm Submissions. URL: https://www.pdffiller.com/ jsfillerdesk19/?requestHash=8de2ef46b36f9282af6dc735e28f38b5e49bd4e061b39a2 de40285f31349f802&projectId=734014315&loader=tips#a58e8ca7cf224468b10bc3d fe10dc884 (дата звернення: 03.09.2022).

29. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

30. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа – Вінниця : ВНТУ, 2016. – 113 с.

ДОДАТКИ

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

УЗГОДЖЕНО
Директор ТОВ ElephantsLab
_____ А.І. Білоконь
«15» вересня 2022р.

ЗАТВЕРДЖУЮ
Завідувач кафедри ЗІ
д. т. н., професор
В.А. Лужецький
«15» вересня 2022р.

ТЕХНІЧНЕ ЗАВДАННЯ
до магістерської кваліфікаційної роботи на тему
«Метод та засіб криптоаналізу геш-функцій»
08-20.МКР.002.00.000 ТЗ

Розробив студент групи БС-21м
_____ В.В. Казміревський

Керівник МКР
к. т. н., доцент каф. ЗІ
_____ Ю.В. Баришев

Вінниця 2022

1 Назва і область використання

Метод та засіб криптоаналізу геш-функцій.
Криптоаналіз малоресурсних геш-функцій на мові Dart.

2 Основа для розробки

Розробка виконується на основі наказу ректора ВНТУ №203 від 14 вересня 2022 р. "Про затвердження тем магістерських кваліфікаційних робіт (проектів) для студентів денної форми навчання в 2022/2023 н. р."

3 Мета та призначення розробки

Збільшення стійкості реалізації програмних засобів, що використовують криптографічні бібліотеки на мові Dart. Перевірка коректності реалізації методів гешування в розроблюваних застосунках або фрагментах коду третіх сторін.

4 Джерела розробки

1. Баришев Ю. В., В. А. Лужецький. Методи та засоби швидкого багатоканального гешування даних в комп'ютерних системах : монографія за заг. ред. В. А. Лужецького. Вінниця, ВНТУ, 2016. 144 с.
2. Антон Кудін, Богдан Коваленко. Диференційний аналіз функцій хешування та блокових шифрів: узагальнений підхід. URL: http://jrnl.nau.edu.ua/index.php/Infosecurity/article/view/8734 (дата звернення: 03.09.2022).
3. NIST. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800- 22r1a.pdf (дата звернення: 03.09.2022).
4. Susila Windarta, Kalamullah Ramli, Bernardi Pranggono, Bernardi Pranggono. Lightweight Cryptographic Hash Functions: Design Trends, Comparative Study, and Future Directions. URL: https://www.researchgate.net/publication/362502388_Lightweight_Cryptographic_Hash_Functions_Design_Trends_Comparative_Study_and_Future_Directions (дата звернення: 03.09.2022).

5 Вимоги до програми

5.1 Вимоги до функціональних характеристик:
- перевірка на коректність реалізації геш-функцій, зокрема, малоресурсних;
- критерії перевірки: лавинний ефект, збалансованість та рівномірність закону розподілу;
- диференційний криптоаналіз функцій гешування;
- реалізація релевантних тестів з наборів DIEHARD, ENT, NIST;
- проведення розробки на мові Dart.

5.2 Вимоги до видів тестування:
- позитивні тести;
- негативні тести;

- тести «чорної скриньки».

6 Стадії та етапи розробки

Етап	Зміст	Початок	Закінчення	Результат
1	Аналіз завдання	01.09.2022	04.09.2022	Вступ, технічне завдання
2	Аналіз джерел за напрямком МКР	05.09.2022	15.09.2022	Основні тези
3	Розробка структури методу та засобу тестування геш-функцій	16.09.2022	26.09.2022	Структура засобу тестування геш-функцій
4	Розробка алгоритму засобу тестування геш-функцій	27.09.2022	30.10.2022	Алгоритм засобу тестування геш-функцій
5	Аналіз результатів тестування, висновки	01.11.2022	24.11.2022	Результати тестування, висновки
6	Оформлення пояснювальної записки	25.11.2022	02.12.2022	Пояснювальна записка

7 Порядок контролю та прийому

7.1 До прийому і захисту магістерської кваліфікаційної роботи подається:
- пояснювальна записка до МКР;
- набір автоматизованих тестів для дослідження методів гешування;
- графічні матеріали.

7.2 Рубіжний контроль керівника_____

7.3 Попередній захист на кафедрі _ _____

7.4 Захист на ЕК _____

ЗАТВЕРДЖУЮ
Директор ТОВ ElephantsLab

_____ А. І. Білоконь
«19» грудня 2022 р.

АКТ
про впровадження результатів магістерської кваліфікаційної роботи
Казміревського Віталія

Комісія у складі: голова комісії – директор Білоконь А. І. – склала цей акт про те, що у ТОВ ElephantsLab впроваджено результати магістерської кваліфікаційної роботи «Метод та засіб криптоаналізу геш-функцій» студента Вінницького національного технічного університету Казміревського В. В., а саме:

– метод криптоаналізу геш-функцій;

– набір тестів для перевірки коректності реалізації геш-функції, розроблених мовою Dart.

Впроваджені результати дозволяють здійснювати перевірку програмних бібліотек з відкритим кодом, що реалізують геш-функції, що дало змогу підвищити ефективність управління підприємством.

