Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Факультет комп'ютерних наук
Кафедра теоретичної та прикладної системотехніки

«Затверджую»
Зав. кафедри теоретичної та
прикладної системотехніки
_____ д.т.н., проф. С. І. Шматков
«___» грудня 2023 р.

Пояснювальна записка

до кваліфікаційної роботи
магістра

на тему: **«Система забезпечення цілісності інформації, що передається в мережах передачі даних»**

Захищено на засіданні
Атестаційної комісії № 42
протокол № __ від __.12.2023 р.
Оцінка _____ / _____
Голова Атестаційної комісії
_____ **СКОБ Ю. О.**

Виконав:
студент 2 курсу, групи КУ– 61
за спеціальністю 151 –Автоматизація та комп'ютерно-інтегровані технології.
Галузь знань 15 – Автоматизація та приладобудування
Маницький Сергій Сергійович

Керівник:
д.т.н. , професор
Мірошник Марина Анатоліївна

Рецензент:
к.т.н. доцент, професор кафедри АПОТ ХНУРЕ
Шкіль Олександр Сергійович

Харків 2023

АНОТАЦІЯ

Пояснювальна записка до магістерської атестаційної роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел і двох додатків. Загальний обсяг роботи складає 73 сторінки, із яких 51 сторінка основної частини з 29 рисунками, 21 найменуванням списку використаних джерел та 4 додатками.

***Метою* дослідження** вдосконалення системи забезпечення цілісності інформації, що передається в мережах передачі за допомогою комбінованого алгоритму перевірки інформації на правильність.

Об'єкт дослідження: Процес забезпечення цілісності інформації що передається в мережах передачі даних

Предмет дослідження: методи та моделі реалізації комбінованого алгоритму для забезпечення цілісності передачі інформації.

На відміну від існуючих популярних методів забезпечення цілісності інформації комбінований метод пропонує одночасно не тільки перевірку цілісності даних а і виправлення певної кількості помилок не витрачаючи час на повторне надсилання інформації.

Область застосування — навчання, мережі передачі даних. Розроблений програмний продукт може широко використовуватися в навчальному процесі студентів.

Ключові слова: мережі передачі даних, цілісність інформації.

ABSTRACT

The explanatory note to the master's attestation work consists of an introduction, three sections, conclusions, a list of used sources and two appendices. The total volume of work is 73 pages, of which 51 pages of the main part with 29 figures, 21 names of the list of used sources and 4 appendices.

The purpose of the research is to improve the system for ensuring the integrity of information transmitted in transmission networks using a combined algorithm for checking information for correctness.

Object of research: The process of ensuring the integrity of information that transmitted in data networks

Research subject: methods and models of implementation of a combined algorithm to ensure the integrity of information transmission.

In contrast to the existing popular methods of ensuring the integrity of information, the combined method simultaneously offers not only a check of data integrity, but also the correction of a certain number of errors without wasting time on resending information.

Field of application — training, data transmission networks. The developed software product can be widely used in the educational process of students.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

ARP- Address Resolution Protocol

CRC - Cyclic redundancy code

EtherCAT- Ethernet for Control Automation Technology

FTP- File Transfer Protocol

HTTP- Hypertext Transfer Protocol

HTTPS- HTTP Secure

ICMP- Internet Control Message Protocol

IP- Internet Protocol

IMAP- Internet Message Access Protocol

MD - Message digest

POP- Post Office Protocol

SHA - Secure Hash Algorithm

SMTP- Simple Mail Transfer Protocol

SSH- Secure Shell

TCP- Transmission Control Protocol

TS - Typescript

UDP- User Datagram Protocol

ВСТУП

Цілісність цифрової інформації є ключовою складовою практично будь-якої системи, пов'язаної з передачею даних. Вона відіграє важливу роль в уникненні втрати або спотворення даних під час процесу передачі, що є фундаментальною проблемою в області обміну інформацією. Цілісність цифрової інформації визначається набором методів, які призначені для виявлення та, у випадку потреби, усунення помилок, які можуть виникнути під час передачі даних в мережах передачі даних. Ця система спрямована на забезпечення надійності та точності передачі інформації в умовах, де можливість виникнення різних помилок може бути високою. Засоби для виявлення помилок можуть використовуватися для ідентифікації будь-яких випадкових чи навмисних змін у переданих даних, забезпечуючи важливий елемент безпеки та надійності в області обміну інформацією.