Голова комісії:
Директор

А. І. Білоконь

Додаток В. Схеми алгоритмів

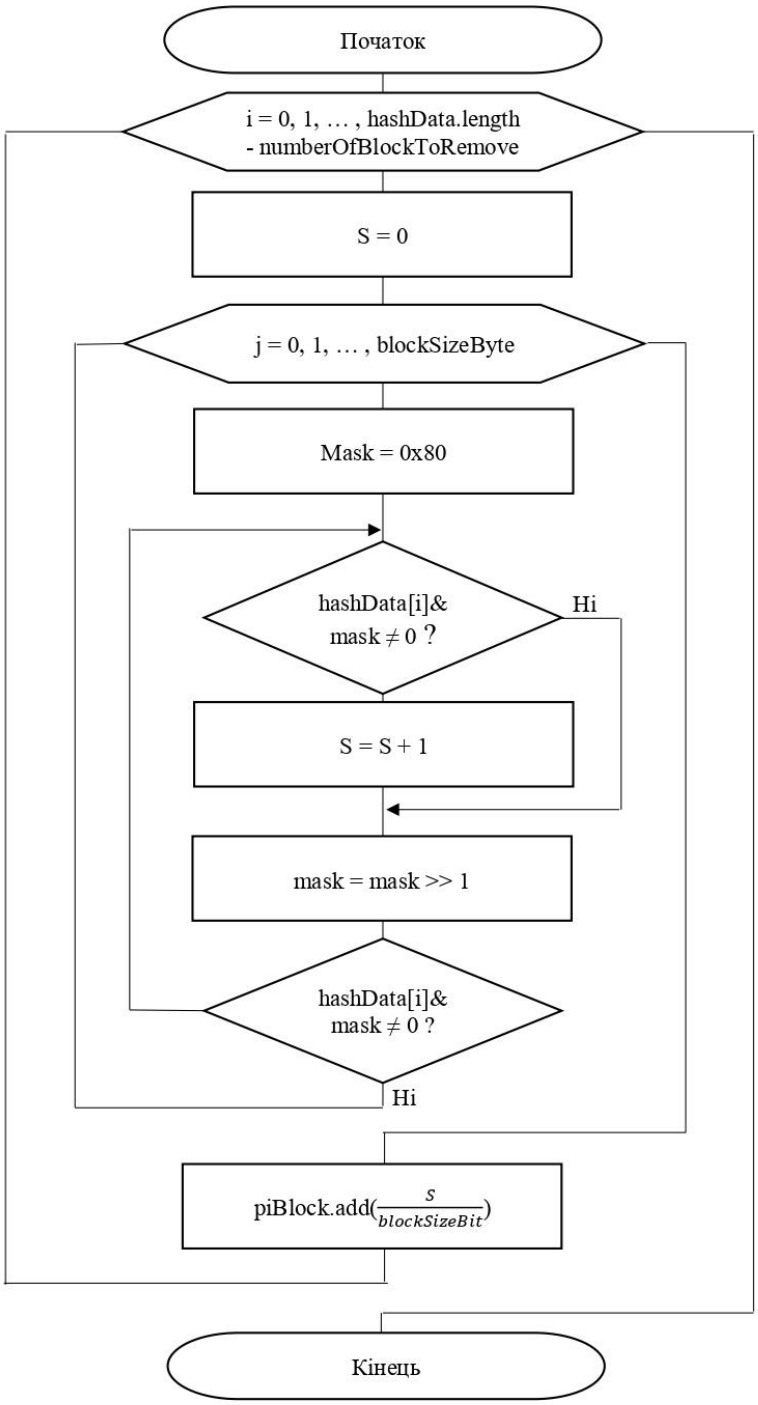


Рисунок В.1 – Алгоритм роботи блоку обчислення частки одиниць в кожному блоці M

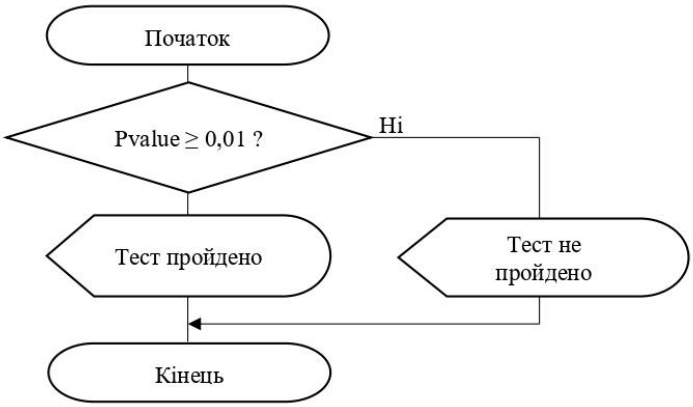


Рисунок В.2 – Алгоритм роботи блоку перевірки результату

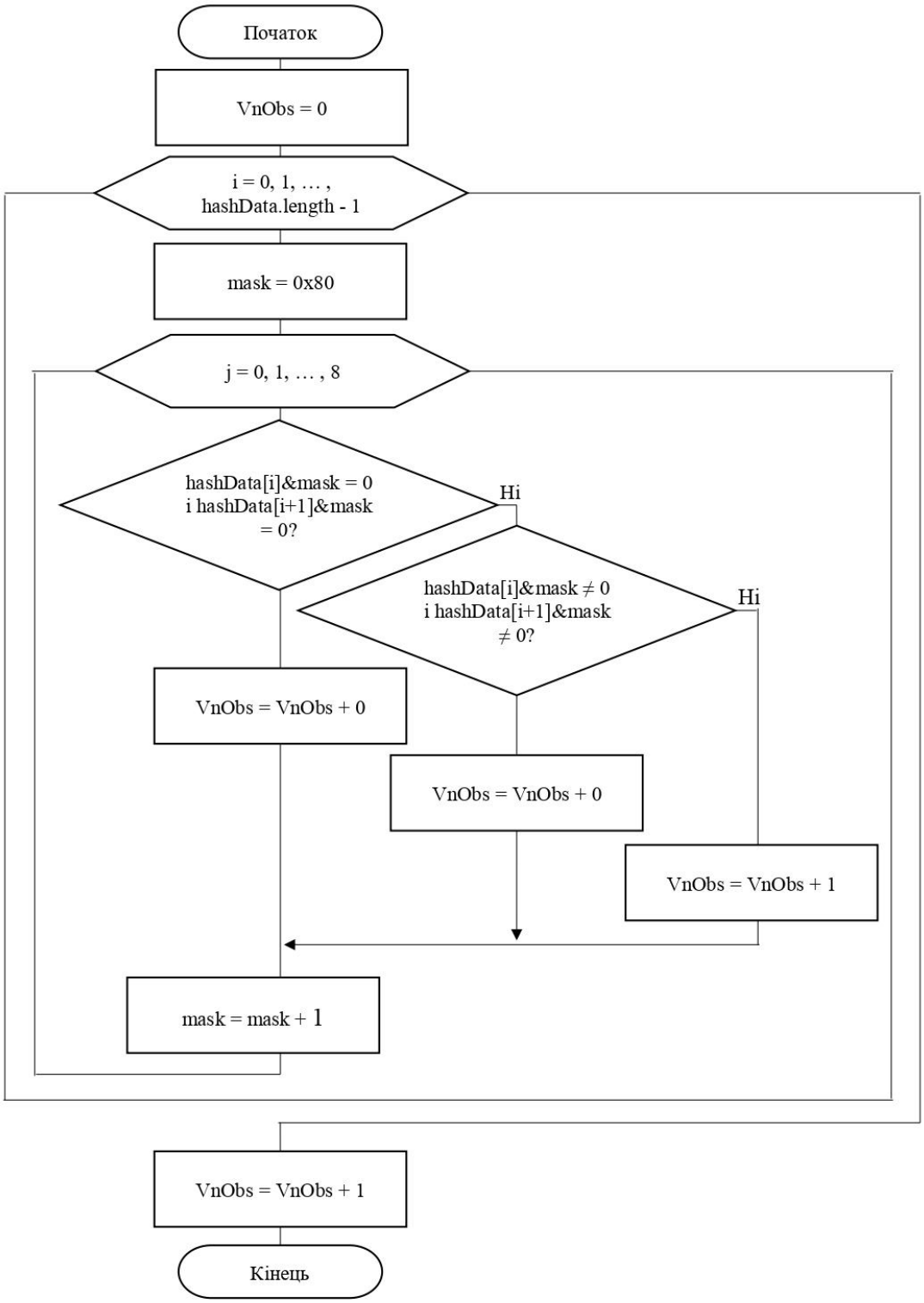


Рисунок В.3 – Алгоритм роботи блоку обчислення статистики тесту

Рисунок В.4 – Алгоритм роботи монобітного тесту

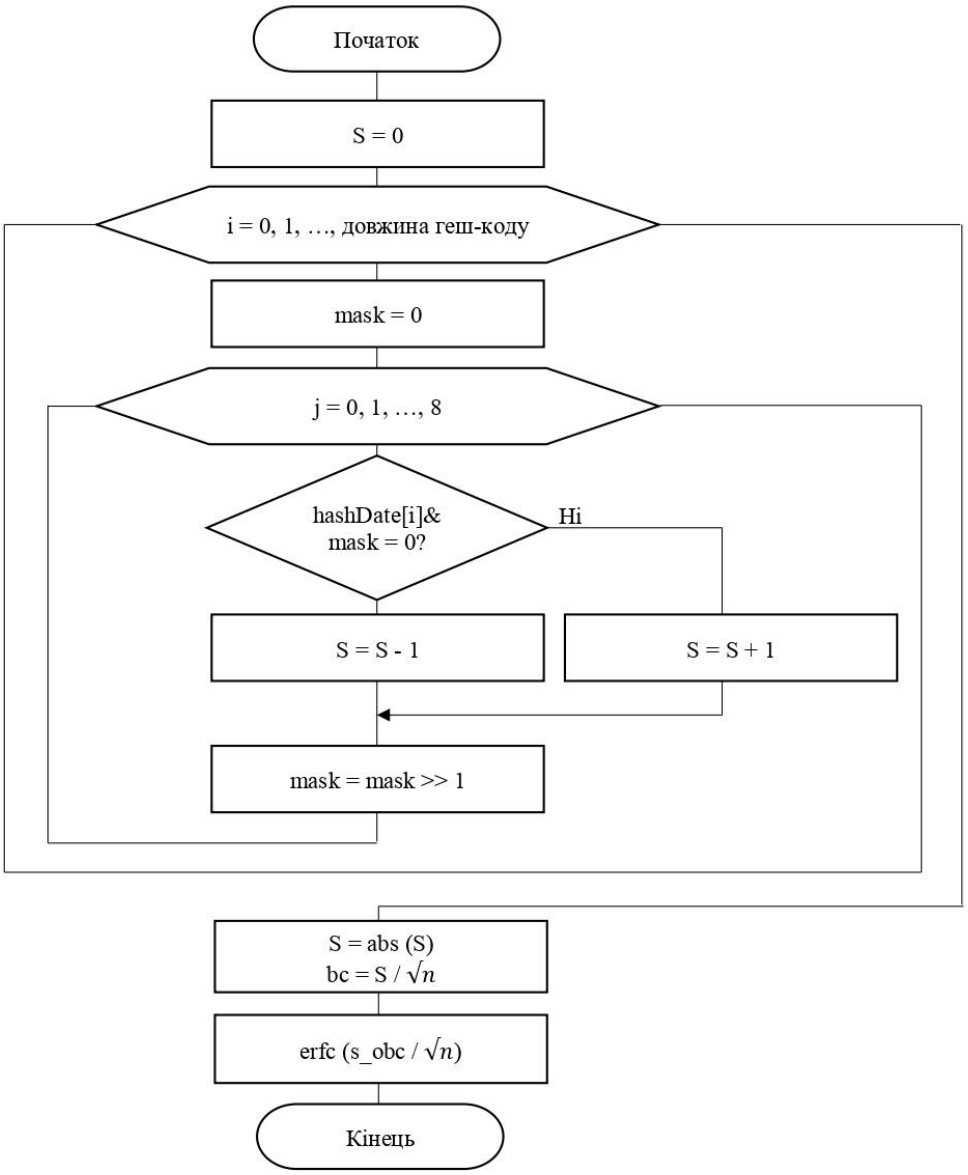


Рисунок В.5 – Алгоритм роботи блоку перевірки розподілу даних

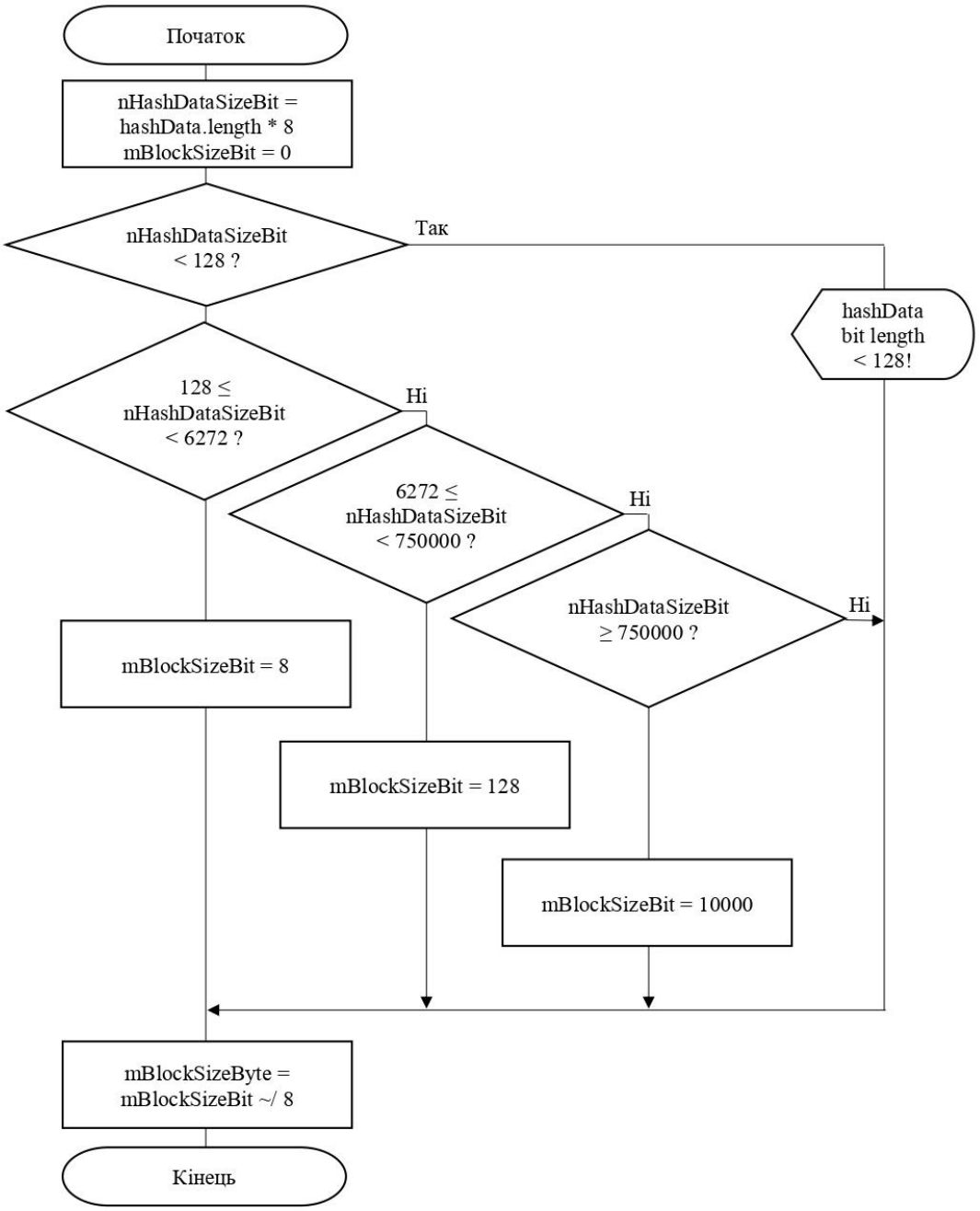


Рисунок В.6 – Алгоритм роботи блоку визначення довжини кожного блоку

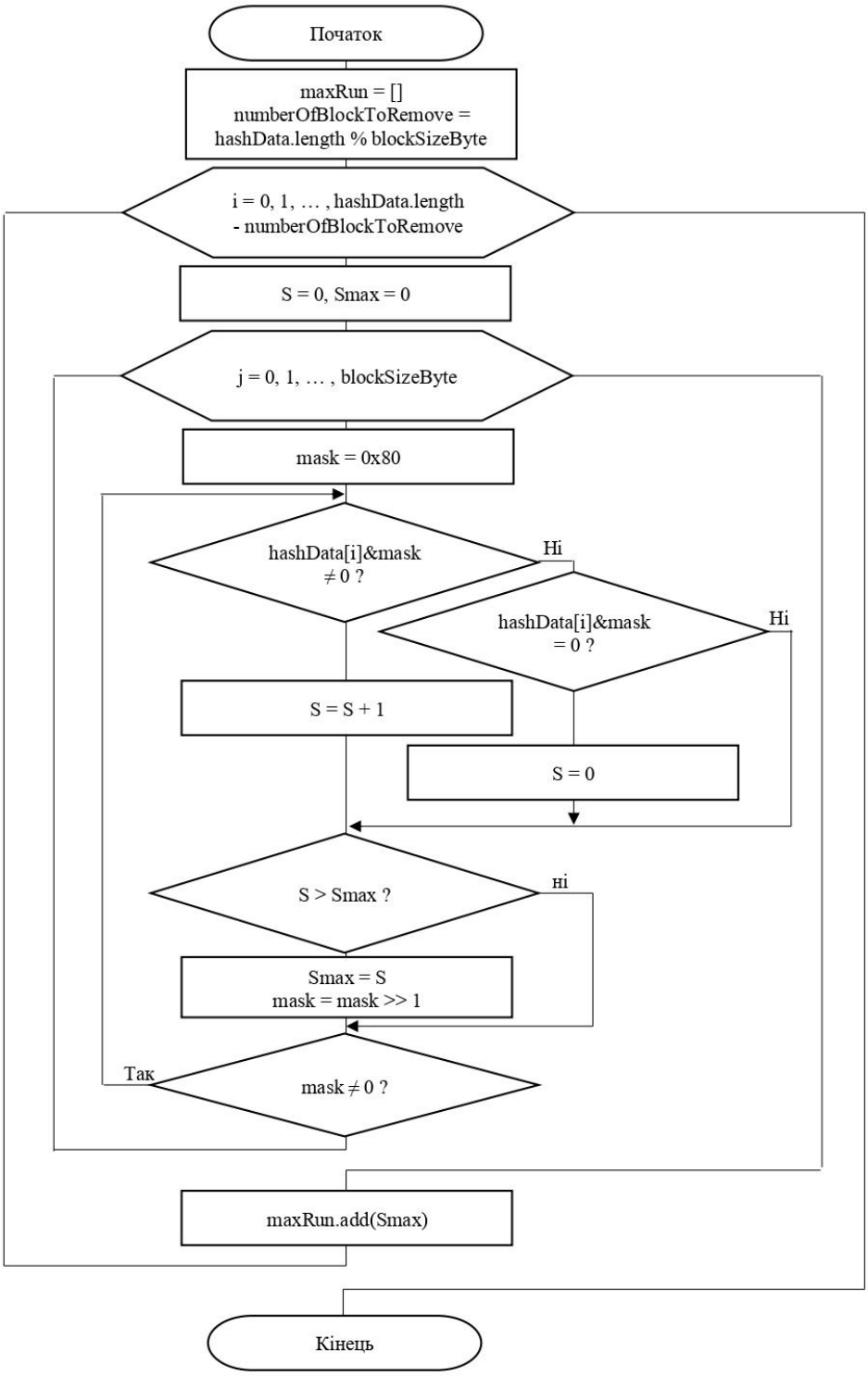


Рисунок В.7 – Алгоритм роботи блоку обчислення найдовшої послідовносці одиниць в кожному блоці

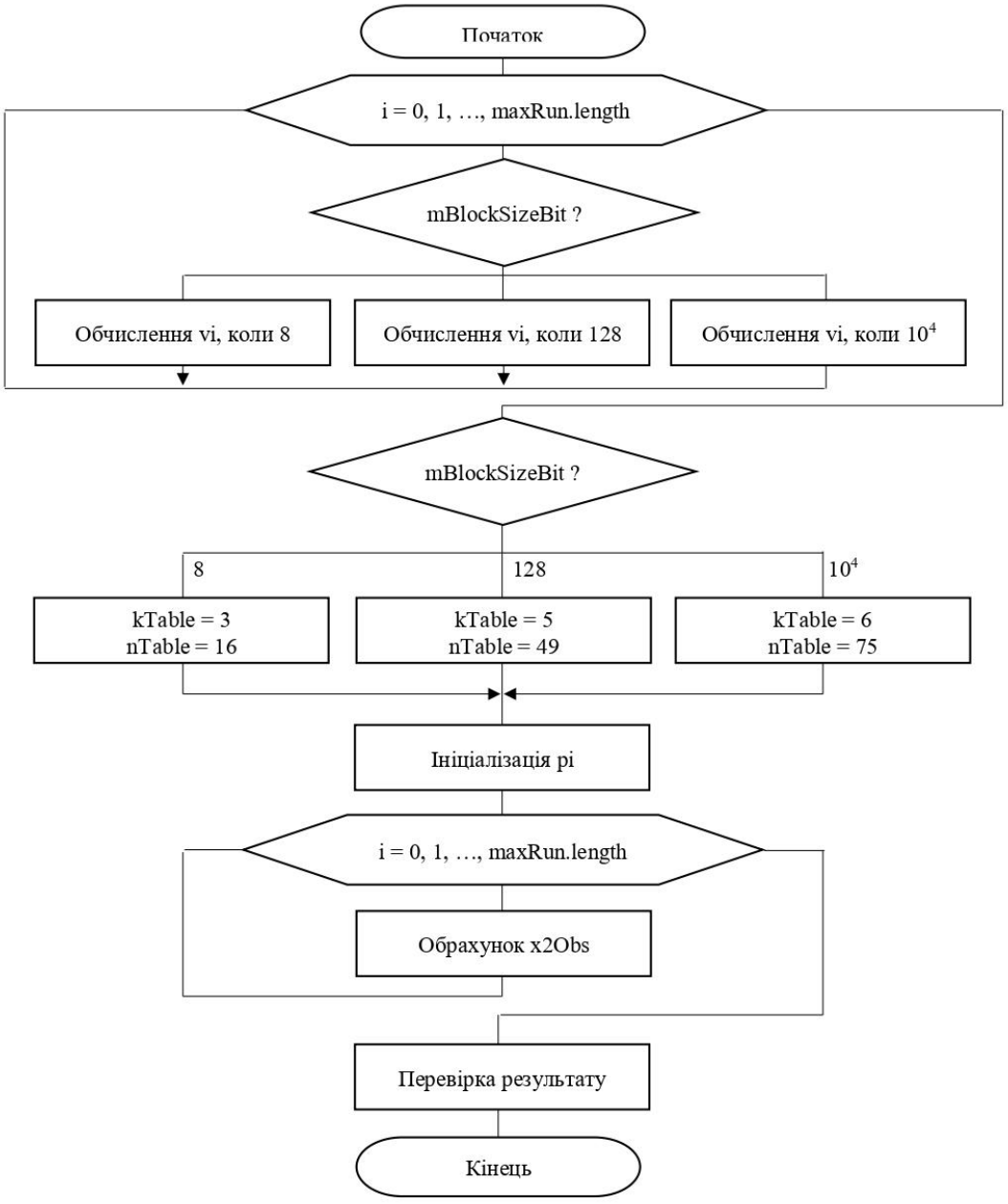


Рисунок В.8 – Алгоритм роботи блоку обчислення параметру v_i

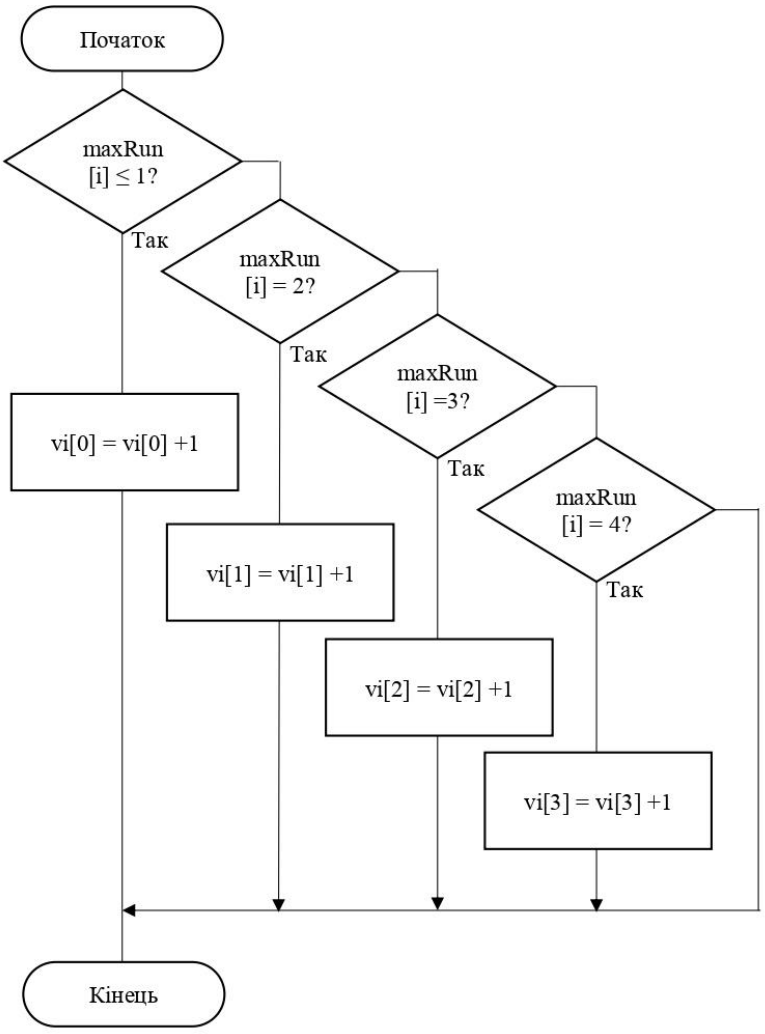


Рисунок В.9 – Алгоритм роботи блоку обчислення параметру v_i, коли mBlockSizeBit дорівнює 8

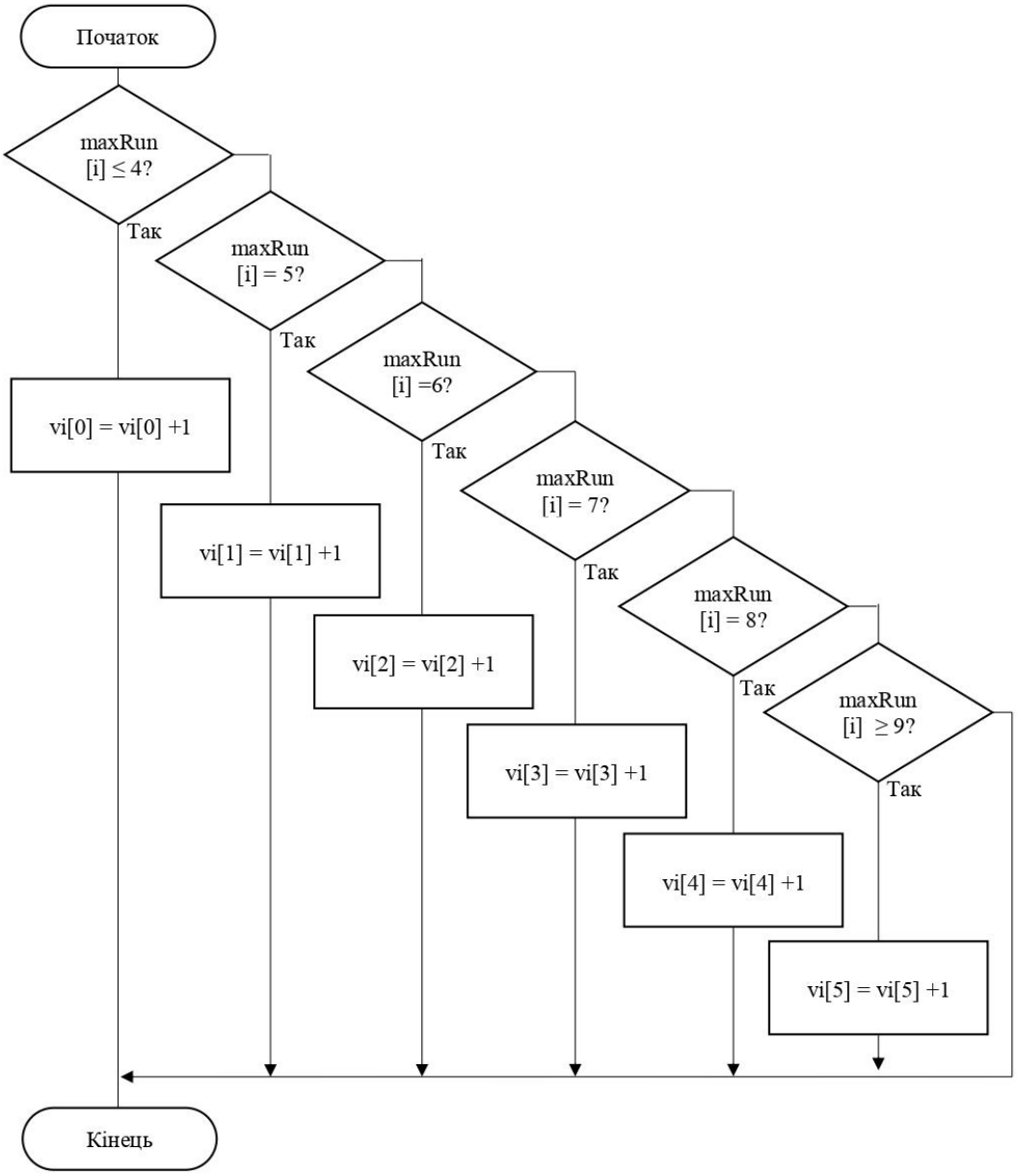


Рисунок В.10 – Алгоритм роботи блоку обчислення параметру v_i, коли mBlockSizeBit дорівнює 128

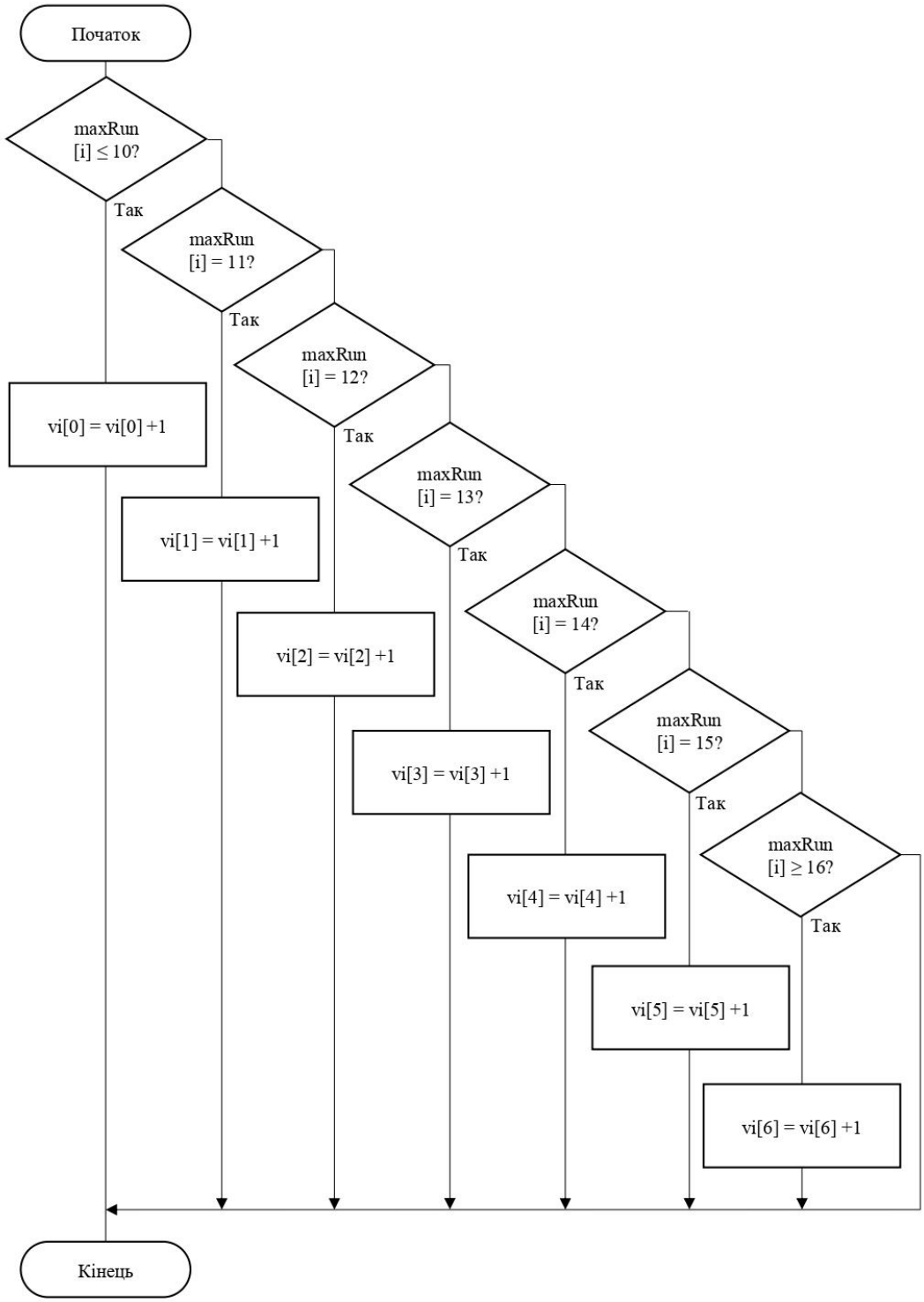


Рисунок В.11 – Алгоритм роботи блоку обчислення параметру v_i, коли mBlockSizeBit дорівнює 10^4

Додаток Г. Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Метод та засіб криптоаналізу геш-функцій
Автор роботи: Казміревський Віталій Віталійович
Тип роботи: магістерська кваліфікаційна робота

Підрозділ кафедра захисту інформації ФІТКІ
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність – 94,68%. Схожість – 5,32%.
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 _____ Каплун В. А.
(підпис) (прізвище, ініціали)

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

Автор роботи _____ Казміревський В.В.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Бармашов Ю.В.
(підпис) (прізвище, ініціали)