Унікальність роботи: Розробка засобів для забезпечення цілісності інформації в мережах передачі даних представляє собою значущу задачу у сучасних умовах розвитку інформаційних систем. Цій темі приділено значну увагу у ряді досліджень, і багато з них фокусуються на загальних аспектах, представлених у навчальній літературі. На відміну від існуючих популярних методів забезпечення цілісності інформації комбінований метод пропонує одночасно не тільки перевірку цілісності даних а і виправлення певної кількості помилок не витрачаючи час на повторне надсилання інформації.

Постановка задачі: Провести аналіз існуючих методів забезпечення цілісності інформації, виявити та порівняти між собою їх переваги та недоліки. Розглянути комбінований алгоритм забезпечення цілісності інформації. Розробити програмну модель візуалізації роботи методів забезпечення цілісності інформації, що передається в мережах передачі даних.

Актуальність теми дослідження: Тема забезпечення цілісності інформації в мережах передачі даних є актуальною з кількох причин. По перше,

зростання обсягів передачі даних. Сучасні мережі передачі даних обробляють величезний обсяг інформації. Збільшення кількості передач даних зумовлює більше можливостей для помилок чи вторгнень, що може призвести до порушення цілісності. По друге, споживачі та бізнес очікують високої якості обслуговування. Користувачі та підприємства очікують надійності та цілісності даних в мережах. Порушення цілісності може призвести до втрати довіри, втрати клієнтів і фінансових втрат. Отже, забезпечення цілісності інформації в мережах передачі даних залишається актуальною, враховуючи зростання складності та обсягів інформаційного обміну.

Метою дослідження вдосконалення системи забезпечення цілісності інформації, що передається в мережах передачі за допомогою комбінованого алгоритму перевірки інформації на правильність.

Об'єкт дослідження: Процес забезпечення цілісності інформації що передається в мережах передачі даних

Предмет дослідження: методи та моделі реалізації комбінованого алгоритму для забезпечення цілісності передачі інформації.

Методи дослідження: У даному дослідженні використовуються підходи, що ґрунтуються на методах аналізу та порівняння функціонування алгоритмів. При розробці програмної моделі, яка має виступати у вигляді веб-сайту з можливістю віддаленого доступу для студентів, було прийнято рішення використовувати мову програмування TypeScript. Цей вибір зумовлений кількома перевагами, що роблять TypeScript ефективним інструментом для створення веб-застосунків. Вона є однією з найпоширеніших мов програмування для швидкого рішення задач, подібних до поставленої. TypeScript забезпечує статичну типізацію, яка може виявити потенційні помилки на ранніх стадіях процесу розробки та підвищити надійність коду. Зазначимо також, що TypeScript володіє багатьма іншими зручностями, такими як читабельний синтаксис, велика спільнота розробників та висока продуктивність. Ці якості роблять її відмінним вибором для веб-розробки