Додаток Ґ. Лістинг

crypto_dart_tests_test.dart

```
import 'package:flutter_test/flutter_test.dart';

import 'package:crypto_dart_tests/crypto_dart_tests.dart';

import 'blockCipherTests/blockCipherTests.dart' as blockCipherTests;
import 'streamCipherTests/streamCipherTests.dart' as streamCipherTests;
import 'hashTests/hashTests.dart' as hashTests;

void main() {
  // blockCipherTests.run();
  // streamCipherTests.run();
  hashTests.run();
}
```

crypto_dart_tests.dart

```
library crypto_dart_tests;

import 'blockCipher/blockCipherToTest.dart';
import 'hash/hashToTest.dart';
import 'streamCipher/streamCipherToTest.dart';
```

hashToTest.dart

```
import 'dart:typed_data';

import 'package:pointycastle/pointycastle.dart';

class HashFunctionsCollection {
  List<Map <String, dynamic>> mapHashFunctions = [
    {
      "name" : "SHA-224",
      "digest" : null,
      "outputLength": 224
    },
    {
      "name" : "SHA-256",
      "digest" : null,
      "outputLength": 256
    },
    {
      "name" : "SHA-384",
      "digest" : null,
      "outputLength": 384
    },
    {
      "name" : "SHA-512",
      "digest" : null,
      "outputLength": 512
    },
    {
      "name" : "SHA3-224",
      "digest" : null,
      "outputLength": 224
    },
    {
      "name" : "SHA3-256",
      "digest" : null,
      "outputLength": 256
    },
    {
      "name" : "SHA3-384",
      "digest" : null,
      "outputLength": 384
    },
    {
      "name" : "SHA3-512",
      "digest" : null,
      "outputLength": 512
    },
    {
      "name" : "RIPEMD-128",
      "digest" : null,
      "outputLength": 128
    },
    {
      "name" : "RIPEMD-160",
      "digest" : null,
      "outputLength": 160
    },
    {
      "name" : "RIPEMD-256",
      "digest" : null,
      "outputLength": 256
    },
    {
      "name" : "RIPEMD-320",
      "digest" : null,
      "outputLength": 320
    },
    {
      "name" : "scrypt",
      "digest" : null,
      "outputLength": 256 //?
    },
  ];


  HashFunctionsCollection() {
    int i;
    for(i = 0; i < mapHashFunctions.length - 1; i++) {
      mapHashFunctions[i]["digest"] = new Digest(mapHashFunctions[i]["name"]);
    }
    var scrypt = new KeyDerivator(mapHashFunctions[i]["name"]);
    scrypt.init(new ScryptParameters(1024, 2, 3, 256, Uint8List.fromList([0xaa, 0xaa]))); //use proper parameters
    mapHashFunctions[i]["digest"] = scrypt;
  }
}
```

vectorsTest.dart

```
import 'dart:typed_data';

class InputVectorsSHA2SHA3 {

 List<Map <String, dynamic>> mapInputVectorsSHA2SHA3 = [
  {
  "name" : "'empty'",
  "input" : null
  },
  {
   "name" : "'abc'",
   "input" : null
  },
  {
    "name" : "'abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq'",
    "input" : null
  },
  {
    "name" : "'abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmnoijklmnopjklmnopqklmnopqrlmnopqrsmnopqrstnopqrstu'",
    "input" : null
  },
 /*  {
    "name" : "'one million (1,000,000) repetitions of the character 'a''",
    "input" : null
  },
  {
    "name" : "'the extremely-long message 'abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmno' repeated 16,777,216 times: a bit string of length 233 bits (1 GB)'",
    "input" : null
  }*/];

 InputVectorsSHA2SHA3() {
  mapInputVectorsSHA2SHA3[0]["input"] = Uint8List.fromList([]);
```

```
    mapInputVectorsSHA2SHA3[1]["input"]                    =
Uint8List.fromList([0x61, 0x62, 0x63]);
    mapInputVectorsSHA2SHA3[2]["input"]                    =
Uint8List.fromList([0x61, 0x62, 0x63, 0x64, 0x62, 0x63,
      0x64, 0x65, 0x63, 0x64, 0x65, 0x66, 0x64, 0x65, 0x66, 0x67,
0x65, 0x66,
      0x67, 0x68, 0x66, 0x67, 0x68, 0x69, 0x67, 0x68, 0x69, 0x6a,
0x68, 0x69,
      0x6a, 0x6b, 0x69, 0x6a, 0x6b, 0x6c, 0x6a, 0x6b, 0x6c, 0x6d, 0x6b,
0x6c,
      0x6d, 0x6e, 0x6c, 0x6d, 0x6e, 0x6f, 0x6d, 0x6e, 0x6f, 0x70, 0x6e,
0x6f,
      0x70, 0x71]);
    mapInputVectorsSHA2SHA3[3]["input"]                    =
Uint8List.fromList([0x61, 0x62, 0x63,
      0x64, 0x65, 0x66, 0x67, 0x68, 0x62, 0x63, 0x64, 0x65, 0x66,
0x67, 0x68,
      0x69, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, 0x6a, 0x64,
0x65, 0x66,
      0x67, 0x68, 0x69, 0x6a, 0x6b, 0x65, 0x66, 0x67, 0x68, 0x69,
0x6a, 0x6b,
      0x6c, 0x66, 0x67, 0x68, 0x69, 0x6a, 0x6b, 0x6c, 0x6d, 0x67,
0x68, 0x69,
      0x6a, 0x6b, 0x6c, 0x6d, 0x6e, 0x68, 0x69, 0x6a, 0x6b, 0x6c, 0x6d,
0x6e,
      0x6f, 0x69, 0x6a, 0x6b, 0x6c, 0x6d, 0x6e, 0x6f, 0x70, 0x6a, 0x6b,
0x6c,
      0x6d, 0x6e, 0x6f, 0x70, 0x71, 0x6b, 0x6c, 0x6d, 0x6e, 0x6f, 0x70,
0x71,
      0x72, 0x6c, 0x6d, 0x6e, 0x6f, 0x70, 0x71, 0x72, 0x73, 0x6d, 0x6e,
0x6f,
      0x70, 0x71, 0x72, 0x73, 0x74, 0x6e, 0x6f, 0x70, 0x71, 0x72,
0x73, 0x74,
      0x75]);

    /*List <int> millionA = [];
    for(int i = 0; i < 1000000; i++){
      millionA.add(0x61);
    }
    mapInputVectorsSHA2SHA3[4]["input"]                    =
Uint8List.fromList(millionA);

    List <int> vector1GB = [];
    List <int> vectorForGB = [0x61, 0x62, 0x63, 0x64, 0x65, 0x66,
0x67, 0x68,
      0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, 0x63, 0x64,
0x65, 0x66,
      0x67, 0x68, 0x69, 0x6a, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69,
0x6a, 0x6b,
      0x65, 0x66, 0x67, 0x68, 0x69, 0x6a, 0x6b, 0x6c, 0x66, 0x67,
0x68, 0x69,
      0x6a, 0x6b, 0x6c, 0x6d, 0x67, 0x68, 0x69, 0x6a, 0x6b, 0x6c,
0x6d, 0x6e,
      0x68, 0x69, 0x6a, 0x6b, 0x6c, 0x6d, 0x6e, 0x6f, 0x69, 0x6a, 0x6b,
0x6c,
      0x6d, 0x6e, 0x6f, 0x70, 0x6a, 0x6b, 0x6c, 0x6d, 0x6e, 0x6f, 0x70,
0x71,
      0x6b, 0x6c, 0x6d, 0x6e, 0x6f, 0x70, 0x71, 0x72, 0x6c, 0x6d, 0x6e,
0x6f,
      0x70, 0x71, 0x72, 0x73, 0x6d, 0x6e, 0x6f, 0x70, 0x71, 0x72,
0x73, 0x74,
      0x6e, 0x6f, 0x70, 0x71, 0x72, 0x73, 0x74, 0x75];
    for(int i = 0; i < 16777216; i++){
      vector1GB += vectorForGB;
    }

    mapInputVectorsSHA2SHA3[5]["input"]                    =
Uint8List.fromList(vector1GB);*/
  }

}

class OutputVectorsSHA224 {

  List<Map <String, dynamic>> mapOutputVectorsSHA224 = [
    {
      "name" : "'empty'",
      "output" : null
    },
    {
      "name" : "'abc'",
      "output" : null
    },
    {
      "name"                                               :
"'abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq'",
      "output" : null
    },
    {
      "name"                                               :
"'abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmn
oijklmnopjklmnopqklmnopqrlmnopqrsmnopqrstnopqrstu'",
      "output" : null
    },
    /*{
      "name" : "'one million (1,000,000) repetitions of the character 'a''",
      "output" : null
    },
    {
      "name"      :      "'the      extremely-long      message
'abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmno
' repeated 16,777,216 times: a bit string of length 233 bits (1 GB)'",
      "output" : null
    }*/];

  OutputVectorsSHA224() {
    mapOutputVectorsSHA224[0]["output"]                    =
Uint8List.fromList([0xd1, 0x4a, 0x02,
      0x8c, 0x2a, 0x3a, 0x2b, 0xc9, 0x47, 0x61, 0x02, 0xbb, 0x28,
0x82, 0x34,
      0xc4, 0x15, 0xa2, 0xb0, 0x1f, 0x82, 0x8e, 0xa6, 0x2a, 0xc5, 0xb3,
0xe4,
      0x2f]);
    mapOutputVectorsSHA224[1]["output"]                    =
Uint8List.fromList([0x23, 0x09, 0x7d,
      0x22, 0x34, 0x05, 0xd8, 0x22, 0x86, 0x42, 0xa4, 0x77, 0xbd,
0xa2, 0x55,
      0xb3, 0x2a, 0xad, 0xbc, 0xe4, 0xbd, 0xa0, 0xb3, 0xf7, 0xe3, 0x6c,
0x9d,
      0xa7]);
    mapOutputVectorsSHA224[2]["output"]                    =
Uint8List.fromList([0x75, 0x38, 0x8b,
      0x16, 0x51, 0x27, 0x76, 0xcc, 0x5d, 0xba, 0x5d, 0xa1, 0xfd, 0x89,
0x01,
      0x50, 0xb0, 0xc6, 0x45, 0x5c, 0xb4, 0xf5, 0x8b, 0x19, 0x52, 0x52,
0x25,
      0x25]);
    mapOutputVectorsSHA224[3]["output"]                    =
Uint8List.fromList([0xc9, 0x7c, 0xa9,
      0xa5, 0x59, 0x85, 0x0c, 0xe9, 0x7a, 0x04, 0xa9, 0x6d, 0xef, 0x6d,
0x99,
      0xa9, 0xe0, 0xe0, 0xe2, 0xab, 0x14, 0xe6, 0xb8, 0xdf, 0x26, 0x5f,
0xc0,
      0xb3]);
    /*    mapOutputVectorsSHA224[4]["output"]                    =
Uint8List.fromList([0x20, 0x79, 0x46,
      0x55, 0x98, 0x0c, 0x91, 0xd8, 0xbb, 0xb4, 0xc1, 0xea, 0x97,
0x61, 0x8a,
      0x4b, 0xf0, 0x3f, 0x42, 0x58, 0x19, 0x48, 0xb2, 0xee, 0x4e, 0xe7,
0xad,
      0x67]);
    mapOutputVectorsSHA224[5]["output"]                    =
Uint8List.fromList([0xb5, 0x98, 0x97,
      0x13, 0xca, 0x4f, 0xe4, 0x7a, 0x00, 0x9f, 0x86, 0x21, 0x98, 0x0b,
0x34,
      0xe6, 0xd6, 0x3e, 0xd3, 0x06, 0x3b, 0x2a, 0x0a, 0x2c, 0x86, 0x7d,
0x8a,
      0x85]);*/
  }
}

class OutputVectorsSHA256 {

  List<Map <String, dynamic>> mapOutputVectorsSHA256 = [
    {
      "name" : "'empty'",
      "output" : null
    },
```

```
    {
      "name" : "'abc'",
      "output" : null
    },
    {
      "name"                                                                    :
"'abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq'",
      "output" : null
    },
    {
      "name"                                                                    :
"'abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmn
oijklmnopjklmnopqklmnopqrlmnopqrsmnopqrstnopqrstu'",
      "output" : null
    },
    /*{
      "name" : "'one million (1,000,000) repetitions of the character 'a'",
      "output" : null
    },
    {
      "name"      :      "'the      extremely-long      message
'abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmno
' repeated 16,777,216 times: a bit string of length 233 bits (1 GB)'",
      "output" : null
    }*/];

  OutputVectorsSHA256() {
    mapOutputVectorsSHA256[0]["output"]                                 =
Uint8List.fromList([0xe3, 0xb0, 0xc4,
      0x42, 0x98, 0xfc, 0x1c, 0x14, 0x9a, 0xfb, 0xf4, 0xc8, 0x99, 0x6f,
0xb9,
      0x24, 0x27, 0xae, 0x41, 0xe4, 0x64, 0x9b, 0x93, 0x4c, 0xa4, 0x95,
0x99,
      0x1b, 0x78, 0x52, 0xb8, 0x55]);
    mapOutputVectorsSHA256[1]["output"]                                 =
Uint8List.fromList([0xba, 0x78, 0x16,
      0xbf, 0x8f, 0x01, 0xcf, 0xea, 0x41, 0x41, 0x40, 0xde, 0x5d, 0xae,
0x22,
      0x23, 0xb0, 0x03, 0x61, 0xa3, 0x96, 0x17, 0x7a, 0x9c, 0xb4,
0x10, 0xff,
      0x61, 0xf2, 0x00, 0x15, 0xad]);
    mapOutputVectorsSHA256[2]["output"]                                 =
Uint8List.fromList([0x24, 0x8d, 0x6a,
      0x61, 0xd2, 0x06, 0x38, 0xb8, 0xe5, 0xc0, 0x26, 0x93, 0x0c,
0x3e, 0x60,
      0x39, 0xa3, 0x3c, 0xe4, 0x59, 0x64, 0xff, 0x21, 0x67, 0xf6, 0xec,
0xed,
      0xd4, 0x19, 0xdb, 0x06, 0xc1]);
    mapOutputVectorsSHA256[3]["output"]                                 =
Uint8List.fromList([0xcf, 0x5b, 0x16,
      0xa7, 0x78, 0xaf, 0x83, 0x80, 0x03, 0x6c, 0xe5, 0x9e, 0x7b, 0x04,
0x92,
      0x37, 0x0b, 0x24, 0x9b, 0x11, 0xe8, 0xf0, 0x7a, 0x51, 0xaf, 0xac,
0x45,
      0x03, 0x7a, 0xfe, 0xe9, 0xd1]);
   /*                    mapOutputVectorsSHA256[4]["output"]             =
Uint8List.fromList([0xcd, 0xc7, 0x6e,
      0x5c, 0x99, 0x14, 0xfb, 0x92, 0x81, 0xa1, 0xc7, 0xe2, 0x84, 0xd7,
0x3e,
      0x67, 0xf1, 0x80, 0x9a, 0x48, 0xa4, 0x97, 0x20, 0x0e, 0x04, 0x6d,
0x39,
      0xcc, 0xc70, 0x11, 0x2c, 0xd0]);
    mapOutputVectorsSHA256[5]["output"]                                 =
Uint8List.fromList([0x50, 0xe7, 0x2a,
      0x0e, 0x26, 0x44, 0x2f, 0xe2, 0x55, 0x2d, 0xc3, 0x93, 0x8a, 0xc5,
0x86,
      0x58, 0x22, 0x8c, 0x0c, 0xbf, 0xb1, 0xd2, 0xca, 0x87, 0x2a, 0xe4,
0x35,
      0x26, 0x6f, 0xcd, 0x05, 0x5e]);*/
  }
}

class OutputVectorsSHA384 {

  List<Map <String, dynamic>> mapOutputVectorsSHA384 = [
    {
      "name" : "'empty'",
      "output" : null
    },
    {
      "name" : "'abc'",
      "output" : null
    },
    {
      "name"                                                                    :
"'abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq'",
      "output" : null
    },
    {
      "name"                                                                    :
"'abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmn
oijklmnopjklmnopqklmnopqrlmnopqrsmnopqrstnopqrstu'",
      "output" : null
    },
    /*{
      "name" : "'one million (1,000,000) repetitions of the character 'a'",
      "output" : null
    },
    {
      "name"      :      "'the      extremely-long      message
'abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmno
' repeated 16,777,216 times: a bit string of length 233 bits (1 GB)'",
      "output" : null
    }*/];

  OutputVectorsSHA384() {
    mapOutputVectorsSHA384[0]["output"]                                 =
Uint8List.fromList([0x38, 0xb0, 0x60,
      0xa7, 0x51, 0xac, 0x96, 0x38, 0x4c, 0xd9, 0x32, 0x7e, 0xb1, 0xb1,
0xe3,
      0x6a, 0x21, 0xfd, 0xb7, 0x11, 0x14, 0xbe, 0x07, 0x43, 0x4c, 0x0c,
0xc7,
      0xbf, 0x63, 0xf6, 0xe1, 0xda, 0x27, 0x4e, 0xde, 0xbf, 0xe7, 0x6f,
0x65,
      0xfb, 0xd5, 0x1a, 0xd2, 0xf1, 0x48, 0x98, 0xb9, 0x5b]);
    mapOutputVectorsSHA384[1]["output"]                                 =
Uint8List.fromList([0xcb, 0x00, 0x75,
      0x3f, 0x45, 0xa3, 0x5e, 0x8b, 0xb5, 0xa0, 0x3d, 0x69, 0x9a, 0xc6,
0x50,
      0x07, 0x27, 0x2c, 0x32, 0xab, 0x0e, 0xde, 0xd1, 0x63, 0x1a, 0x8b,
0x60,
      0x5a, 0x43, 0xff, 0x5b, 0xed, 0x80, 0x86, 0x07, 0x2b, 0xa1, 0xe7,
0xcc,
      0x23, 0x58, 0xba, 0xec, 0xa1, 0x34, 0xc8, 0x25, 0xa7]);
    mapOutputVectorsSHA384[2]["output"]                                 =
Uint8List.fromList([0x33, 0x91, 0xfd,
      0xdd, 0xfc, 0x8d, 0xc7, 0x39, 0x37, 0x07, 0xa6, 0x5b, 0x1b, 0x47,
0x09,
      0x39, 0x7c, 0xf8, 0xb1, 0xd1, 0x62, 0xaf, 0x05, 0xab, 0xfe, 0x8f,
0x45,
      0x0d, 0xe5, 0xf3, 0x6b, 0xc6, 0xb0, 0x45, 0x5a, 0x85, 0x20, 0xbc,
0x4e,
      0x6f, 0x5f, 0xe9, 0x5b, 0x1f, 0xe3, 0xc8, 0x45, 0x2b]);
    mapOutputVectorsSHA384[3]["output"]                                 =
Uint8List.fromList([0x09, 0x33, 0x0c,
      0x33, 0xf7, 0x11, 0x47, 0xe8, 0x3d, 0x19, 0x2f, 0xc7, 0x82, 0xcd,
0x1b,
      0x47, 0x53, 0x11, 0x1b, 0x17, 0x3b, 0x3b, 0x05, 0xd2, 0x2f,
0xa0, 0x80,
      0x86, 0xe3, 0xb0, 0xf7, 0x12, 0xfc, 0xc7, 0xc7, 0x1a, 0x55, 0x7e,
0x2d,
      0xb9, 0x66, 0xc3, 0xe9, 0xfa, 0x91, 0x74, 0x60, 0x39]);
   /*                    mapOutputVectorsSHA384[4]["output"]             =
Uint8List.fromList([0x9d, 0x0e, 0x18,
      0x09, 0x71, 0x64, 0x74, 0xcb, 0x08, 0x6e, 0x83, 0x4e, 0x31,
0x0a, 0x4a,
      0x1c, 0xed, 0x14, 0x9e, 0x9c, 0x00, 0xf2, 0x48, 0x52, 0x79, 0x72,
0xce,
      0xc5, 0x70, 0x4c, 0x2a, 0x5b, 0x07, 0xb8, 0xb3, 0xdc, 0x38, 0xec,
0xc4,
      0xeb, 0xae, 0x97, 0xdd, 0xd8, 0x7f, 0x3d, 0x89, 0x85]);
    mapOutputVectorsSHA384[5]["output"]                                 =
Uint8List.fromList([0x54, 0x41, 0x23,
      0x5c, 0xc0, 0x23, 0x53, 0x41, 0xed, 0x80, 0x6a, 0x64, 0xfb, 0x35,
0x47,
      0x42, 0xb5, 0xe5, 0xc0, 0x2a, 0x3c, 0x5c, 0xb7, 0x1b, 0x5f, 0x63,
0xfb,
```

```
    0x79, 0x34, 0x58, 0xd8, 0xfd, 0xae, 0x59, 0x9c, 0x8c, 0xd8, 0x88,
0x49,
    0x43, 0xc0, 0x4f, 0x11, 0xb3, 0x1b, 0x89, 0xf0, 0x23]);*/
  }
}

class OutputVectorsSHA512 {

  List<Map <String, dynamic>> mapOutputVectorsSHA512 = [
    {
      "name" : "'empty'",
      "output" : null
    },
    {
      "name" : "'abc'",
      "output" : null
    },
    {
      "name"                                                    :
"'abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq'",
      "output" : null
    },
    {
      "name"                                                    :
"'abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmn
oijklmnopjklmnopqklmnopqrlmnopqrsmnopqrstnopqrstu'",
      "output" : null
    },
    /*{
      "name" : "'one million (1,000,000) repetitions of the character 'a''",
      "output" : null
    },
    {
      "name"      :      "'the      extremely-long      message
'abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmno
' repeated 16,777,216 times: a bit string of length 233 bits (1 GB)'",
      "output" : null
    }*/];

  OutputVectorsSHA512() {
    mapOutputVectorsSHA512[0]["output"]                         =
Uint8List.fromList([0xcf, 0x83, 0xe1,
      0x35,0x7e, 0xef, 0xb8, 0xbd, 0xf1, 0x54, 0x28, 0x50, 0xd6, 0x6d,
0x80,
      0x07, 0xd6, 0x20, 0xe4, 0x05, 0x0b, 0x57, 0x15, 0xdc, 0x83, 0xf4,
0xa9,
      0x21, 0xd3, 0x6c, 0xe9, 0xce, 0x47, 0xd0, 0xd1, 0x3c, 0x5d, 0x85,
0xf2,
      0xb0, 0xff, 0x83, 0x18, 0xd2, 0x87, 0x7e, 0xec, 0x2f, 0x63, 0xb9,
0x31,
      0xbd, 0x47, 0x41, 0x7a, 0x81, 0xa5, 0x38, 0x32, 0x7a, 0xf9, 0x27,
0xda,
      0x3e]);
    mapOutputVectorsSHA512[1]["output"]                         =
Uint8List.fromList([0xdd, 0xaf, 0x35,
      0xa1, 0x93, 0x61, 0x7a, 0xba, 0xcc, 0x41, 0x73, 0x49, 0xae, 0x20,
0x41,
      0x31, 0x12, 0xe6, 0xfa, 0x4e, 0x89, 0xa9, 0x7e, 0xa2, 0x0a, 0x9e,
0xee,
      0xe6, 0x4b, 0x55, 0xd3, 0x9a, 0x21, 0x92, 0x99, 0x2a, 0x27, 0x4f,
0xc1,
      0xa8, 0x36, 0xba, 0x3c, 0x23, 0xa3, 0xfe, 0xeb, 0xbd, 0x45, 0x4d,
0x44,
      0x23, 0x64, 0x3c, 0xe8, 0x0e, 0x2a, 0x9a, 0xc9, 0x4f, 0xa5, 0x4c,
0xa4,
      0x9f]);
    mapOutputVectorsSHA512[2]["output"]                         =
Uint8List.fromList([0x20, 0x4a, 0x8f,
      0xc6, 0xdd, 0xa8, 0x2f, 0x0a, 0x0c, 0xed, 0x7b, 0xeb, 0x8e, 0x08,
0xa4,
      0x16, 0x57, 0xc1, 0x6e, 0xf4, 0x68, 0xb2, 0x28, 0xa8, 0x27, 0x9b,
0xe3,
      0x31, 0xa7, 0x03, 0xc3, 0x35, 0x96, 0xfd, 0x15, 0xc1, 0x3b, 0x1b,
0x07,
      0xf9, 0xaa, 0x1d, 0x3b, 0xea, 0x57, 0x78, 0x9c, 0xa0, 0x31, 0xad,
0x85,
      0xc7, 0xa7, 0x1d, 0xd7, 0x03, 0x54, 0xec, 0x63, 0x12, 0x38, 0xca,
0x34,
      0x45]);
    mapOutputVectorsSHA512[3]["output"]                         =
Uint8List.fromList([0x8e, 0x95, 0x9b,
      0x75, 0xda, 0xe3, 0x13, 0xda, 0x8c, 0xf4, 0xf7, 0x28, 0x14, 0xfc,
0x14,
      0x3f, 0x8f, 0x77, 0x79, 0xc6, 0xeb, 0x9f, 0x7f, 0xa1, 0x72, 0x99,
0xae,
      0xad, 0xb6, 0x88, 0x90, 0x18, 0x50, 0x1d, 0x28, 0x9e, 0x49,
0x00, 0xf7,
      0xe4, 0x33, 0x1b, 0x99, 0xde, 0xc4, 0xb5, 0x43, 0x3a, 0xc7, 0xd3,
0x29,
      0xee, 0xb6, 0xdd, 0x26, 0x54, 0x5e, 0x96, 0xe5, 0x5b, 0x87,
0x4b, 0xe9,
      0x09]);
    /*mapOutputVectorsSHA512[4]["output"]                         =
Uint8List.fromList([0xe7, 0x18, 0x48,
      0x3d, 0x0c, 0xe7, 0x69, 0x64, 0x4e, 0x2e, 0x42, 0xc7, 0xbc, 0x15,
0xb4,
      0x63, 0x8e, 0x1f, 0x98, 0xb1, 0x3b, 0x20, 0x44, 0x28, 0x56,
0x32, 0xa8,
      0x03, 0xaf, 0xa9, 0x73, 0xeb, 0xde, 0x0f, 0xf2, 0x44, 0x87, 0x7e,
0xa6,
      0x0a, 0x4c, 0xb0, 0x43, 0x2c, 0xe5, 0x77, 0xc3, 0x1b, 0xeb, 0x00,
0x9c,
      0x5c, 0x2c, 0x49, 0xaa, 0x2e, 0x4e, 0xad, 0xb2, 0x17, 0xad, 0x8c,
0xc0,
      0x9b]);
    mapOutputVectorsSHA512[5]["output"]                         =
Uint8List.fromList([0xb4, 0x7c, 0x93,
      0x34, 0x21, 0xea, 0x2d, 0xb1, 0x49, 0xad, 0x6e, 0x10, 0xfc, 0xe6,
0xc7,
      0xf9, 0x3d, 0x07, 0x52, 0x38, 0x01, 0x80, 0xff, 0xd7, 0xf4, 0x62,
0x9a,
      0x71, 0x21, 0x34, 0x83, 0x1d, 0x77, 0xbe, 0x60, 0x91, 0xb8,
0x19, 0xed,
      0x35, 0x2c, 0x29, 0x67, 0xa2, 0xe2, 0xd4, 0xfa, 0x50, 0x50, 0x72,
0x3c,
      0x96, 0x30, 0x69, 0x1f, 0x1a, 0x05, 0xa7, 0x28, 0x1d, 0xbe, 0x6c,
0x10,
      0x86]);*/
  }
}



class OutputVectorsSHA3224 {

  List<Map <String, dynamic>> mapOutputVectorsSHA3224 = [
    {
      "name" : "'empty'",
      "output" : null
    },
    {
      "name" : "'abc'",
      "output" : null
    },
    {
      "name"                                                    :
"'abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq'",
      "output" : null
    },
    {
      "name"                                                    :
"'abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmn
oijklmnopjklmnopqklmnopqrlmnopqrsmnopqrstnopqrstu'",
      "output" : null
    },
    /*{
      "name" : "'one million (1,000,000) repetitions of the character 'a''",
      "output" : null
    },
    {
      "name"      :      "'the      extremely-long      message
'abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmno
' repeated 16,777,216 times: a bit string of length 233 bits (1 GB)'",
      "output" : null
    }*/];

  OutputVectorsSHA3224() {
```

```
    mapOutputVectorsSHA3224[0]["output"] =
Uint8List.fromList([0x6b, 0x4e, 0x03,
      0x42, 0x36, 0x67, 0xdb, 0xb7, 0x3b, 0x6e, 0x15, 0x45, 0x4f, 0x0e,
0xb1,
      0xab, 0xd4, 0x59, 0x7f, 0x9a, 0x1b, 0x07, 0x8e, 0x3f, 0x5b, 0x5a,
0x6b,
      0xc7]);
    mapOutputVectorsSHA3224[1]["output"] =
Uint8List.fromList([0xe6, 0x42, 0x82,
      0x4c, 0x3f, 0x8c, 0xf2, 0x4a, 0xd0, 0x92, 0x34, 0xee, 0x7d, 0x3c,
0x76,
      0x6f, 0xc9, 0xa3, 0xa5, 0x16, 0x8d, 0x0c, 0x94, 0xad, 0x73, 0xb4,
0x6f,
      0xdf]);
    mapOutputVectorsSHA3224[2]["output"] =
Uint8List.fromList([0x8a, 0x24, 0x10,
      0x8b, 0x15, 0x4a, 0xda, 0x21, 0xc9, 0xfd, 0x55, 0x74, 0x49, 0x44,
0x79,
      0xba, 0x5c, 0x7e, 0x7a, 0xb7, 0x6e, 0xf2, 0x64, 0xea, 0xd0, 0xfc,
0xce,
      0x33]);
    mapOutputVectorsSHA3224[3]["output"] =
Uint8List.fromList([0x54, 0x3e, 0x68,
      0x68, 0xe1, 0x66, 0x6c, 0x1a, 0x64, 0x36, 0x30, 0xdf, 0x77, 0x36,
0x7a,
      0xe5, 0xa6, 0x2a, 0x85, 0x07, 0x0a, 0x51, 0xc1, 0x4c, 0xbf, 0x66,
0x5c,
      0xbc]);
    /*              mapOutputVectorsSHA3224[4]["output"] =
Uint8List.fromList([0xd6, 0x93, 0x35,
      0xb9, 0x33, 0x25, 0x19, 0x2e, 0x51, 0x6a, 0x91, 0x2e, 0x6d,
0x19, 0xa1,
      0x5c, 0xb5, 0x1c, 0x6e, 0xd5, 0xc1, 0x52, 0x43, 0xe7, 0xa7, 0xfd,
0x65,
      0x3c]);
    mapOutputVectorsSHA3224[5]["output"] =
Uint8List.fromList([0xc6, 0xd6, 0x6e,
      0x77, 0xae, 0x28, 0x95, 0x66, 0xaf, 0xb2, 0xce, 0x39, 0x27, 0x77,
0x52,
      0xd6, 0xda, 0x2a, 0x3c, 0x46, 0x01, 0x0f, 0x1e, 0x0a, 0x09, 0x70,
0xff,
      0x60]);*/
  }
}

class OutputVectorsSHA3256 {

  List<Map <String, dynamic>> mapOutputVectorsSHA3256 = [
    {
      "name" : "'empty'",
      "output" : null
    },
    {
      "name" : "'abc'",
      "output" : null
    },
    {
      "name"                                                    :
"'abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq'",
      "output" : null
    },
    {
      "name"                                                    :
"'abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmn
oijklmnopjklmnopqklmnopqrlmnopqrsmnopqrstnopqrstu'",
      "output" : null
    },
    /*{
      "name" : "'one million (1,000,000) repetitions of the character 'a'",
      "output" : null
    },
    {
      "name"      :      "'the      extremely-long      message
'abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmno
' repeated 16,777,216 times: a bit string of length 233 bits (1 GB)'",
      "output" : null
    }*/];

  OutputVectorsSHA3256() {
```

```
    mapOutputVectorsSHA3256[0]["output"] =
Uint8List.fromList([0xa7, 0xff, 0xc6,
      0xf8, 0xbf, 0x1e, 0xd7, 0x66, 0x51, 0xc1, 0x47, 0x56, 0xa0, 0x61,
0xd6,
      0x62, 0xf5, 0x80, 0xff, 0x4d, 0xe4, 0x3b, 0x49, 0xfa, 0x82, 0xd8,
0x0a,
      0x4b, 0x80, 0xf8, 0x43, 0x4a]);
    mapOutputVectorsSHA3256[1]["output"] =
Uint8List.fromList([0x3a, 0x98, 0x5d,
      0xa7, 0x4f, 0xe2, 0x25, 0xb2, 0x04, 0x5c, 0x17, 0x2d, 0x6b, 0xd3,
0x90,
      0xbd, 0x85, 0x5f, 0x08, 0x6e, 0x3e, 0x9d, 0x52, 0x5b, 0x46, 0xbf,
0xe2,
      0x45, 0x11, 0x43, 0x15, 0x32]);
    mapOutputVectorsSHA3256[2]["output"] =
Uint8List.fromList([0x41, 0xc0, 0xdb,
      0xa2, 0xa9, 0xd6, 0x24, 0x08, 0x49, 0x10, 0x03, 0x76, 0xa8,
0x23, 0x5e,
      0x2c, 0x82, 0xe1, 0xb9, 0x99, 0x8a, 0x99, 0x9e, 0x21, 0xdb,
0x32, 0xdd,
      0x97, 0x49, 0x6d, 0x33, 0x76]);
    mapOutputVectorsSHA3256[3]["output"] =
Uint8List.fromList([0x91, 0x6f, 0x60,
      0x61, 0xfe, 0x87, 0x97, 0x41, 0xca, 0x64, 0x69, 0xb4, 0x39, 0x71,
0xdf,
      0xdb, 0x28, 0xb1, 0xa3, 0x2d, 0xc3, 0x6c, 0xb3, 0x25, 0x4e,
0x81, 0x2b,
      0xe2, 0x7a, 0xad, 0x1d, 0x18]);
    /*mapOutputVectorsSHA3256[4]["output"] =
Uint8List.fromList([0x5c, 0x88, 0x75,
      0xae, 0x47, 0x4a, 0x36, 0x34, 0xba, 0x4f, 0xd5, 0x5e, 0xc8, 0x5b,
0xff,
      0xd6, 0x61, 0xf3, 0x2a, 0xca, 0x75, 0xc6, 0xd6, 0x99, 0xd0, 0xcd,
0xcb,
      0x6c, 0x11, 0x58, 0x91, 0xc1]);
    mapOutputVectorsSHA3256[5]["output"] =
Uint8List.fromList([0xec, 0xbb, 0xc4,
      0x2c, 0xbf, 0x29, 0x66, 0x03, 0xac, 0xb2, 0xc6, 0xbc, 0x04, 0x10,
0xef,
      0x43, 0x78, 0xba, 0xfb, 0x24, 0xb7, 0x10, 0x35, 0x7f, 0x12, 0xdf,
0x60,
      0x77, 0x58, 0xb3, 0x3e, 0x2b]);*/
  }
}

class OutputVectorsSHA3384 {

  List<Map <String, dynamic>> mapOutputVectorsSHA3384 = [
    {
      "name" : "'empty'",
      "output" : null
    },
    {
      "name" : "'abc'",
      "output" : null
    },
    {
      "name"                                                    :
"'abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq'",
      "output" : null
    },
    {
      "name"                                                    :
"'abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmn
oijklmnopjklmnopqklmnopqrlmnopqrsmnopqrstnopqrstu'",
      "output" : null
    },
    /*{
      "name" : "'one million (1,000,000) repetitions of the character 'a'",
      "output" : null
    },
    {
      "name"      :      "'the      extremely-long      message
'abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmno
' repeated 16,777,216 times: a bit string of length 233 bits (1 GB)'",
      "output" : null
    }*/];

  OutputVectorsSHA3384() {
```

```
    mapOutputVectorsSHA3384[0]["output"] =
Uint8List.fromList([0x0c, 0x63, 0xa7,
      0x5b, 0x84, 0x5e, 0x4f, 0x7d, 0x01, 0x10, 0x7d, 0x85, 0x2e, 0x4c,
0x24,
      0x85, 0xc5, 0x1a, 0x50, 0xaa, 0xaa, 0x94, 0xfc, 0x61, 0x99, 0x5e,
0x71,
      0xbb, 0xee, 0x98, 0x3a, 0x2a, 0xc3, 0x71, 0x38, 0x31, 0x26, 0x4a,
0xdb,
      0x47, 0xfb, 0x6b, 0xd1, 0xe0, 0x58, 0xd5, 0xf0, 0x04]);
    mapOutputVectorsSHA3384[1]["output"] =
Uint8List.fromList([0xec, 0x01, 0x49,
      0x82, 0x88, 0x51, 0x6f, 0xc9, 0x26, 0x45, 0x9f, 0x58, 0xe2, 0xc6,
0xad,
      0x8d, 0xf9, 0xb4, 0x73, 0xcb, 0x0f, 0xc0, 0x8c, 0x25, 0x96, 0xda,
0x7c,
      0xf0, 0xe4, 0x9b, 0xe4, 0xb2, 0x98, 0xd8, 0x8c, 0xea, 0x92, 0x7a,
0xc7,
      0xf5, 0x39, 0xf1, 0xed, 0xf2, 0x28, 0x37, 0x6d, 0x25]);
    mapOutputVectorsSHA3384[2]["output"] =
Uint8List.fromList([0x99, 0x1c, 0x66,
      0x57, 0x55, 0xeb, 0x3a, 0x4b, 0x6b, 0xbd, 0xfb, 0x75, 0xc7, 0x8a,
0x49,
      0x2e, 0x8c, 0x56, 0xa2, 0x2c, 0x5c, 0x4d, 0x7e, 0x42, 0x9b, 0xfd,
0xbc,
      0x32, 0xb9, 0xd4, 0xad, 0x5a, 0xa0, 0x4a, 0x1f, 0x07, 0x6e, 0x62,
0xfe,
      0xa1, 0x9e, 0xef, 0x51, 0xac, 0xd0, 0x65, 0x7c, 0x22]);
    mapOutputVectorsSHA3384[3]["output"] =
Uint8List.fromList([0x79, 0x40, 0x7d,
      0x3b, 0x59, 0x16, 0xb5, 0x9c, 0x3e, 0x30, 0xb0, 0x98, 0x22,
0x97, 0x47,
      0x91, 0xc3, 0x13, 0xfb, 0x9e, 0xcc, 0x84, 0x9e, 0x40, 0x6f, 0x23,
0x59,
      0x2d, 0x04, 0xf6, 0x25, 0xdc, 0x8c, 0x70, 0x9b, 0x98, 0xb4, 0x3b,
0x38,
      0x52, 0xb3, 0x37, 0x21, 0x61, 0x79, 0xaa, 0x7f, 0xc7]);
    /*mapOutputVectorsSHA3384[4]["output"] =
Uint8List.fromList([0xee, 0xe9, 0xe2,
      0x4d, 0x78, 0xc1, 0x85, 0x53, 0x37, 0x98, 0x34, 0x51, 0xdf,
0x97, 0xc8,
      0xad, 0x9e, 0xed, 0xf2, 0x56, 0xc6, 0x33, 0x4f, 0x8e, 0x94, 0x8d,
0x25,
      0x2d, 0x5e, 0x0e, 0x76, 0x84, 0x7a, 0xa0, 0x77, 0x4d, 0xdb,
0x90, 0xa8,
      0x42, 0x19, 0x0d, 0x2c, 0x55, 0x8b, 0x4b, 0x83, 0x40]);
    mapOutputVectorsSHA3384[5]["output"] =
Uint8List.fromList([0xa0, 0x42, 0x96,
      0xf4, 0xfc, 0xaa, 0xe1, 0x48, 0x71, 0xbb, 0x5a, 0xd3, 0x3e, 0x28,
0xdc,
      0xf6, 0x92, 0x38, 0xb0, 0x42, 0x04, 0xd9, 0x94, 0x1b, 0x87,
0x82, 0xe8,
      0x16, 0xd0, 0x14, 0xbc, 0xb7, 0x54, 0x0e, 0x4a, 0xf5, 0x4f, 0x30,
0xd5,
      0x78, 0xf1, 0xa1, 0xca, 0x29, 0x30, 0x84, 0x7a, 0x12]);*/
  }
}

class OutputVectorsSHA3512 {

  List<Map <String, dynamic>> mapOutputVectorsSHA3512 = [
    {
      "name" : "'empty'",
      "output" : null
    },
    {
      "name" : "'abc'",
      "output" : null
    },
    {
      "name"                                                    :
"'abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq'",
      "output" : null
    },
    {
      "name"                                                    :
"'abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmn
oijklmnopjklmnopqklmnopqrlmnopqrsmnopqrstnopqrstu'",
      "output" : null
    },
    /*{
      "name" : "'one million (1,000,000) repetitions of the character 'a''",
      "output" : null
    },
    {
      "name"      :      "'the      extremely-long      message
'abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmno
' repeated 16,777,216 times: a bit string of length 233 bits (1 GB)'",
      "output" : null
    }*/];

  OutputVectorsSHA3512() {
    mapOutputVectorsSHA3512[0]["output"] =
Uint8List.fromList([0xa6, 0x9f, 0x73,
      0xcc, 0xa2, 0x3a, 0x9a, 0xc5, 0xc8, 0xb5, 0x67, 0xdc, 0x18, 0x5a,
0x75,
      0x6e, 0x97, 0xc9, 0x82, 0x16, 0x4f, 0xe2, 0x58, 0x59, 0xe0, 0xd1,
0xdc,
      0xc1, 0x47, 0x5c, 0x80, 0xa6, 0x15, 0xb2, 0x12, 0x3a, 0xf1, 0xf5,
0xf9,
      0x4c, 0x11, 0xe3, 0xe9, 0x40, 0x2c, 0x3a, 0xc5, 0x58, 0xf5, 0x00,
0x19,
      0x9d, 0x95, 0xb6, 0xd3, 0xe3, 0x01, 0x75, 0x85, 0x86, 0x28,
0x1d, 0xcd,
      0x26]);
    mapOutputVectorsSHA3512[1]["output"] =
Uint8List.fromList([0xb7, 0x51, 0x85,
      0x0b, 0x1a, 0x57, 0x16, 0x8a, 0x56, 0x93, 0xcd, 0x92, 0x4b,
0x6b, 0x09,
      0x6e, 0x08, 0xf6, 0x21, 0x82, 0x74, 0x44, 0xf7, 0x0d, 0x88, 0x4f,
0x5d,
      0x02, 0x40, 0xd2, 0x71, 0x2e, 0x10, 0xe1, 0x16, 0xe9, 0x19,
0x2a, 0xf3,
      0xc9, 0x1a, 0x7e, 0xc5, 0x76, 0x47, 0xe3, 0x93, 0x40, 0x57, 0x34,
0x0b,
      0x4c, 0xf4, 0x08, 0xd5, 0xa5, 0x65, 0x92, 0xf8, 0x27, 0x4e, 0xec,
0x53,
      0xf0]);
    mapOutputVectorsSHA3512[2]["output"] =
Uint8List.fromList([0x04, 0xa3, 0x71,
      0xe8, 0x4e, 0xcf, 0xb5, 0xb8, 0xb7, 0x7c, 0xb4, 0x86, 0x10, 0xfc,
0xa8,
      0x18, 0x2d, 0xd4, 0x57, 0xce, 0x6f, 0x32, 0x6a, 0x0f, 0xd3, 0xd7,
0xec,
      0x2f, 0x1e, 0x91, 0x63, 0x6d, 0xee, 0x69, 0x1f, 0xbe, 0x0c, 0x98,
0x53,
      0x02, 0xba, 0x1b, 0x0d, 0x8d, 0xc7, 0x8c, 0x08, 0x63, 0x46,
0xb5, 0x33,
      0xb4, 0x9c, 0x03, 0x0d, 0x99, 0xa2, 0x7d, 0xaf, 0x11, 0x39, 0xd6,
0xe7,
      0x5e]);
    mapOutputVectorsSHA3512[3]["output"] =
Uint8List.fromList([0xaf, 0xeb, 0xb2,
      0xef, 0x54, 0x2e, 0x65, 0x79, 0xc5, 0x0c, 0xad, 0x06, 0xd2, 0xe5,
0x78,
      0xf9, 0xf8, 0xdd, 0x68, 0x81, 0xd7, 0xdc, 0x82, 0x4d, 0x26, 0x36,
0x0f,
      0xee, 0xbf, 0x18, 0xa4, 0xfa, 0x73, 0xe3, 0x26, 0x11, 0x22, 0x94,
0x8e,
      0xfc, 0xfd, 0x49, 0x2e, 0x74, 0xe8, 0x2e, 0x21, 0x89, 0xed, 0x0f,
0xb4,
      0x40, 0xd1, 0x87, 0xf3, 0x82, 0x27, 0x0c, 0xb4, 0x55, 0xf2, 0x1d,
0xd1,
      0x85]);
    /*mapOutputVectorsSHA3512[4]["output"] =
Uint8List.fromList([0x3c, 0x3a, 0x87,
      0x6d, 0xa1, 0x40, 0x34, 0xab, 0x60, 0x62, 0x7c, 0x07, 0x7b,
0xb9, 0x8f,
      0x7e, 0x12, 0x0a, 0x2a, 0x53, 0x70, 0x21, 0x2d, 0xff, 0xb3, 0x38,
0x5a,
      0x18, 0xd4, 0xf3, 0x88, 0x59, 0xed, 0x31, 0x1d, 0x0a, 0x9d, 0x51,
0x41,
      0xce, 0x9c, 0xc5, 0xc6, 0x6e, 0xe6, 0x89, 0xb2, 0x66, 0xa8, 0xaa,
0x18,
      0xac, 0xe8, 0x28, 0x2a, 0x0e, 0x0d, 0xb5, 0x96, 0xc9, 0x0b, 0x0a,
0x7b,
      0x87]);
    mapOutputVectorsSHA3512[5]["output"] =
Uint8List.fromList([0x23, 0x5f, 0xfd,
```

```
    0x53, 0x50, 0x4e, 0xf8, 0x36, 0xa1, 0x34, 0x2b, 0x48, 0x8f, 0x48,
0x3b,
    0x39, 0x6e, 0xab, 0xbf, 0xe6, 0x42, 0xcf, 0x78, 0xee, 0x0d, 0x31,
0xfe,
    0xec, 0x78, 0x8b, 0x23, 0xd0, 0xd1, 0x8d, 0x5c, 0x33, 0x95,
0x50, 0xdd,
    0x59, 0x58, 0xa5, 0x00, 0xd4, 0xb9, 0x53, 0x63, 0xda, 0x1b, 0x5f,
0xa1,
    0x8a, 0xff, 0xc1, 0xba, 0xb2, 0x29, 0x2d, 0xc6, 0x3b, 0x7d, 0x85,
0x09,
    0x7c]);*/
  }
}

class InputVectorsRIPEMD128160256320 {

 List<Map                    <String,                    dynamic>>
mapInputVectorsRIPEMD128160256320 = [
  {
    "name" : "'empty'",
    "input" : null
  },
  {
    "name" : "'a'",
    "input" : null
  },
  {
    "name" : "'abc'",
    "input" : null
  },
  {
    "name" : "'message digest'",
    "input" : null
  },
  {
    "name" : "'abcdefghijklmnopqrstuvwxyz'",
    "input" : null
  },
  {
    "name"                                                          :
"'abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq'",
    "input" : null
  },
  {
    "name"                                                          :
"'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuv
wxyz0123456789'",
    "input" : null
  },
  {
    "name" : "'8 times '1234567890'",
    "input" : null
  },
  /*{
    "name" : "'1 million times 'a'",
    "input" : null
  }*/];

 InputVectorsRIPEMD128160256320() {
  mapInputVectorsRIPEMD128160256320[0]["input"]                =
Uint8List.fromList([]);
  mapInputVectorsRIPEMD128160256320[1]["input"]                =
Uint8List.fromList([0x61]);
  mapInputVectorsRIPEMD128160256320[2]["input"]                =
Uint8List.fromList([0x61, 0x62, 0x63]);
  mapInputVectorsRIPEMD128160256320[3]["input"]                =
Uint8List.fromList([0x6d, 0x65,
    0x73, 0x73, 0x61, 0x67, 0x65, 0x20, 0x64, 0x69, 0x67, 0x65,
0x73, 0x74]);
  mapInputVectorsRIPEMD128160256320[4]["input"]                =
Uint8List.fromList([0x61, 0x62,
    0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, 0x6a, 0x6b, 0x6c,
0x6d, 0x6e,
    0x6f, 0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x78,
0x79, 0x7a]);
  mapInputVectorsRIPEMD128160256320[5]["input"]                =
Uint8List.fromList([0x61, 0x62,
    0x63, 0x64, 0x62, 0x63, 0x64, 0x65, 0x63, 0x64, 0x65, 0x66,
0x64, 0x65,
    0x66, 0x67, 0x65, 0x66, 0x67, 0x68, 0x66, 0x67, 0x68, 0x69,
0x67, 0x68,
    0x69, 0x6a, 0x68, 0x69, 0x6a, 0x6b, 0x69, 0x6a, 0x6b, 0x6c, 0x6a,
0x6b,
    0x6c, 0x6d, 0x6b, 0x6c, 0x6d, 0x6e, 0x6c, 0x6d, 0x6e, 0x6f, 0x6d,
0x6e,
    0x6f, 0x70, 0x6e, 0x6f, 0x70, 0x71]);
  mapInputVectorsRIPEMD128160256320[6]["input"]                =
Uint8List.fromList([0x41, 0x42,
    0x43, 0x44, 0x45, 0x46, 0x47, 0x48, 0x49, 0x4a, 0x4b, 0x4c,
0x4d, 0x4e,
    0x4f, 0x50, 0x51, 0x52, 0x53, 0x54, 0x55, 0x56, 0x57, 0x58,
0x59, 0x5a,
    0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, 0x6a,
0x6b, 0x6c,
    0x6d, 0x6e, 0x6f, 0x70, 0x71, 0x72, 0x73, 0x74, 0x75, 0x76,
0x77, 0x78,
    0x79, 0x7a, 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
0x38, 0x39]);

  List <int> time8 = [];
  List <int> timeFor8 = [0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
0x38,
    0x39, 0x30];
  for(int i = 0; i < 8; i++){
    time8 += timeFor8;
  }

  mapInputVectorsRIPEMD128160256320[7]["input"]                =
Uint8List.fromList(time8);

  /*List <int> millionA = [];
  for(int i = 0; i < 1000000; i++){
    millionA.add(0x61);
  }
  mapInputVectorsRIPEMD128160256320[8]["input"]                =
Uint8List.fromList(millionA);*/
 }

}

class OutputVectorsRIPEMD128 {

 List<Map <String, dynamic>> mapOutputVectorsRIPEMD128 = [
  {
    "name" : "'empty'",
    "output" : null
  },
  {
    "name" : "'a'",
    "output" : null
  },
  {
    "name" : "'abc'",
    "output" : null
  },
  {
    "name" : "'message digest'",
    "output" : null
  },
  {
    "name" : "'abcdefghijklmnopqrstuvwxyz'",
    "output" : null
  },
  {
    "name"                                                          :
"'abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq'",
    "output" : null
  },
  {
    "name"                                                          :
"'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuv
wxyz0123456789'",
    "output" : null
  },
  {
    "name" : "'8 times '1234567890'",
    "input" : null
  },
```

```
  /* {
    "name" : "'1 million times 'a'",
    "output" : null
  }*/];

 OutputVectorsRIPEMD128() {
   mapOutputVectorsRIPEMD128[0]["output"]                    =
Uint8List.fromList([0xcd, 0xf2,
    0x62, 0x13, 0xa1, 0x50, 0xdc, 0x3e, 0xcb, 0x61, 0x0f, 0x18, 0xf6,
0xb3,
    0x8b, 0x46]);
   mapOutputVectorsRIPEMD128[1]["output"]                    =
Uint8List.fromList([0x86, 0xbe,
    0x7a, 0xfa, 0x33, 0x9d, 0x0f, 0xc7, 0xcf, 0xc7, 0x85, 0xe7, 0x2f,
0x57,
    0x8d, 0x33]);
   mapOutputVectorsRIPEMD128[2]["output"]                    =
Uint8List.fromList([0xc1, 0x4a,
    0x12, 0x19, 0x9c, 0x66, 0xe4, 0xba, 0x84, 0x63, 0x6b, 0x0f, 0x69,
0x14,
    0x4c, 0x77]);
   mapOutputVectorsRIPEMD128[3]["output"]                    =
Uint8List.fromList([0x9e, 0x32,
    0x7b, 0x3d, 0x6e, 0x52, 0x30, 0x62, 0xaf, 0xc1, 0x13, 0x2d, 0x7d,
    0xf9, 0xd1, 0xb8]);
   mapOutputVectorsRIPEMD128[4]["output"]                    =
Uint8List.fromList([0xfd, 0x2a,
    0xa6, 0x07, 0xf7, 0x1d, 0xc8, 0xf5, 0x10, 0x71, 0x49, 0x22, 0xb3,
0x71,
    0x83, 0x4e]);
   mapOutputVectorsRIPEMD128[5]["output"]                    =
Uint8List.fromList([0xa1, 0xaa,
    0x06, 0x89, 0xd0, 0xfa, 0xfa, 0x2d, 0xdc, 0x22, 0xe8, 0x8b, 0x49,
0x13,
    0x3a, 0x06]);
   mapOutputVectorsRIPEMD128[6]["output"]                    =
Uint8List.fromList([0xd1, 0xe9,
    0x59, 0xeb, 0x17, 0x9c, 0x91, 0x1f, 0xae, 0xa4, 0x62, 0x4c, 0x60,
0xc5,
    0xc7, 0x02]);
   mapOutputVectorsRIPEMD128[7]["output"]                    =
Uint8List.fromList([0x3f, 0x45,
    0xef, 0x19, 0x47, 0x32, 0xc2, 0xdb, 0xb2, 0xc4, 0xa2, 0xc7, 0x69,
0x79,
    0x5f, 0xa3]);
   /*mapOutputVectorsRIPEMD128[8]["output"]                    =
Uint8List.fromList([0x4a, 0x7f,
    0x57, 0x23, 0xf9, 0x54, 0xeb, 0xa1, 0x21, 0x6c, 0x9d, 0x8f, 0x63,
0x20,
    0x43, 0x1f]);*/
  }
}

class OutputVectorsRIPEMD160 {

 List<Map <String, dynamic>> mapOutputVectorsRIPEMD160 = [
   {
    "name" : "'empty'",
    "output" : null
   },
   {
    "name" : "'a'",
    "output" : null
   },
   {
    "name" : "'abc'",
    "output" : null
   },
   {
    "name" : "'message digest'",
    "output" : null
   },
   {
    "name" : "'abcdefghijklmnopqrstuvwxyz'",
    "output" : null
   },
   {
    "name"                                                    :
"'abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq'",
    "output" : null
   },
   {
    "name"                                                    :
"'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuv
wxyz0123456789'",
    "output" : null
   },
   {
    "name" : "'8 times '1234567890'",
    "input" : null
   },
   /*{
    "name" : "'1 million times 'a'",
    "output" : null
   }*/];

 OutputVectorsRIPEMD160() {
   mapOutputVectorsRIPEMD160[0]["output"]                    =
Uint8List.fromList([0x9c, 0x11,
    0x85, 0xa5, 0xc5, 0xe9, 0xfc, 0x54, 0x61, 0x28, 0x08, 0x97, 0x7e,
0xe8,
    0xf5, 0x48, 0xb2, 0x25, 0x8d, 0x31]);
   mapOutputVectorsRIPEMD160[1]["output"]                    =
Uint8List.fromList([0x0b, 0xdc,
    0x9d, 0x2d, 0x25, 0x6b, 0x3e, 0xe9, 0xda, 0xae, 0x34, 0x7b, 0xe6,
0xf4,
    0xdc, 0x83, 0x5a, 0x46, 0x7f, 0xfe]);
   mapOutputVectorsRIPEMD160[2]["output"]                    =
Uint8List.fromList([0x8e, 0xb2,
    0x08, 0xf7, 0xe0, 0x5d, 0x98, 0x7a, 0x9b, 0x04, 0x4a, 0x8e, 0x98,
0xc6,
    0xb0, 0x87, 0xf1, 0x5a, 0x0b, 0xfc]);
   mapOutputVectorsRIPEMD160[3]["output"]                    =
Uint8List.fromList([0x5d, 0x06,
    0x89, 0xef, 0x49, 0xd2, 0xfa, 0xe5, 0x72, 0xb8, 0x81, 0xb1, 0x23,
0xa8,
    0x5f, 0xfa, 0x21, 0x59, 0x5f, 0x36]);
   mapOutputVectorsRIPEMD160[4]["output"]                    =
Uint8List.fromList([0xf7, 0x1c,
    0x27, 0x10, 0x9c, 0x69, 0x2c, 0x1b, 0x56, 0xbb, 0xdc, 0xeb,
0x5b, 0x9d,
    0x28, 0x65, 0xb3, 0x70, 0x8d, 0xbc]);
   mapOutputVectorsRIPEMD160[5]["output"]                    =
Uint8List.fromList([0x12, 0xa0,
    0x53, 0x38, 0x4a, 0x9c, 0x0c, 0x88, 0xe4, 0x05, 0xa0, 0x6c, 0x27,
0xdc,
    0xf4, 0x9a, 0xda, 0x62, 0xeb, 0x2b]);
   mapOutputVectorsRIPEMD160[6]["output"]                    =
Uint8List.fromList([0xb0, 0xe2,
    0x0b, 0x6e, 0x31, 0x16, 0x64, 0x02, 0x86, 0xed, 0x3a, 0x87,
0xa5, 0x71,
    0x30, 0x79, 0xb2, 0x1f, 0x51, 0x89]);
   mapOutputVectorsRIPEMD160[7]["output"]                    =
Uint8List.fromList([0x9b, 0x75,
    0x2e, 0x45, 0x57, 0x3d, 0x4b, 0x39, 0xf4, 0xdb, 0xd3, 0x32, 0x3c,
0xab,
    0x82, 0xbf, 0x63, 0x32, 0x6b, 0xfb]);
   /*mapOutputVectorsRIPEMD160[8]["output"]                    =
Uint8List.fromList([0x52, 0x78,
    0x32, 0x43, 0xc1, 0x69, 0x7b, 0xdb, 0xe1, 0x6d, 0x37, 0xf9, 0x7f,
0x68,
    0xf0, 0x83, 0x25, 0xdc, 0x15, 0x28]);*/
  }
}

class OutputVectorsRIPEMD256 {

 List<Map <String, dynamic>> mapOutputVectorsRIPEMD256 = [
   {
    "name" : "'empty'",
    "output" : null
   },
   {
    "name" : "'a'",
    "output" : null
   },
   {
    "name" : "'abc'",
```

```
    "output" : null
   },
   {
    "name" : "'message digest'",
    "output" : null
   },
   {
    "name" : "'abcdefghijklmnopqrstuvwxyz'",
    "output" : null
   },
   {
    "name"                                                    :
"'abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq'",
    "output" : null
   },
   {
    "name"                                                    :
"'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuv
wxyz0123456789'",
    "output" : null
   },
   {
    "name" : "'8 times '1234567890'",
    "input" : null
   },
   /*{
    "name" : "'1 million times 'a'",
    "output" : null
   }*/];

  OutputVectorsRIPEMD256() {
    mapOutputVectorsRIPEMD256[0]["output"]                    =
Uint8List.fromList([0x02, 0xba,
     0x4c, 0x4e, 0x5f, 0x8e, 0xcd, 0x18, 0x77, 0xfc, 0x52, 0xd6, 0x4d,
0x30,
     0xe3, 0x7a, 0x2d, 0x97, 0x74, 0xfb, 0x1e, 0x5d, 0x02, 0x63, 0x80,
0xae,
     0x01, 0x68, 0xe3, 0xc5, 0x52, 0x2d]);
    mapOutputVectorsRIPEMD256[1]["output"]                    =
Uint8List.fromList([0xf9, 0x33,
     0x3e, 0x45, 0xd8, 0x57, 0xf5, 0xd9, 0x0a, 0x91, 0xba, 0xb7, 0x0a,
0x1e,
     0xba, 0x0c, 0xfb, 0x1b, 0xe4, 0xb0, 0x78, 0x3c, 0x9a, 0xcf, 0xcd,
0x88,
     0x3a, 0x91, 0x34, 0x69, 0x29, 0x25]);
    mapOutputVectorsRIPEMD256[2]["output"]                    =
Uint8List.fromList([0xaf, 0xbd,
     0x6e, 0x22, 0x8b, 0x9d, 0x8c, 0xbb, 0xce, 0xf5, 0xca, 0x2d, 0x03,
0xe6,
     0xdb, 0xa1, 0x0a, 0xc0, 0xbc, 0x7d, 0xcb, 0xe4, 0x68, 0x0e, 0x1e,
0x42,
     0xd2, 0xe9, 0x75, 0x45, 0x9b, 0x65]);
    mapOutputVectorsRIPEMD256[3]["output"]                    =
Uint8List.fromList([0x87, 0xe9,
     0x71, 0x75, 0x9a, 0x1c, 0xe4, 0x7a, 0x51, 0x4d, 0x5c, 0x91, 0x4c,
0x39,
     0x2c, 0x90, 0x18, 0xc7, 0xc4, 0x6b, 0xc1, 0x44, 0x65, 0x55, 0x4a,
0xfc,
     0xdf, 0x54, 0xa5, 0x07, 0x0c, 0x0e]);
    mapOutputVectorsRIPEMD256[4]["output"]                    =
Uint8List.fromList([0x64, 0x9d,
     0x30, 0x34, 0x75, 0x1e, 0xa2, 0x16, 0x77, 0x6b, 0xf9, 0xa1, 0x8a,
0xcc,
     0x81, 0xbc, 0x78, 0x96, 0x11, 0x8a, 0x51, 0x97, 0x96, 0x87,
0x82, 0xdd,
     0x1f, 0xd9, 0x7d, 0x8d, 0x51, 0x33]);
    mapOutputVectorsRIPEMD256[5]["output"]                    =
Uint8List.fromList([0x38, 0x43,
     0x04, 0x55, 0x83, 0xaa, 0xc6, 0xc8, 0xc8, 0xd9, 0x12, 0x85, 0x73,
0xe7,
     0xa9, 0x80, 0x9a, 0xfb, 0x2a, 0x0f, 0x34, 0xcc, 0xc3, 0x6e, 0xa9,
0xe7,
     0x2f, 0x16, 0xf6, 0x36, 0x8e, 0x3f]);
    mapOutputVectorsRIPEMD256[6]["output"]                    =
Uint8List.fromList([0x57, 0x40,
     0xa4, 0x08, 0xac, 0x16, 0xb7, 0x20, 0xb8, 0x44, 0x24, 0xae, 0x93,
0x1c,
     0xbb, 0x1f, 0xe3, 0x63, 0xd1, 0xd0, 0xbf, 0x40, 0x17, 0xf1, 0xa8,
0x9f,
     0x7e, 0xa6, 0xde, 0x77, 0xa0, 0xb8]);
    mapOutputVectorsRIPEMD256[7]["output"]                    =
Uint8List.fromList([0x06, 0xfd,
     0xcc, 0x7a, 0x40, 0x95, 0x48, 0xaa, 0xf9, 0x13, 0x68, 0xc0, 0x6a,
0x62,
     0x75, 0xb5, 0x53, 0xe3, 0xf0, 0x99, 0xbf, 0x0e, 0xa4, 0xed, 0xfd,
0x67,
     0x78, 0xdf, 0x89, 0xa8, 0x90, 0xdd]);
    /*mapOutputVectorsRIPEMD256[8]["output"]                    =
Uint8List.fromList([0xac, 0x95,
     0x37, 0x44, 0xe1, 0x0e, 0x31, 0x51, 0x4c, 0x15, 0x0d, 0x4d,
0x8d, 0x7b,
     0x67, 0x73, 0x42, 0xe3, 0x33, 0x99, 0x78, 0x82, 0x96, 0xe4,
0x3a, 0xe4,
     0x85, 0x0c, 0xe4, 0xf9, 0x79, 0x78]);*/
  }
}

class OutputVectorsRIPEMD320 {

  List<Map <String, dynamic>> mapOutputVectorsRIPEMD320 = [
   {
    "name" : "'empty'",
    "output" : null
   },
   {
    "name" : "'a'",
    "output" : null
   },
   {
    "name" : "'abc'",
    "output" : null
   },
   {
    "name" : "'message digest'",
    "output" : null
   },
   {
    "name" : "'abcdefghijklmnopqrstuvwxyz'",
    "output" : null
   },
   {
    "name"                                                    :
"'abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq'",
    "output" : null
   },
   {
    "name"                                                    :
"'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuv
wxyz0123456789'",
    "output" : null
   },
   {
    "name" : "'8 times '1234567890'",
    "input" : null
   },
   /*{
    "name" : "'1 million times 'a'",
    "output" : null
   }*/];

  OutputVectorsRIPEMD320() {
    mapOutputVectorsRIPEMD320[0]["output"]                    =
Uint8List.fromList([0x22, 0xd6,
     0x5d, 0x56, 0x61, 0x53, 0x6c, 0xdc, 0x75, 0xc1, 0xfd, 0xf5, 0xc6,
0xde,
     0x7b, 0x41, 0xb9, 0xf2, 0x73, 0x25, 0xeb, 0xc6, 0x1e, 0x85, 0x57,
0x17,
     0x7d, 0x70, 0x5a, 0x0e, 0xc8, 0x80, 0x15, 0x1c, 0x3a, 0x32, 0xa0,
0x08,
     0x99, 0xb8]);
    mapOutputVectorsRIPEMD320[1]["output"]                    =
Uint8List.fromList([0xce, 0x78,
     0x85, 0x06, 0x38, 0xf9, 0x26, 0x58, 0xa5, 0xa5, 0x85, 0x09, 0x75,
0x79,
     0x92, 0x6d, 0xda, 0x66, 0x7a, 0x57, 0x16, 0x56, 0x2c, 0xfc, 0xf6,
0xfb,
     0xe7, 0x7f, 0x63, 0x54, 0x2f, 0x99, 0xb0, 0x47, 0x05, 0xd6, 0x97,
0x0d,
```

```
    0xff, 0x5d]);
  mapOutputVectorsRIPEMD320[2]["output"] =
Uint8List.fromList([0xde, 0x4c,
    0x01, 0xb3, 0x05, 0x4f, 0x89, 0x30, 0xa7, 0x9d, 0x09, 0xae, 0x73,
0x8e,
    0x92, 0x30, 0x1e, 0x5a, 0x17, 0x08, 0x5b, 0xef, 0xfd, 0xc1, 0xb8,
0xd1,
    0x16, 0x71, 0x3e, 0x74, 0xf8, 0x2f, 0xa9, 0x42, 0xd6, 0x4c, 0xdb,
0xc4,
    0x68, 0x2d]);
  mapOutputVectorsRIPEMD320[3]["output"] =
Uint8List.fromList([0x3a, 0x8e,
    0x28, 0x50, 0x2e, 0xd4, 0x5d, 0x42, 0x2f, 0x68, 0x84, 0x4f, 0x9d,
0xd3,
    0x16, 0xe7, 0xb9, 0x85, 0x33, 0xfa, 0x3f, 0x2a, 0x91, 0xd2, 0x9f,
0x84,
    0xd4, 0x25, 0xc8, 0x8d, 0x6b, 0x4e, 0xff, 0x72, 0x7d, 0xf6, 0x6a,
0x7c,
    0x01, 0x97]);
  mapOutputVectorsRIPEMD320[4]["output"] =
Uint8List.fromList([0xca, 0xbd,
    0xb1, 0x81, 0x0b, 0x92, 0x47, 0x0a, 0x20, 0x93, 0xaa, 0x6b, 0xce,
0x05,
    0x95, 0x2c, 0x28, 0x34, 0x8c, 0xf4, 0x3f, 0xf6, 0x08, 0x41, 0x97,
0x51,
    0x66, 0xbb, 0x40, 0xed, 0x23, 0x40, 0x04, 0xb8, 0x82, 0x44,
0x63, 0xe6,
    0xb0, 0x09]);
  mapOutputVectorsRIPEMD320[5]["output"] =
Uint8List.fromList([0xd0, 0x34,
    0xa7, 0x95, 0x0c, 0xf7, 0x22, 0x02, 0x1b, 0xa4, 0xb8, 0x4d, 0xf7,
0x69,
    0xa5, 0xde, 0x20, 0x60, 0xe2, 0x59, 0xdf, 0x4c, 0x9b, 0xb4, 0xa4,
0x26,
    0x8c, 0x0e, 0x93, 0x5b, 0xbc, 0x74, 0x70, 0xa9, 0x69, 0xc9, 0xd0,
0x72,
    0xa1, 0xac]);
  mapOutputVectorsRIPEMD320[6]["output"] =
Uint8List.fromList([0xed, 0x54,
    0x49, 0x40, 0xc8, 0x6d, 0x67, 0xf2, 0x50, 0xd2, 0x32, 0xc3, 0x0b,
0x7b,
    0x3e, 0x57, 0x70, 0xe0, 0xc6, 0x0c, 0x8c, 0xb9, 0xa4, 0xca, 0xfe,
0x3b,
    0x11, 0x38, 0x8a, 0xf9, 0x92, 0x0e, 0x1b, 0x99, 0x23, 0x0b, 0x84,
0x3c,
    0x86, 0xa4]);
  mapOutputVectorsRIPEMD320[7]["output"] =
Uint8List.fromList([0x55, 0x78,
    0x88, 0xaf, 0x5f, 0x6d, 0x8e, 0xd6, 0x2a, 0xb6, 0x69, 0x45, 0xc6,
0xd2,
    0xa0, 0xa4, 0x7e, 0xcd, 0x53, 0x41, 0xe9, 0x15, 0xeb, 0x8f, 0xea,
0x1d,
    0x05, 0x24, 0x95, 0x5f, 0x82, 0x5d, 0xc7, 0x17, 0xe4, 0xa0, 0x08,
0xab,
    0x2d, 0x42]);
  /*mapOutputVectorsRIPEMD320[8]["output"] =
Uint8List.fromList([0xbd, 0xee,
    0x37, 0xf4, 0x37, 0x1e, 0x20, 0x64, 0x6b, 0x8b, 0x0d, 0x86,
0x2d, 0xda,
    0x16, 0x29, 0x2a, 0xe3, 0x6f, 0x40, 0x96, 0x5e, 0x8c, 0x85, 0x09,
0xe6,
    0x3d, 0x1d, 0xbd, 0xde, 0xcc, 0x50, 0x3e, 0x2b, 0x63, 0xeb, 0x92,
0x45,
    0xbb, 0x66]);*/
 }
}
```