РОЗДІЛ 1

АНАЛІЗ СУЧАСНИХ МЕТОДІВ ТА ЗАСОБІВ ЗАБЕЗПЕЧЕННЯ ЦІЛІСНОСТІ ІНФОРМАЦІЇ У МЕРЕЖАХ ПЕРЕДАЧІ ДАНИХ

Методологічний апарат збереження цілісності інформації в комп’ютерній мережі налічує в собі чимало методів для використання. Все різноманіття цього інструментарію можна поділити на дві великих групи: контрольна сума та кодування. Зокрема, можна виділити дві ключові групи методів у цьому апараті: методи, що використовують контрольні суми, і методи, що оперують кодуванням. Контрольні суми використовуються для створення короткого числового значення (суми), яке є результатом арифметичних операцій над бітами або байтами переданих даних. Це значення потім включається в передачу даних і може використовуватися отримувачем для виявлення невідповідностей в переданих даних. Особливістю контрольних сум є можливість визначення втрати інформації наприкінці процесу передачі даних, шляхом порівняння контрольних сум первісно відправленого повідомлення та отриманого повідомлення. Втрата чи спотворення певної інформації виявляється тим чином, що контрольні суми повідомлень не співпадають. З іншого боку, методи кодування використовують спеціальні алгоритми для представлення та виправлення помилок в переданих даних. Вони дозволяють виявляти та усувати помилки, що можуть виникнути під час передачі, забезпечуючи високий рівень цілісності і надійності інформації в мережі.

1.1 Загальні відомості про основні протоколи передачі інформації

1.1.1 Протокол TCP/IP

Протокол TCP/IP - це набір стандартів, що визначають спосіб передачі даних в комп'ютерних мережах, зокрема в Інтернеті. Розроблений в кінці 1970-х років, цей набір протоколів є основою для функціонування Інтернету та

багатьох сучасних комп'ютерних мереж[1]. Розглянемо основні компоненти протоколу TCP/IP:

IP : Цей протокол відповідає за маршрутизацію та передачу даних в мережі. Кожному пристрою в мережі присвоюється унікальна IP-адреса, яка використовується для ідентифікації пристрою в мережі.

TCP: TCP забезпечує надійну та упорядковану передачу даних між пристроями в мережі. Він керує встановленням, підтримкою та розривом з'єднання між відправником та одержувачем.

UDP: На відміну від TCP, UDP забезпечує швидший, але менш надійний спосіб передачі даних. Використовується в додатках, де невелике затримання є більш важливим, ніж гарантована доставка.

ICMP: ICMP використовується для відправки повідомлень про помилки та діагностичної інформації, такої як запити на exo (ping).

ARP: ARP використовується для перетворення IP-адрес в фізичні адреси мережевих пристроїв.

Процес передачі даних за допомогою протоколу TCP/IP включає наступні кроки:

- Упаковка даних у пакети.

- Присвоєння кожному пакету заголовка, який містить інформацію про порядок передачі та інші параметри.

- Маршрутизація пакетів через мережу за допомогою IP-адрес.

- Передача пакетів через встановлене TCP-з'єднання.

- Розпакування даних на стороні отримувача.

Протокол TCP/IP є відкритим стандартом, що означає, що його специфікації доступні громадськості, і його можна використовувати в різних типах мереж. Це робить його основою для Інтернету та багатьох сучасних мережевих технологій.

1.1.2 Протокол HTTP

Протокол HTTP є основним протоколом для передачі інформації в інтернеті. Він використовується для передачі гіпертекстових документів, таких як веб-сторінки, між веб-серверами та клієнтськими браузерами. Основною метою HTTP є забезпечення зручного способу обміну текстовою інформацією, такою як HTML-сторінки[1].