hashTests.dart

```
import 'dart:typed_data';
import 'dart:math';
import 'package:grizzly_distuv/math.dart';
import 'package:flutter_test/flutter_test.dart';
import 'package:flutter/foundation.dart';

import 'package:crypto_dart_tests/hash/hashToTest.dart';
import 'package:crypto_dart_tests/hash/vectorsTest.dart';


void run() {
 HashFunctionsCollection collection = new HashFunctionsCollection
();
 for(int i = 0; i < collection.mapHashFunctions.length; i++) {
  group(collection.mapHashFunctions[i]["name"] + " testing: ", () {
   test("Monobit test", () {
    int n = 10 * collection.mapHashFunctions[i]["outputLength"];

    Uint8List hashData =
getOutputFromHashFunction(collection.mapHashFunctions[i]["diges
t"],
      n,
      collection.mapHashFunctions[i]["outputLength"]
    );

    double Pvalue = computePvalue_MonobitTest(hashData, n);
    expect(Pvalue >= 0.01, true, reason: "The actual P-value value is
" + Pvalue.toString());
   });

   test("Frequency Test within a 8-bit Block", () {

    int n = 100 * collection.mapHashFunctions[i]["outputLength"];

    Uint8List hashData =
getOutputFromHashFunction(collection.mapHashFunctions[i]["diges
t"],
      n,
      collection.mapHashFunctions[i]["outputLength"]
    );

    double Pvalue =
computePvalue_FrequencyTestWithinABlock(hashData, 8);

    expect(Pvalue >= 0.01, true, reason: "The actual P-value value is
" + Pvalue.toString());

   });

   test("Frequency Test within a 16-bit Block", () {

    int n = 200 * collection.mapHashFunctions[i]["outputLength"];

    Uint8List hashData =
getOutputFromHashFunction(collection.mapHashFunctions[i]["diges
t"],
      n,
      collection.mapHashFunctions[i]["outputLength"]
    );

    double Pvalue =
computePvalue_FrequencyTestWithinABlock(hashData, 16);

    expect(Pvalue >= 0.01, true, reason: "The actual P-value value is
" + Pvalue.toString());

   });

   test("Runs Test", () {
    //Monobit test
    int n = 100 * collection.mapHashFunctions[i]["outputLength"];

    Uint8List hashData =
getOutputFromHashFunction(collection.mapHashFunctions[i]["diges
t"],
      n,
      collection.mapHashFunctions[i]["outputLength"]
    );

    double Pvalue = computePvalue_RunsTest(hashData, n);
    expect(Pvalue >= 0.01, true, reason: "The actual P-value value is
" + Pvalue.toString());
   });

   test("Test for the Longest Run of Ones in a Block", () {
    int n  = collection.mapHashFunctions[i]["outputLength"];

    Uint8List hashData =
getOutputFromHashFunction(collection.mapHashFunctions[i]["diges
t"],
```

```
        n,
        collection.mapHashFunctions[i]["outputLength"]
      );

      double Pvalue =
computePvalue_TestForTheLongestRunOfOnesInABlock(hashData,
n);
      expect(Pvalue >= 0.01, true, reason: "The actual P-value value is
" + Pvalue.toString());
    });

    test("Avalanche effect test", () {

      List <int> input = [];
      for(int i = 0; i < 1024; i++){
        input.add(0x61);
      }
      Uint8List inp = Uint8List.fromList(input);


      if(i == 12){
        double Pvalue =
computeProbability_TestForTheAvalancheEffect(inp,
            collection.mapHashFunctions[i]["digest"]);
        expect(Pvalue >= 0.01, true, reason: "The actual P-value value
is "
            + Pvalue.toString(), skip: "The test takes 1 hour");
      }else{
        double Pvalue =
computeProbability_TestForTheAvalancheEffect(inp,
            collection.mapHashFunctions[i]["digest"]);
        expect(Pvalue >= 0.01, true, reason: "The actual P-value value
is "
            + Pvalue.toString());
      }

    });


  });
 }
 group(collection.mapHashFunctions[0]["name"] + " testing: ", ()
//SHA-224
 {
  test("Test Vectors", () {

    InputVectorsSHA2SHA3 inputVectorsSHA2SHA3 =
InputVectorsSHA2SHA3();
    OutputVectorsSHA224 outputVectorsSHA224 =
OutputVectorsSHA224();

    var digest = [];
    for(int i = 0; i <
inputVectorsSHA2SHA3.mapInputVectorsSHA2SHA3.length; i++)
{
      digest = collection.mapHashFunctions[0]["digest"].process(

inputVectorsSHA2SHA3.mapInputVectorsSHA2SHA3[i]["input"]);
      expect(compareArrays(
        digest,
outputVectorsSHA224.mapOutputVectorsSHA224[i]["output"]),
          true, reason: "The actual digest value is "
            +
inputVectorsSHA2SHA3.mapInputVectorsSHA2SHA3[i]["name"]
            + digest.toString() + "                Vector " +

outputVectorsSHA224.mapOutputVectorsSHA224[i]["output"]
              .toString());
    }
  });
 }); //SHA-224 vector test

 group(collection.mapHashFunctions[1]["name"] + " testing: ", ()
//SHA-256
 {
  test("Test Vectors", () {

    InputVectorsSHA2SHA3 inputVectorsSHA2SHA3 =
InputVectorsSHA2SHA3();
    OutputVectorsSHA256 outputVectorsSHA256 =
OutputVectorsSHA256();

    var digest = [];
    for(int i = 0; i <
inputVectorsSHA2SHA3.mapInputVectorsSHA2SHA3.length; i++)
{
      digest = collection.mapHashFunctions[1]["digest"].process(

inputVectorsSHA2SHA3.mapInputVectorsSHA2SHA3[i]["input"]);
      expect(compareArrays(
        digest,
outputVectorsSHA256.mapOutputVectorsSHA256[i]["output"]),
          true, reason: "The actual digest value is "
            +
inputVectorsSHA2SHA3.mapInputVectorsSHA2SHA3[i]["name"]
            + digest.toString() + "                Vector " +

outputVectorsSHA256.mapOutputVectorsSHA256[i]["output"]
              .toString());
    }
  });
 }); //SHA-256 vector test

 group(collection.mapHashFunctions[2]["name"] + " testing: ", ()
//SHA-384
 {
  test("Test Vectors", () {

    InputVectorsSHA2SHA3 inputVectorsSHA2SHA3 =
InputVectorsSHA2SHA3();
    OutputVectorsSHA384 outputVectorsSHA384 =
OutputVectorsSHA384();

    var digest = [];
    for(int i = 0; i <
inputVectorsSHA2SHA3.mapInputVectorsSHA2SHA3.length; i++)
{
      digest = collection.mapHashFunctions[2]["digest"].process(

inputVectorsSHA2SHA3.mapInputVectorsSHA2SHA3[i]["input"]);
      expect(compareArrays(
        digest,
outputVectorsSHA384.mapOutputVectorsSHA384[i]["output"]),
          true, reason: "The actual digest value is "
            +
inputVectorsSHA2SHA3.mapInputVectorsSHA2SHA3[i]["name"]
            + digest.toString() + "                Vector " +

outputVectorsSHA384.mapOutputVectorsSHA384[i]["output"]
              .toString());
    }
  });
 }); //SHA-384 vector test

 group(collection.mapHashFunctions[3]["name"] + " testing: ", ()
//SHA-512
 {
  test("Test Vectors", () {

    InputVectorsSHA2SHA3 inputVectorsSHA2SHA3 =
InputVectorsSHA2SHA3();
    OutputVectorsSHA512 outputVectorsSHA512 =
OutputVectorsSHA512();

    var digest = [];
    for(int i = 0; i <
inputVectorsSHA2SHA3.mapInputVectorsSHA2SHA3.length; i++)
{
      digest = collection.mapHashFunctions[3]["digest"].process(

inputVectorsSHA2SHA3.mapInputVectorsSHA2SHA3[i]["input"]);
      expect(compareArrays(
        digest,
outputVectorsSHA512.mapOutputVectorsSHA512[i]["output"]),
          true, reason: "The actual digest value is "
            +
inputVectorsSHA2SHA3.mapInputVectorsSHA2SHA3[i]["name"]
            + digest.toString() + "                Vector " +
```

```
outputVectorsSHA512.mapOutputVectorsSHA512[i]["output"]
                .toString());
    }
   });
 }); //SHA-512 vector test

 group(collection.mapHashFunctions[4]["name"] + " testing: ", ()
//SHA-3224
 {
  test("Test Vectors", () {

    InputVectorsSHA2SHA3 inputVectorsSHA2SHA3 =
InputVectorsSHA2SHA3();
    OutputVectorsSHA3224 outputVectorsSHA3224 =
OutputVectorsSHA3224();

    var digest = [];
    for(int i = 0; i <
inputVectorsSHA2SHA3.mapInputVectorsSHA2SHA3.length; i++)
{
      digest = collection.mapHashFunctions[4]["digest"].process(

inputVectorsSHA2SHA3.mapInputVectorsSHA2SHA3[i]["input"]);
      expect(compareArrays(
          digest,
outputVectorsSHA3224.mapOutputVectorsSHA3224[i]["output"]),
          true, reason: "The actual digest value is "
              +
inputVectorsSHA2SHA3.mapInputVectorsSHA2SHA3[i]["name"]
            + digest.toString() + "                Vector " +

outputVectorsSHA3224.mapOutputVectorsSHA3224[i]["output"]
                .toString());
    }
   });
 }); //SHA-3224 vector test

 group(collection.mapHashFunctions[5]["name"] + " testing: ", ()
//SHA-3256
 {
  test("Test Vectors", () {

    InputVectorsSHA2SHA3 inputVectorsSHA2SHA3 =
InputVectorsSHA2SHA3();
    OutputVectorsSHA3256 outputVectorsSHA3256 =
OutputVectorsSHA3256();

    var digest = [];
    for(int i = 0; i <
inputVectorsSHA2SHA3.mapInputVectorsSHA2SHA3.length; i++)
{
      digest = collection.mapHashFunctions[5]["digest"].process(

inputVectorsSHA2SHA3.mapInputVectorsSHA2SHA3[i]["input"]);
      expect(compareArrays(
          digest,
outputVectorsSHA3256.mapOutputVectorsSHA3256[i]["output"]),
          true, reason: "The actual digest value is "
              +
inputVectorsSHA2SHA3.mapInputVectorsSHA2SHA3[i]["name"]
            + digest.toString() + "                Vector " +

outputVectorsSHA3256.mapOutputVectorsSHA3256[i]["output"]
                .toString());
    }
   });
 }); //SHA-3256 vector test

 group(collection.mapHashFunctions[6]["name"] + " testing: ", ()
//SHA-3384
 {
  test("Test Vectors", () {

    InputVectorsSHA2SHA3 inputVectorsSHA2SHA3 =
InputVectorsSHA2SHA3();
    OutputVectorsSHA3384 outputVectorsSHA3384 =
OutputVectorsSHA3384();

    var digest = [];
    for(int i = 0; i <
inputVectorsSHA2SHA3.mapInputVectorsSHA2SHA3.length; i++)
{
      digest = collection.mapHashFunctions[6]["digest"].process(

inputVectorsSHA2SHA3.mapInputVectorsSHA2SHA3[i]["input"]);
      expect(compareArrays(
          digest,
outputVectorsSHA3384.mapOutputVectorsSHA3384[i]["output"]),
          true, reason: "The actual digest value is "
              +
inputVectorsSHA2SHA3.mapInputVectorsSHA2SHA3[i]["name"]
            + digest.toString() + "                Vector " +

outputVectorsSHA3384.mapOutputVectorsSHA3384[i]["output"]
                .toString());
    }
   });
 }); //SHA-3384 vector test

 group(collection.mapHashFunctions[7]["name"] + " testing: ", ()
//SHA-3512
 {
  test("Test Vectors", () {

    InputVectorsSHA2SHA3 inputVectorsSHA2SHA3 =
InputVectorsSHA2SHA3();
    OutputVectorsSHA3512 outputVectorsSHA3512 =
OutputVectorsSHA3512();

    var digest = [];
    for(int i = 0; i <
inputVectorsSHA2SHA3.mapInputVectorsSHA2SHA3.length; i++)
{
      digest = collection.mapHashFunctions[7]["digest"].process(

inputVectorsSHA2SHA3.mapInputVectorsSHA2SHA3[i]["input"]);
      expect(compareArrays(
          digest,
outputVectorsSHA3512.mapOutputVectorsSHA3512[i]["output"]),
          true, reason: "The actual digest value is "
              +
inputVectorsSHA2SHA3.mapInputVectorsSHA2SHA3[i]["name"]
            + digest.toString() + "                Vector " +

outputVectorsSHA3512.mapOutputVectorsSHA3512[i]["output"]
                .toString());
    }
   });
 }); //SHA-3512 vector test

 group(collection.mapHashFunctions[8]["name"] + " testing: ", ()
//RIPEMD128
 {
  test("Test Vectors", () {

    InputVectorsRIPEMD128160256320
inputVectorsRIPEMD128160256320 =
InputVectorsRIPEMD128160256320();
    OutputVectorsRIPEMD128 outputVectorsRIPEMD128 =
OutputVectorsRIPEMD128();

    var digest = [];
    for(int i = 0; i <
inputVectorsRIPEMD128160256320.mapInputVectorsRIPEMD128
160256320.length; i++) {
      digest = collection.mapHashFunctions[8]["digest"].process(

inputVectorsRIPEMD128160256320.mapInputVectorsRIPEMD128
160256320[i]["input"]);
      expect(compareArrays(
          digest,
outputVectorsRIPEMD128.mapOutputVectorsRIPEMD128[i]["outp
ut"]),
          true, reason: "The actual digest value is "
              +
inputVectorsRIPEMD128160256320.mapInputVectorsRIPEMD128
160256320[i]["name"]
```

```
            + digest.toString() + "                    Vector " +
outputVectorsRIPEMD128.mapOutputVectorsRIPEMD128[i]["outp
ut"]
                .toString());
    }
   });
 }); //RIPEMD128 vector test

 group(collection.mapHashFunctions[9]["name"] + " testing: ", ()
//RIPEMD160
 {
  test("Test Vectors", () {

    InputVectorsRIPEMD128160256320
inputVectorsRIPEMD128160256320                                =
InputVectorsRIPEMD1281602563200();
    OutputVectorsRIPEMD160      outputVectorsRIPEMD160      =
OutputVectorsRIPEMD160();

    var digest = [];
    for(int              i              =              0;              i              <
inputVectorsRIPEMD128160256320.mapInputVectorsRIPEMD128
160256320.length; i++) {
      digest = collection.mapHashFunctions[9]["digest"].process(
inputVectorsRIPEMD128160256320.mapInputVectorsRIPEMD128
160256320[i]["input"]);
      expect(compareArrays(
        digest,
outputVectorsRIPEMD160.mapOutputVectorsRIPEMD160[i]["outp
ut"]),
          true, reason: "The actual digest value is "
            +
inputVectorsRIPEMD128160256320.mapInputVectorsRIPEMD128
160256320[i]["name"]
            + digest.toString() + "                    Vector " +
outputVectorsRIPEMD160.mapOutputVectorsRIPEMD160[i]["outp
ut"]
                .toString());
    }
   });
 }); //RIPEMD160 vector test

 group(collection.mapHashFunctions[10]["name"] + " testing: ", ()
//RIPEMD256
 {
  test("Test Vectors", () {

    InputVectorsRIPEMD128160256320
inputVectorsRIPEMD128160256320                                =
InputVectorsRIPEMD1281602563200();
    OutputVectorsRIPEMD256      outputVectorsRIPEMD256      =
OutputVectorsRIPEMD256();

    var digest = [];
    for(int              i              =              0;              i              <
inputVectorsRIPEMD128160256320.mapInputVectorsRIPEMD128
160256320.length; i++) {
      digest = collection.mapHashFunctions[10]["digest"].process(
inputVectorsRIPEMD128160256320.mapInputVectorsRIPEMD128
160256320[i]["input"]);
      expect(compareArrays(
        digest,
outputVectorsRIPEMD256.mapOutputVectorsRIPEMD256[i]["outp
ut"]),
          true, reason: "The actual digest value is "
            +
inputVectorsRIPEMD128160256320.mapInputVectorsRIPEMD128
160256320[i]["name"]
            + digest.toString() + "                    Vector " +
outputVectorsRIPEMD256.mapOutputVectorsRIPEMD256[i]["outp
ut"]
                .toString());
    }
   });
 }); //RIPEMD256 vector test

 group(collection.mapHashFunctions[11]["name"] + " testing: ", ()
//RIPEMD320
 {
  test("Test Vectors", () {

    InputVectorsRIPEMD128160256320
inputVectorsRIPEMD128160256320                                =
InputVectorsRIPEMD1281602563200();
    OutputVectorsRIPEMD320      outputVectorsRIPEMD320      =
OutputVectorsRIPEMD320();

    var digest = [];
    for(int              i              =              0;              i              <
inputVectorsRIPEMD128160256320.mapInputVectorsRIPEMD128
160256320.length; i++) {
      digest = collection.mapHashFunctions[11]["digest"].process(
inputVectorsRIPEMD128160256320.mapInputVectorsRIPEMD128
160256320[i]["input"]);
      expect(compareArrays(
        digest,
outputVectorsRIPEMD320.mapOutputVectorsRIPEMD320[i]["outp
ut"]),
          true, reason: "The actual digest value is "
            +
inputVectorsRIPEMD128160256320.mapInputVectorsRIPEMD128
160256320[i]["name"]
            + digest.toString() + "                    Vector " +
outputVectorsRIPEMD320.mapOutputVectorsRIPEMD320[i]["outp
ut"]
                .toString());
    }
   });
 }); //RIPEMD320 vector test

 group(collection.mapHashFunctions[1]["name"] + " testing: ", ()
//SHA-256
 {
  test("Extremaly Long Message Test", () {

    List <int> vector1GB = [];
    List <int> vectorForGB = [0x61, 0x62, 0x63, 0x64, 0x65, 0x66,
0x67, 0x68,
      0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69, 0x63, 0x64,
0x65, 0x66,
      0x67, 0x68, 0x69, 0x6a, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69,
0x6a, 0x6b,
      0x65, 0x66, 0x67, 0x68, 0x69, 0x6a, 0x6b, 0x6c, 0x66, 0x67,
0x68, 0x69,
      0x6a, 0x6b, 0x6c, 0x6d, 0x67, 0x68, 0x69, 0x6a, 0x6b, 0x6c,
0x6d, 0x6e,
      0x68, 0x69, 0x6a, 0x6b, 0x6c, 0x6d, 0x6e, 0x6f, 0x69, 0x6a,
0x6b, 0x6c,
      0x6d, 0x6e, 0x6f, 0x70, 0x6a, 0x6b, 0x6c, 0x6d, 0x6e, 0x6f,
0x70, 0x71,
      0x6b, 0x6c, 0x6d, 0x6e, 0x6f, 0x70, 0x71, 0x72, 0x6c, 0x6d,
0x6e, 0x6f,
      0x70, 0x71, 0x72, 0x73, 0x6d, 0x6e, 0x6f, 0x70, 0x71, 0x72,
0x73, 0x74,
      0x6e, 0x6f, 0x70, 0x71, 0x72, 0x73, 0x74, 0x75];
//abcdefghbcdefghicdefghijdefghijkefghijklfghijklmghijklmnhijklmn
o
    for(int i = 0; i < 16777216; i++){
     vector1GB += vectorForGB;
    }

    var digest = [];
    digest                                                        =
collection.mapHashFunctions[1]["digest"].process(Uint8List.fromLis
t(vector1GB));

    int S = 0;
    for(int i = 0; i < digest.length; i++) {
     int mask = 0x80; //10000000
     for(int j = 0; j < 8; j++) {
      if(digest[i]&mask != 0) {
```

```
          S++;
        }
        mask = mask >> 1;
      }
    }

    double fractionOfUnits = S/digest.length * 8;

    expect(fractionOfUnits >= 0.3 && fractionOfUnits <= 0.7,
        true, reason: "The hash degenerates to zero or one " +
          digest.toString() + "                    Fraction of units " +
            fractionOfUnits.toString());
  });
 }); //SHA-256 Extremaly Long Message Test



}


Uint8List getOutputFromHashFunction(dynamic hashFunction, int n,
int digestSize) {
   //input data preparation
   if(n % digestSize != 0) {
     throw Exception("Cannot create $n bits length output for
$digestSize digest size, "
         "i.e. $n must be divisible by $digestSize");
   }
   List<int> intData = [];

   for(int i = 0; i < 1024; i++) {
    intData.add(i);
   }
   Uint8List data = Uint8List.fromList(intData);

   List<int> hashData = [];
   for(int i = 0, j = 0; i < n; i += digestSize, j++) {
    hashData.addAll(hashFunction.process(data.sublist(j)));
   }

   return Uint8List.fromList(hashData);
}


bool compareArrays(List array1, List array2) {

 if (array1.length == array2.length) {
  return array1.every( (value) => array2.contains(value) );
 } else {
  return false;
 }
}


double computePvalue_MonobitTest(Uint8List hashData, int n) {
 //checking data distribution
 int S = 0;
 for(int i = 0; i < hashData.length; i++) {
  int mask = 0x80; //10000000
  for(int j = 0; j < 8; j++) {
   if(hashData[i]&mask == 0) {
    S--;
   } else {
    S++;
   }
   mask = mask >> 1;
  }
 }
 S = S > 0 ? S : -S; // abs
 double sObc = S/sqrt(n);
 return erfc(sObc/sqrt(2));
}


double computePvalue_FrequencyTestWithinABlock (Uint8List
hashData, int blockSizeBit) {
 //checking data distribution
 if(blockSizeBit % 8 != 0) {
   throw Exception("blockSize should be divisible by 8!");
 }

 int blockSizeByte = blockSizeBit ~/ 8;

 int S = 0;
 List <double> piBlock = [];

 int numberOfBlocksToRemove = hashData.length % blockSizeByte;

 for(int i = 0; i < hashData.length - numberOfBlocksToRemove;) {
  S = 0;
  for(int j = 0; j < blockSizeByte; j++, i++){
   int mask = 0x80;
   while(mask != 0) {
    if(hashData[i]&mask != 0) {
     S++;
    }
    mask = mask >> 1;
   }
  }

  piBlock.add(S/blockSizeBit);
 }

 double X2Obs = 4.0 * blockSizeBit;
 double summX2Obs = 0;
 for(int i = 0; i < piBlock.length; i++){
  summX2Obs += pow(piBlock[i] - 0.5, 2);
 }
 X2Obs *= summX2Obs;

 return gammaIncLower(piBlock.length/2, X2Obs/2);
}


double computePvalue_RunsTest(Uint8List hashData, int n) {
 //checking data distribution
 //Monobit test

 int S = 0, Sk = 0;
 for(int i = 0; i < hashData.length; i++) {
  int mask = 0x80; //10000000
  for(int j = 0; j < 8; j++) {
   if(hashData[i]&mask == 0) {
    S--;
   } else {
    S++;
    Sk++;
   }
   mask = mask >> 1;
  }
 }
 S = S > 0 ? S : -S; // abs
 double sObc = S/sqrt(n);

 double pi = Sk/n;

 double PvalueForTheMonobitTest = erfc(sObc/sqrt(2));

 if(PvalueForTheMonobitTest < 0.01) {
  throw Exception("The Monobit test condition is not met! P-value =
0.0000");
 }


 int VnObs = 0;
 for(int i = 0; i < hashData.length - 1; i++) {
  int mask = 0x80; //10000000
  for(int j = 0; j < 8; j++) {
   if(hashData[i]&mask == 0 && hashData[i+1]&mask == 0) {
    VnObs += 0;
   } else if (hashData[i]&mask != 0 && hashData[i+1]&mask != 0)
{
    VnObs += 0;
   } else {
    VnObs += 1;
   }
   mask = mask >> 1;
  }
```

```
  }
  VnObs += 1;

  double numeratorPvalue = VnObs - 2 * n * pi * (1 - pi);
  numeratorPvalue = numeratorPvalue > 0 ? numeratorPvalue : -
numeratorPvalue; // abs

  return erfc(numeratorPvalue/(2*(sqrt(2*n)) * pi * (1 - pi)));
}


double
computePvalue_TestForTheLongestRunOfOnesInABlock(Uint8List
hashData, int n) {
  //checking data distribution
  int nHashDataSizeBit = hashData.length * 8;
  int mBlockSizeBit = 0;

  if(nHashDataSizeBit < 128 ){
    throw Exception("hashData bit length < 128!");
  } else if (nHashDataSizeBit >= 128 && nHashDataSizeBit < 6272){
    mBlockSizeBit = 8;
  } else if (nHashDataSizeBit >= 6272 && nHashDataSizeBit <
750000){
    mBlockSizeBit = 128;
  } else if (nHashDataSizeBit >= 750000){
    mBlockSizeBit = 10000;
  }

  int blockSizeByte = mBlockSizeBit ~/ 8;

  int S = 0, Smax = 0;
  List <int> maxRun = [];

  int numberOfBlocksToRemove = hashData.length % blockSizeByte;

  for(int i = 0; i < hashData.length - numberOfBlocksToRemove;) {
    S = 0;
    Smax = 0;
    for(int j = 0; j < blockSizeByte; j++, i++){
      int mask = 0x80;
      while(mask != 0) {
        if(hashData[i]&mask != 0) {
          S++;
        } else if (hashData[i]&mask == 0) {
          S = 0;
        }
        if (S > Smax) {
          Smax = S;
        }
        mask = mask >> 1;
      }
    }

    //debugPrint('Smax: $Smax');
    maxRun.add(Smax);
  }

  var vi = List.filled(7, 0);
  for(int i = 0; i < maxRun.length; i++) {
    switch (mBlockSizeBit) {
      case 8:
        {
          if (maxRun[i] <= 1) {
            vi[0] += 1;
          } else if (maxRun[i] == 2) {
            vi[1] += 1;
          } else if (maxRun[i] == 3) {
            vi[2] += 1;
          } else if (maxRun[i] >= 4) {
            vi[3] += 1;
          }
          break;
        }
      case 128:
        {
          if (maxRun[i] <= 4) {
            vi[0] += 1;
          } else if (maxRun[i] == 5) {
            vi[1] += 1;
          } else if (maxRun[i] == 6) {
            vi[2] += 1;
          } else if (maxRun[i] == 7) {
            vi[3] += 1;
          } else if (maxRun[i] == 8) {
            vi[4] += 1;
          } else if (maxRun[i] >= 9) {
            vi[5] += 1;
          }
          break;
        }
      case 10000:
        {
          if (maxRun[i] <= 10) {
            vi[0] += 1;
          } else if (maxRun[i] == 11) {
            vi[1] += 1;
          } else if (maxRun[i] == 12) {
            vi[2] += 1;
          } else if (maxRun[i] == 13) {
            vi[3] += 1;
          } else if (maxRun[i] == 14) {
            vi[4] += 1;
          } else if (maxRun[i] == 15) {
            vi[5] += 1;
          } else if (maxRun[i] >= 16) {
            vi[6] += 1;
          }
          break;
        }

      default:
        throw Exception("The error in determining the length for each
block M. "
            + "M did not take any of the values: 8, 128, 10 000 !!!");
    }
  }

  //debugPrint('Pi: $vi');

  int kTable = 0, nTable = 0;
  if(mBlockSizeBit == 8){
    kTable = 3;
    nTable = 16;
  }
  if(mBlockSizeBit == 128){
    kTable = 5;
    nTable = 49;
  }
  if(mBlockSizeBit == 10000){
    kTable = 6;
    nTable = 75;
  }

  var pi = [];
  if (mBlockSizeBit == 8 && kTable == 3){
    pi = [0.2148, 0.3672, 0.2305, 0.1875];
  }
  if (mBlockSizeBit == 128 && kTable == 5){
    pi = [0.1174, 0.2430, 0.2493, 0.1752, 0.1027, 0.1124];
  }
  if (mBlockSizeBit == 10000 && kTable == 6){
    pi = [0.0882, 0.2092, 0.2483, 0.1933, 0.1208, 0.0675, 0.0727];
  }

  double x2Obs = 0;
  for (int i = 0; i <= kTable; i++){
    x2Obs += (pow(vi[i] - nTable * pi[i], 2))/(nTable * pi[i]);
  }

  return gammaIncLower(kTable/2, x2Obs/2);
}


double    computeProbability_TestForTheAvalancheEffect(Uint8List
input,  dynamic hashFunction) {
  //checking data distribution
```