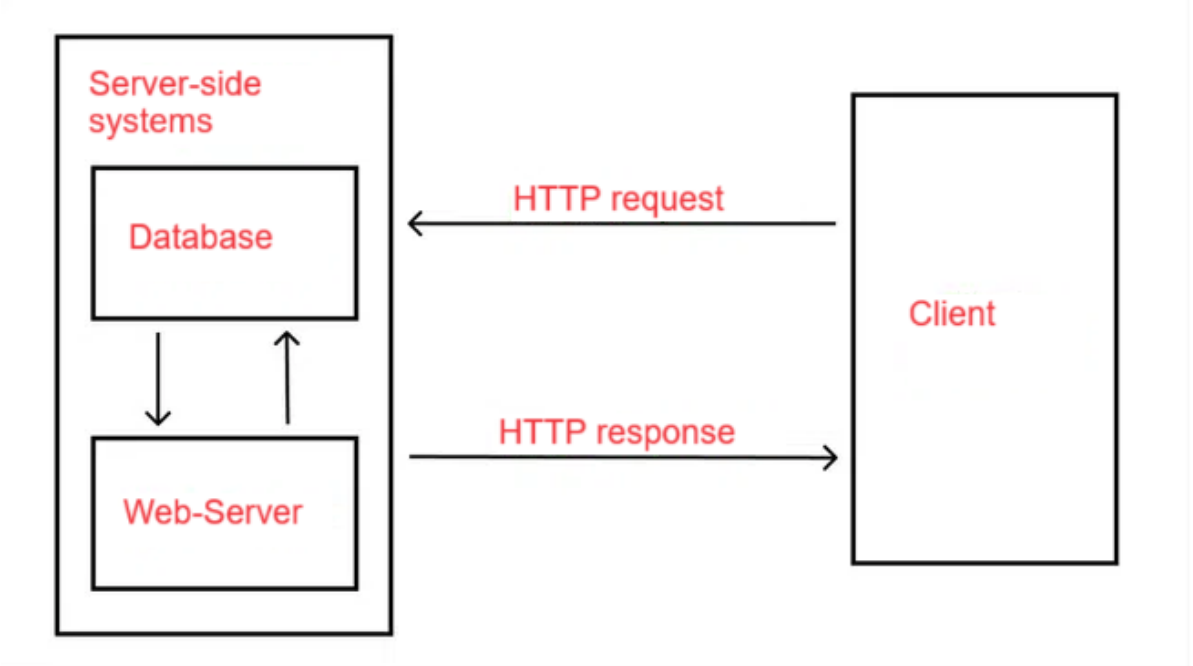


Рисунок 1.1 - Протокол HTTP

Основні характеристики HTTP:

Простота інтерфейсу: HTTP має простий та інтуїтивно зрозумілий інтерфейс, що дозволяє клієнтам і серверам легко обмінюватися даними.

Безстандартність: HTTP – безстандартний протокол, що означає, що кожний запит і відповідь обробляються незалежно, без збереження стану між ними. Це допомагає забезпечити швидкі та прості взаємодії.

Протокол засобів безпеки: Однак HTTP сам по собі є небезпечним протоколом, оскільки дані передаються у відкритому вигляді. Для забезпечення безпеки передачі даних використовуються додаткові шари, такі як HTTPS (HTTP Secure), який використовує шифрування за допомогою SSL або TLS для захисту інформації.

Модель клієнт-сервер: HTTP працює на основі моделі клієнт-сервер, де клієнтський браузер ініціює запит до сервера, а сервер повертає відповідь.

Методи запиту: HTTP визначає різні методи запиту, такі як GET (отримати ресурс), POST (надіслати дані для обробки на сервері), PUT (оновити існуючий ресурс або створити новий), DELETE (видалити ресурс) та інші.

Статус-коди відповіді: Кожна відповідь сервера супроводжується статус-кодом, який вказує на результат виконання запиту (наприклад, 200 OK, 404 Not Found, 500 Internal Server Error).

Заголовки: HTTP-запити і відповіді можуть містити заголовки, які надають додаткову інформацію про передачу даних.

HTTP є основою для великої частини взаємодії в Інтернеті. З браузерами, серверами та іншими інтернет-сервісами використовують його для обміну інформацією. Більш безпечна версія протоколу, HTTPS, застосовує шифрування та інші заходи безпеки[1].

1.1.3 Протокол FTP

Протокол передачі файлів (FTP) представляє собою стандартний механізм для пересилання файлів в мережі Інтернет. Цей протокол застосовується для зручного обміну файлами між комп'ютерами, незалежно від їхнього розташування в одній локальній мережі або в Інтернеті. Основна концепція FTP полягає в наданні користувачам зручного та ефективного засобу обміну файлами. Це забезпечує високий рівень доступності для вивантаження та завантаження файлів, роблячи процес обміну інформацією швидким та надійним.

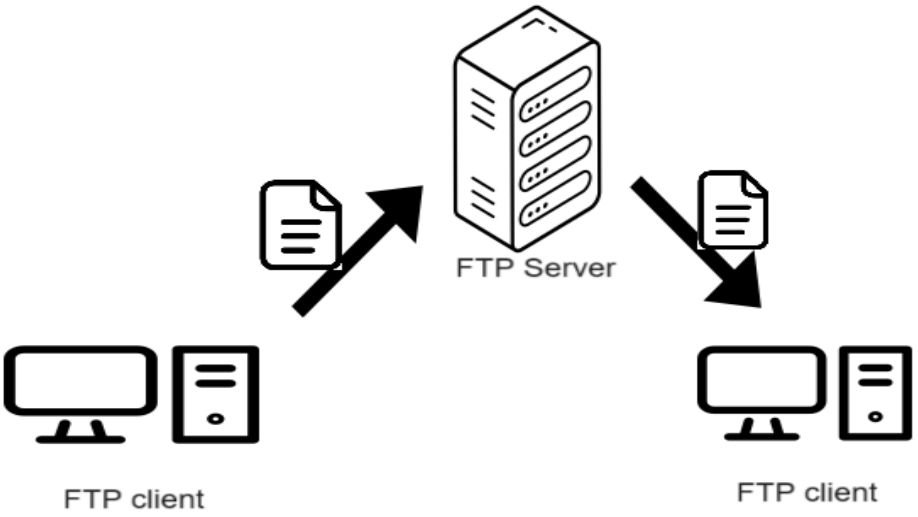


Рисунок 1.2 - Протокол FTP

Основні характеристики FTP:

Модель клієнт-сервер: FTP використовує модель клієнт-сервер, де клієнтські програми здійснюють підключення до серверів для передачі файлів.

Механізми авторизації: Перед початком передачі файлів клієнт повинен авторизуватися на сервері за допомогою ідентифікаційних даних, таких як ім'я користувача та пароль[1].

Керування командами та даними: FTP використовує два канали для обміну даними. Керуючий канал використовується для відправки команд (наприклад, для зміни каталогу), а даний канал - для передачі самого файлу.

Режими передачі даних: FTP підтримує два режими передачі даних: активний та пасивний. У режимі активного з'єднання сервер встановлює нове з'єднання для передачі даних, тоді як у пасивному режимі клієнт встановлює нове з'єднання.

Операції з файлами: FTP дозволяє виконувати різноманітні операції з файлами, такі як завантаження, вивантаження, перейменування, видалення та інші.

Захист даних: Оригінальний FTP передає дані у відкритому вигляді, що може бути небезпечним. Однак існують захищені версії протоколу, такі як FTPS

(FTP Secure) та SFTP (SSH File Transfer Protocol), які використовують шифрування для захисту конфіденційності даних.

FTP залишається важливим інструментом для обміну файлами, хоча в деяких випадках його використання може бути заміщено більш безпечними протоколами передачі файлів, такими як SCP або HTTPS.

1.1.4 Протокол SMTP

Протокол передачі електронної пошти є стандартним протоколом для відправки електронних листів в мережі Інтернет. SMTP використовується для надсилання листів від клієнтів до серверів електронної пошти та між серверами для подальшої доставки[1].

Основні характеристики SMTP:

Модель клієнт-сервер: SMTP використовує клієнт-серверну модель, де клієнтські програми надсилають електронні листи серверам для подальшої обробки та передачі.

Надсилання повідомлень: Основне завдання SMTP - передача повідомлень. Клієнт надсилає повідомлення на сервер, який потім спробує доставити його до адресата.

Порти: Для надсилання електронних листів використовуються два порти: порт 25 для незашифрованого з'єднання та порт 587 для захищеного (часто використовується для аутентифікації).

Попередження про відмову: У разі неможливості відправити лист SMTP повертає відповідь про відмову (зазвичай за допомогою кодів помилок), яку може отримати відправник.

Аутентифікація: Деякі сервери вимагають аутентифікації, особливо якщо відправник намагається відправити повідомлення через сервер, який не належить йому.