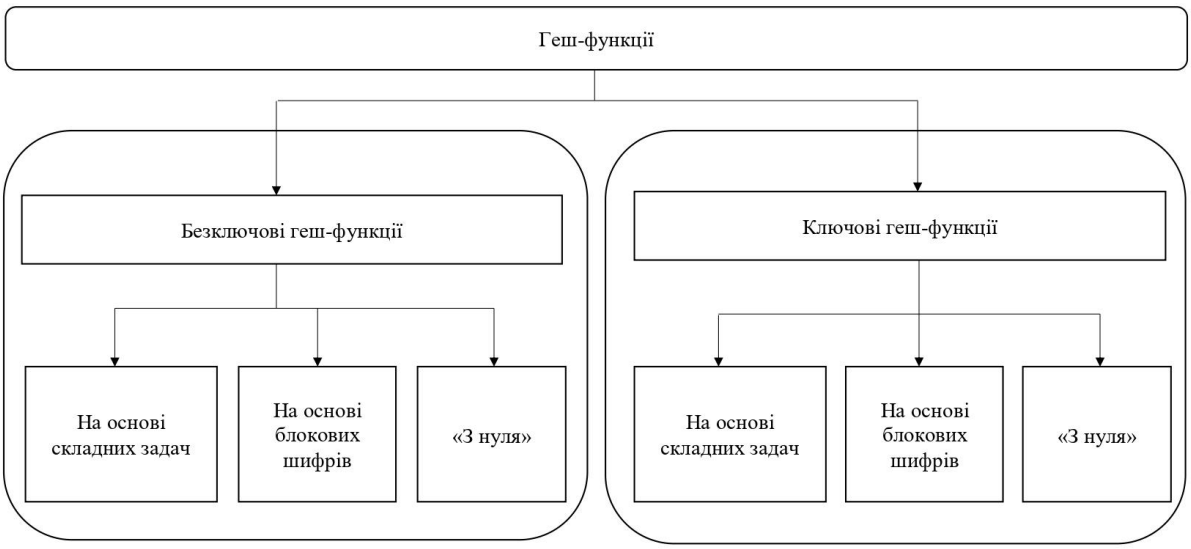
```
  List <int> copyInput = [];
  List <double> theRatioOfTheChangedBits = [];

  for(int i = 0; i < input.length; i++) {
    int mask = 0x80; //10000000
    for(int j = 0; j < 8; j++) {
      copyInput = [];
      for(int l = 0; l < input.length; l++){
        copyInput.add(input[l]);
      }

      copyInput[i] = copyInput[i]^mask;

      Uint8List hashData = hashFunction.process(input);
      Uint8List                hashDataCopyInput                =
hashFunction.process(Uint8List.fromList(copyInput));

      for(int k = 0; k < hashData.length; k++) {

        hashDataCopyInput [k] = hashData[k]^hashDataCopyInput[k];

        // 0101
        // 0010
        // 0111
      }

      int S = 0;
      for(int m = 0; m < hashDataCopyInput.length; m++) {
        int maskTwo = 0x80; //10000000
        for(int n = 0; n < 8; n++) {
          if(hashDataCopyInput[m]&maskTwo != 0) {
            S++;
          }
          maskTwo = maskTwo >> 1;
        }
      }

      theRatioOfTheChangedBits.add(S/(hashDataCopyInput.length   *
8 ));
      mask = mask >> 1;
    }
  }
  //debugPrint('theRatioOfTheChangedBits:
$theRatioOfTheChangedBits');

  double x2Obs = 0;
  for(int i = 0; i < theRatioOfTheChangedBits.length; i++){
    x2Obs += pow((theRatioOfTheChangedBits[i] - 0.5), 2)/0.5;

    /*double cv = pow((theRatioOfTheChangedBits[i] - 0.5), 2)/0.5;
    debugPrint('cv: $cv');*/
  }
  //debugPrint('x2Obs: $x2Obs');

  return          gammaIncLower(theRatioOfTheChangedBits.length/2,
x2Obs/2);
}
```