Взаємодія з POP/IMAP: SMTP працює разом з протоколами отримання електронної пошти, такими як POP3 і IMAP (Internet Message Access Protocol), які використовуються для отримання листів з поштових скриньок. SMTP

вважається найважливішим протоколом для надсилання електронних листів, і багато програм та сервісів використовують його для обміну листами. Однак важливо відзначити, що він відповідає тільки за відправку листів і не включає в себе функції отримання або зберігання листів.

1.1.5 Криптографічний мережевий протокол SSH

SSH (Secure Shell) - це криптографічний мережевий протокол, який використовується для безпечного управління мережевими пристроями та обміну даними між двома пристроями через ненадійну мережу. Основна мета SSH - забезпечити захищену передачу даних через небезпечну мережу, таку як Інтернет.

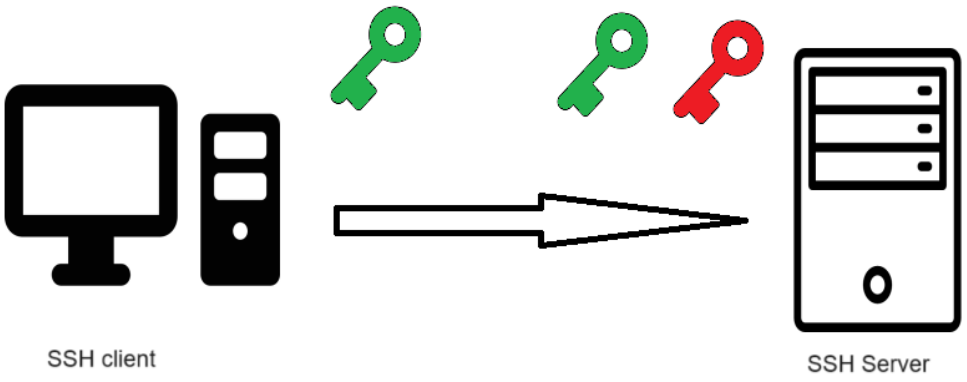


Рисунок 1.3 - Протокол SSH

Основні характеристики SSH:

Шифрування: SSH використовує криптографічні методи для шифрування всієї переданої інформації, включаючи логіни, паролі та команди.

Аутентифікація: SSH використовує аутентифікацію за допомогою публічного та приватного ключів, що робить процес більш безпечним, ніж традиційний ввід паролю.

Безпечний туннель: SSH може встановлювати безпечний тунель для захищеної передачі інших мережевих протоколів, таких як FTP або HTTP.

Проксі і перенаправлення портів: Шелл забезпечує можливість встановлення проксі-серверів та перенаправлення портів для безпечного доступу до різних служб.

Підтримка версій: Існує декілька версій протоколу SSH, включаючи SSH-1 та більш сучасний SSH-2. Остання версія є більш безпечною і зазвичай використовується в сучасних системах.

Командний інтерфейс: Шелл також може використовуватися як командний інтерфейс для виконання команд та програм на віддаленому пристрої.

SSH широко використовується адміністраторами систем, розробниками та іншими користувачами для забезпечення безпеки та конфіденційності в мережах. Цей протокол є невід'ємною частиною безпеки в багатьох організаціях і гарантує, що конфіденційна інформація залишається захищеною під час передачі через мережі.

1.1.6 Протокол Modbus

Modbus - це протокол передачі даних, який був розроблений для забезпечення взаємодії між електронними пристроями в індустрії. Цей протокол використовується для обміну інформацією між промисловими контролерами та обладнанням, таким як сенсори, вимикачі та інші пристрої автоматизації.

Основні характеристики Modbus:

Простота: Modbus є досить простим у реалізації протоколом, що дозволяє йому працювати на різних платформах і з різним обладнанням.

Відкритий стандарт: Modbus є відкритим протоколом, що робить його доступним для використання різними виробниками.

Підтримка різних інтерфейсів передачі даних: Modbus може використовуватися через різні фізичні середовища передачі даних, такі як RS-232, RS-485, TCP/IP.

Мастер-слейв архітектура: В основі Modbus лежить концепція мастер-слейв, де один пристрій (мастер) ініціює звернення до інших пристроїв

(слейвів) для отримання або передачі даних.