ІЛЮСТРАТИВНА ЧАСТИНА

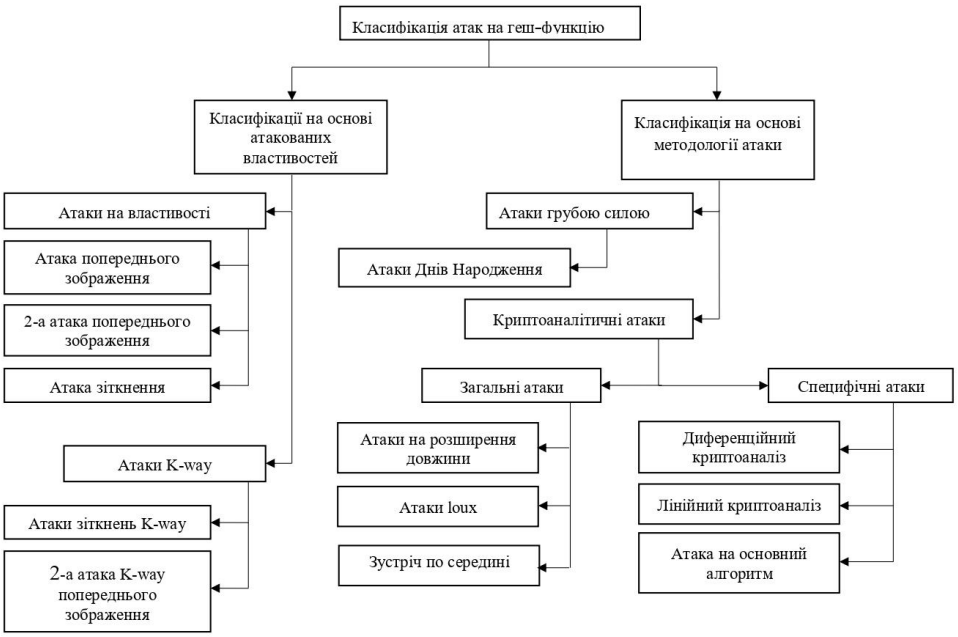
МЕТОД ТА ЗАСІБ КРИПТОАНАЛІЗУ ГЕШ-ФУНКЦІЙ

(Назва магістерської кваліфікаційної роботи)

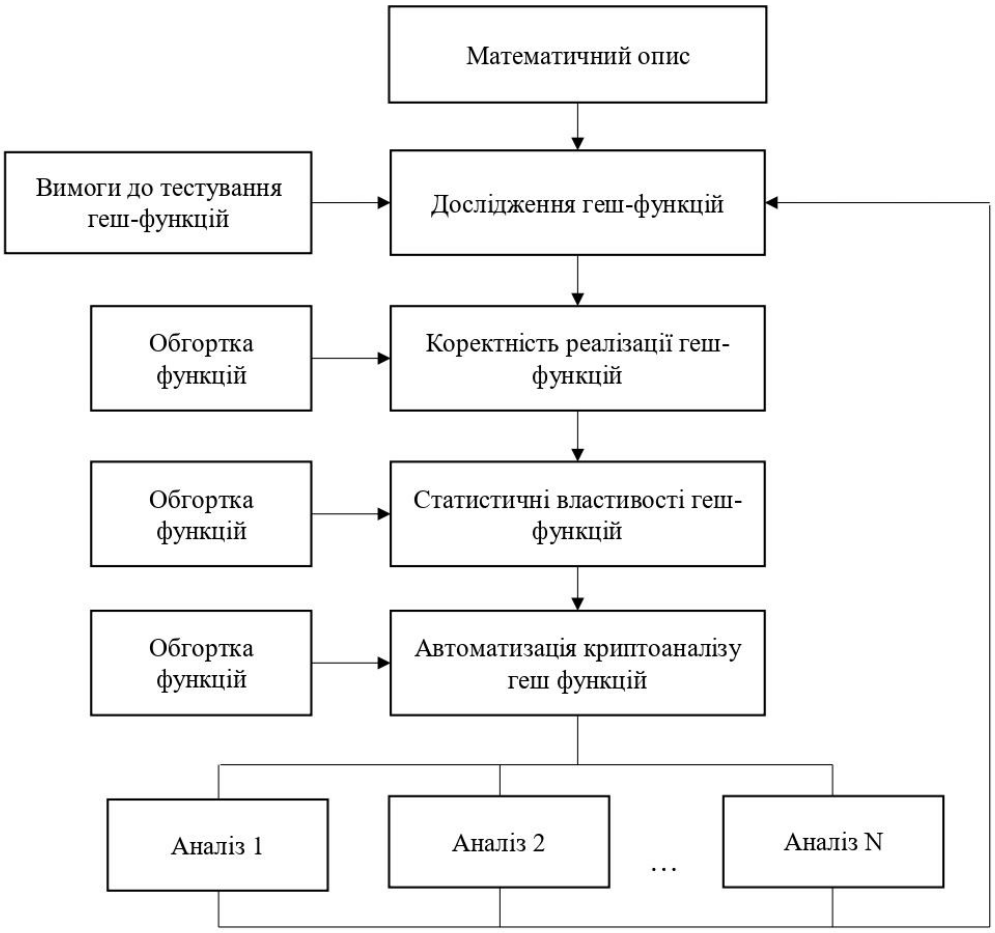
Класифікація геш-функцій



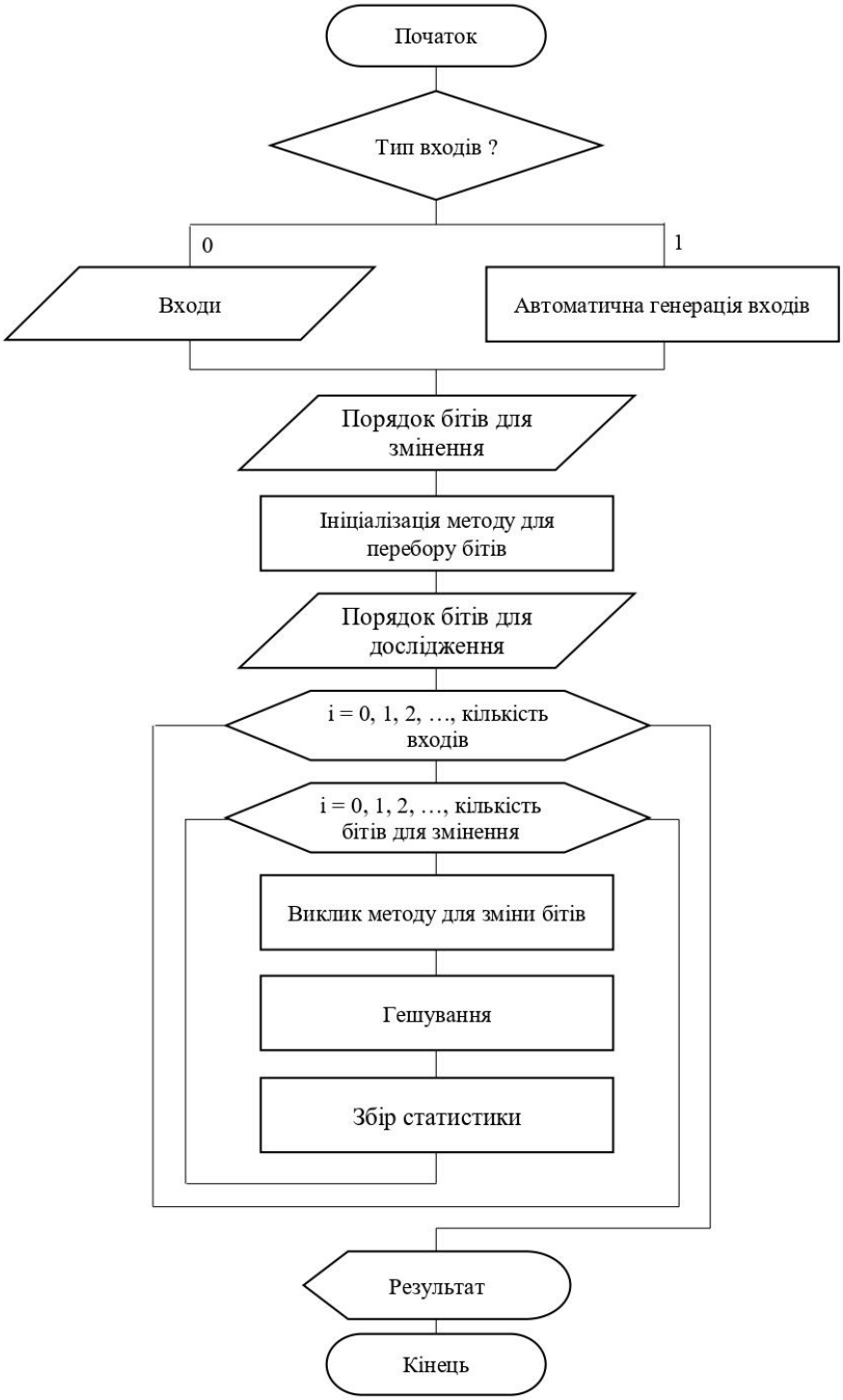
Результати порівняльного аналізу атак на геш-функції



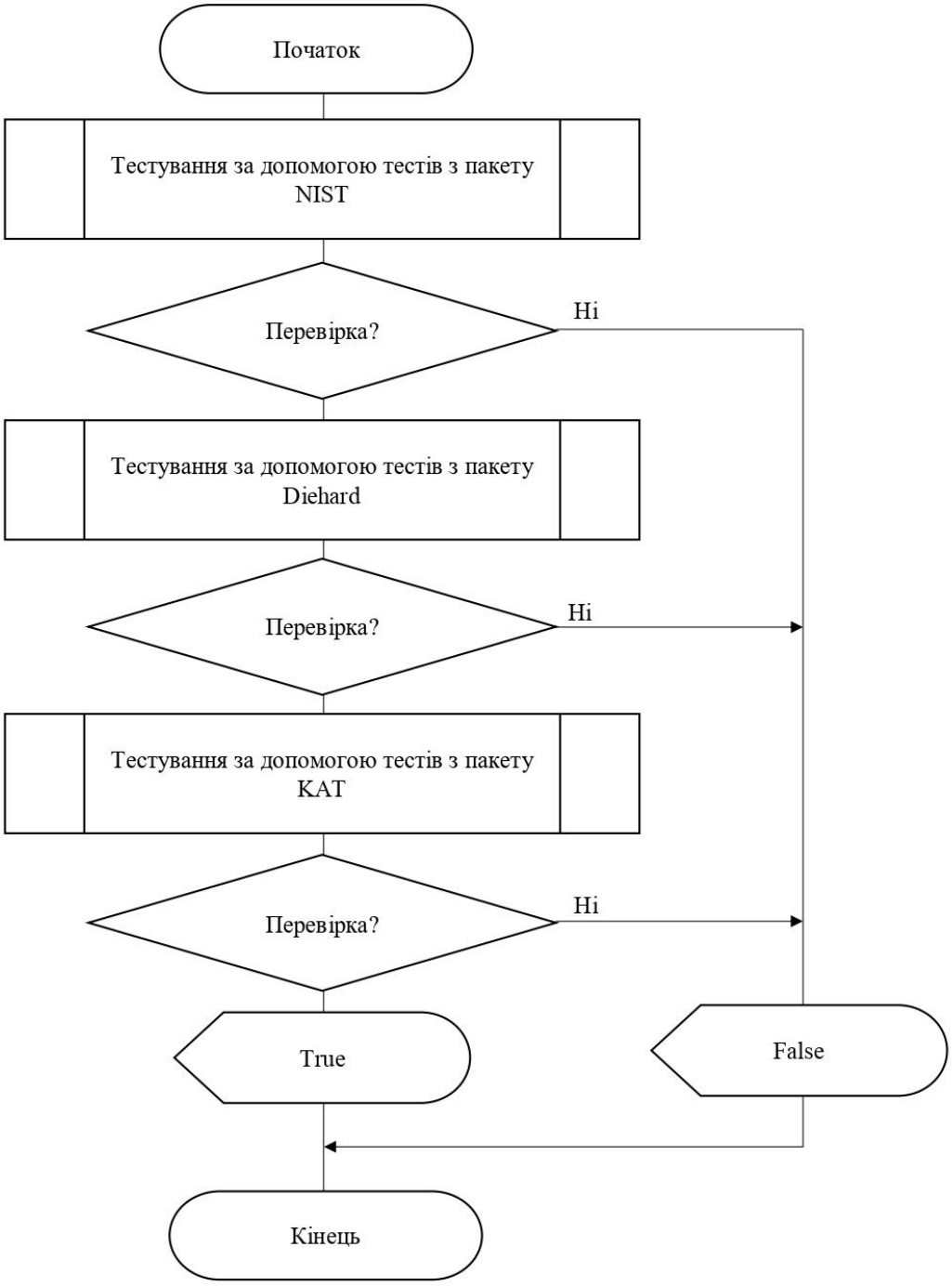
Метод криптоаналізу геш-функцій



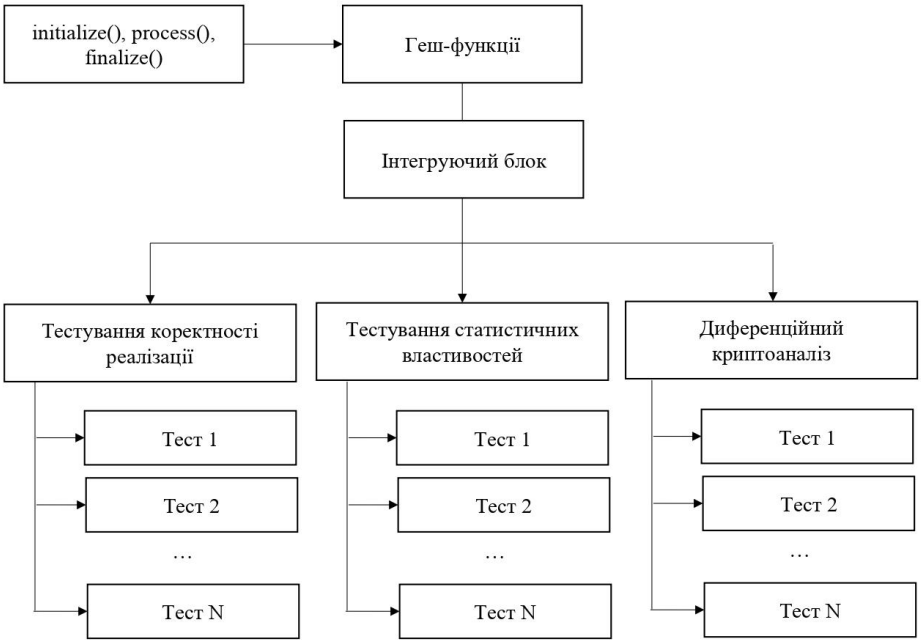
Алгоритм диференційного криптоаналізу



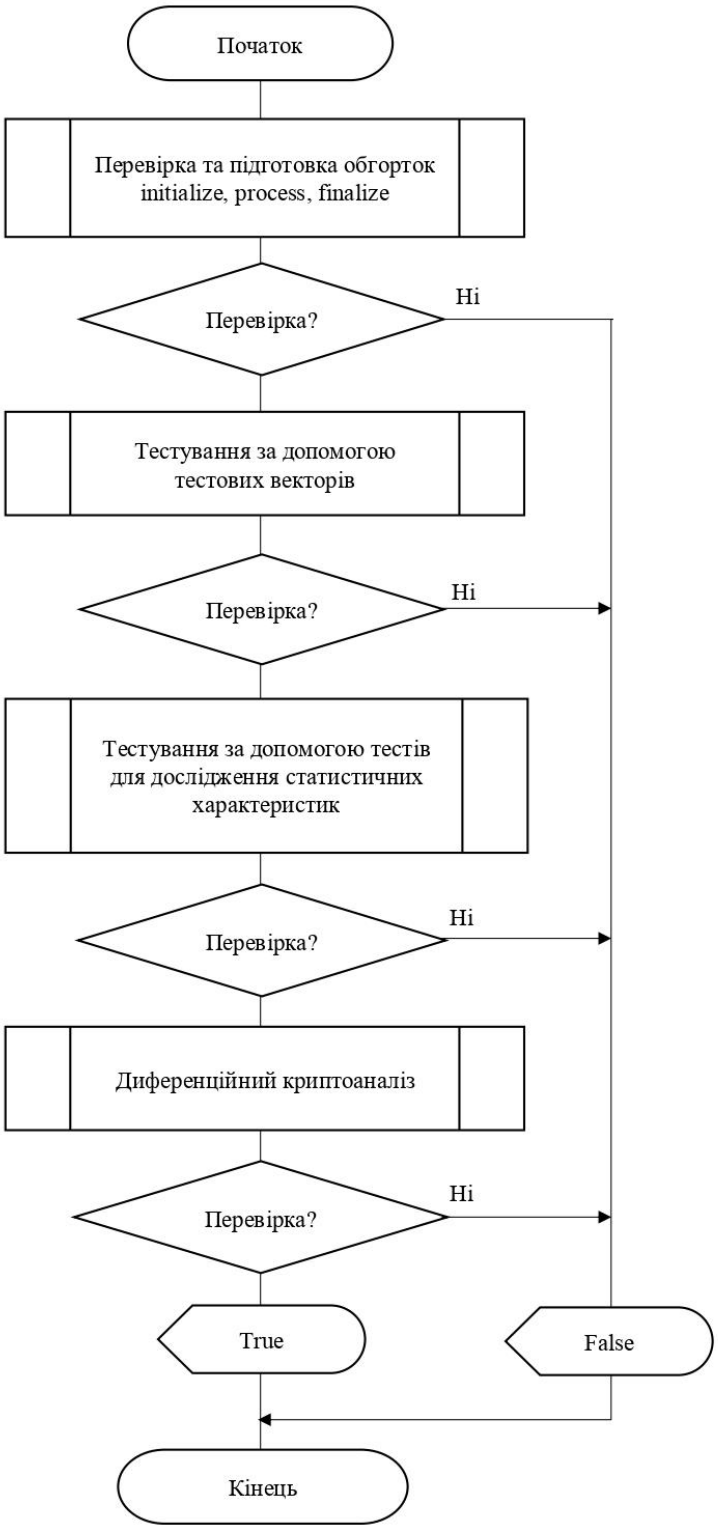
Алгоритм дослідження статистичних властивостей геш-функцій



Структура засобу криптоаналізу геш-функцій



Алгоритм роботи засобу криптоаналізу геш-функцій



Результати експериментального дослідження засобу

Вигляд програми під час виконання тесту

```
C:\Users\vital\AndroidStudioProjects\crypto_dart_tests>flutter test test
00:03 +6: SHA-256 testing:  Runs Test
```

Вигляд програми після успішного проходження тестів

```
C:\Users\vital\AndroidStudioProjects\crypto_dart_tests>flutter test test
00:07 +52: All tests passed!
```

Вигляд програми після успішного проходження тестів

```
C:\Users\vital\AndroidStudioProjects\crypto_dart_tests>flutter test test
00:29 +48 -1: scrypt testing:  Monobit test [E]
  Expected: <true>
    Actual: <false>
  The actual P-value value is 0.0017500065824132586

  package:test_api                                  expect
  package:flutter_test/src/widget_tester.dart 429:3  expect
  test\hashTests\hashTests.dart 21:9                 run.<fn>.<fn>

00:29 +51 -1: Some tests failed.
```

Вигляд результату тесту тестових векторів алгоритму SHA-256 для SHA-512

```
00:11 +1 -1: SHA-512 testing:  Test Vectors [E]
  Expected: <true>
    Actual: <false>
  The actual digest value is 'empty'[207, 131, 225, 53, 126, 239, 184, 189, 241, 84, 40, 80, 214, 109, 128, 7, 214, 32, 228, 5, 11, 87, 21, 220,
08, 209, 60, 93, 133, 242, 176, 255, 131, 24, 210, 135, 126, 236, 47, 99, 185, 49, 189, 71, 65, 122, 129, 165, 56, 50, 122, 249, 39, 218, 62]
 152, 252, 28, 20, 154, 251, 244, 200, 153, 111, 185, 36, 39, 174, 65, 228, 100, 155, 147, 76, 164, 149, 153, 27, 120, 82, 184, 85]

  package:test_api                                  expect
  package:flutter_test/src/widget_tester.dart 429:3  expect
  test\hashTests\hashTests.dart 141:9                run.<fn>.<fn>
```

Результати експериментального дослідження засобу

Вигляд результату монобітного тесту для нульової геш-функції

```
C:\Users\vital\AndroidStudioProjects\crypto_dart_tests>flutter test test
00:08 +0 -1: ZeroHashFunction  testing:  Monobit test [E]
  Expected: <true>
    Actual: <false>
  The actual P-value value is 1.0301065608617454e-64

  package:test_api                                  expect
  package:flutter_test/src/widget_tester.dart 429:3  expect
  test\hashTests\hashTests.dart 120:7               run.<fn>.<fn>
```

Вигляд результату тестування алгоритмів MD2 та MD5

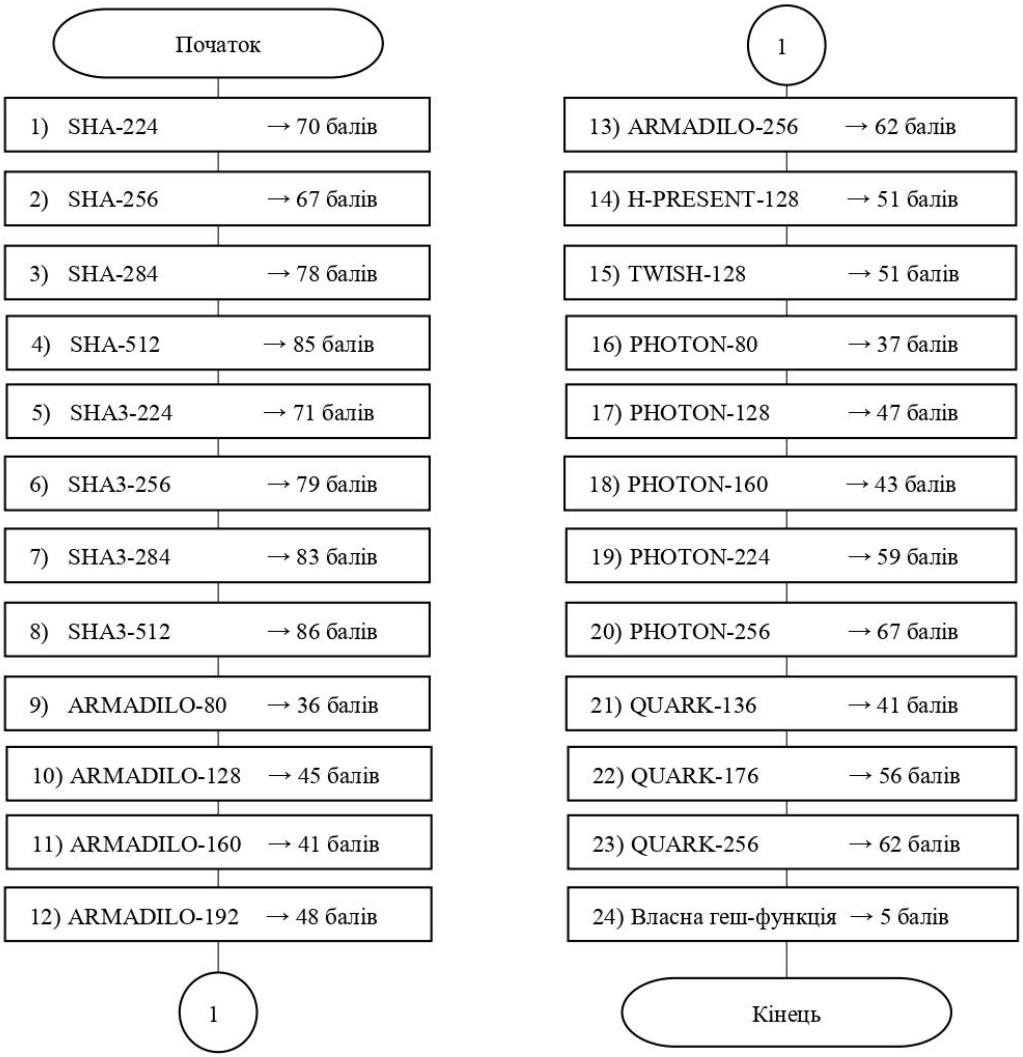
```
05:44 +65 -13: MD2 testing:  Avalanche effect test [E]
  Expected: <true>
    Actual: <false>
  The actual P-value value is 0.0

  package:test_api                                  expect
  package:flutter_test/src/widget_tester.dart 429:3  expect
  test\hashTests\hashTests.dart 99:11               run.<fn>.<fn>

05:47 +70 -14: MD5 testing:  Avalanche effect test [E]
  Expected: <true>
    Actual: <false>
  The actual P-value value is 0.0
```

Вигляд результату тестування алгоритмів Keccak-512

```
C:\Users\vital\AndroidStudioProjects\crypto_dart_tests>flutter test test
00:07 +52: All tests passed!
```

Результати дослідження стійкості та коректності реалізації геш-функцій

Підсумок результатів тестування геш-функцій



Результати дослідження стійкості та коректності реалізації геш-функцій

Вигляд графіку порівняння коректності реалізації та стійкості алгоритмів гешування