Підтримка змішаних мереж: Modbus може працювати в змішаних мережах, де використовуються різні фізичні середовища передачі даних.

Різновиди Modbus: Існують різні варіанти Modbus, такі як Modbus RTU (з використанням байт-орієнтованого протоколу передачі), Modbus ASCII (використовує людино-читабельний формат) і Modbus TCP (протокол передачі даних через TCP/IP).

Modbus залишається одним із найпопулярніших промислових протоколів і знаходить широке застосування в системах автоматизації та контролю.

1.1.7 Протокол EtherCAT

EtherCAT- це високопродуктивний мережевий протокол, спроектований для застосувань в області автоматизації та управління. Він використовує технологію етапного передавання даних, що дозволяє досягти високої швидкості обміну даними між пристроями в реальному часі.

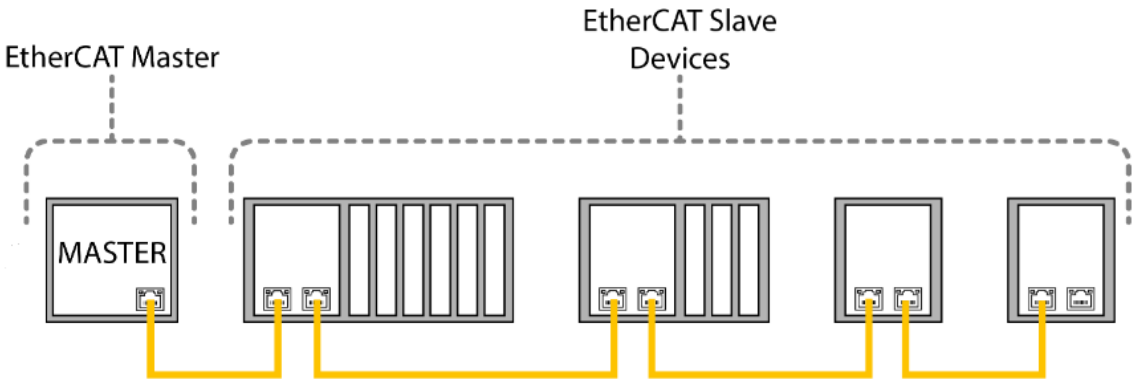


Рисунок 1.4 - Протокол EtherCAT

Основні характеристики EtherCAT:

Швидкість передачі даних в реальному часі: EtherCAT спроектований для забезпечення високої швидкості передачі даних в реальному часі, що робить його ідеальним для високопродуктивних промислових застосувань.

Топологія лінійної шини з зірковим підключенням: Мережа EtherCAT може бути реалізована у вигляді лінійної шини з можливістю зіркового підключення пристроїв, що робить її гнучкою та легкою для розширення.

Висока точність синхронізації: EtherCAT забезпечує високу точність синхронізації між пристроями в мережі, що є критичним для багатьох застосувань в області автоматизації.

Підтримка технології введення/виведення (I/O): Протокол має вбудовану підтримку технології введення/виведення, що дозволяє ефективно взаємодіяти з різними пристроями у високопродуктивних системах управління.

Відкритий стандарт: EtherCAT є відкритим стандартом, що дозволяє виробникам використовувати його без обмежень.

Підтримка стандарту Ethernet: Використання технології Ethernet забезпечує сумісність з існуючими мережевими інфраструктурами.

EtherCAT знаходить широке застосування в різних галузях промисловості, таких як машинобудування, автоматизовані виробництва, тестування та вимірювання, де важлива висока точність та швидкість обміну даними в реальному часі.

1.2 Методи контрольних сум

1.2.1 Хеш-функція MD4

MD4, що стоїть за абревіатурою Message Digest 4, є алгоритмом хешування, розробленим професором Рональдом Рівестом в Массачусетському університеті в 1990 році. Цей алгоритм приймає будь-яке вхідне повідомлення і генерує 128-бітне хеш-значення, відоме як дайджест повідомлення[2]. Цей алгоритм є попередником MD5 (рис. 1.1).

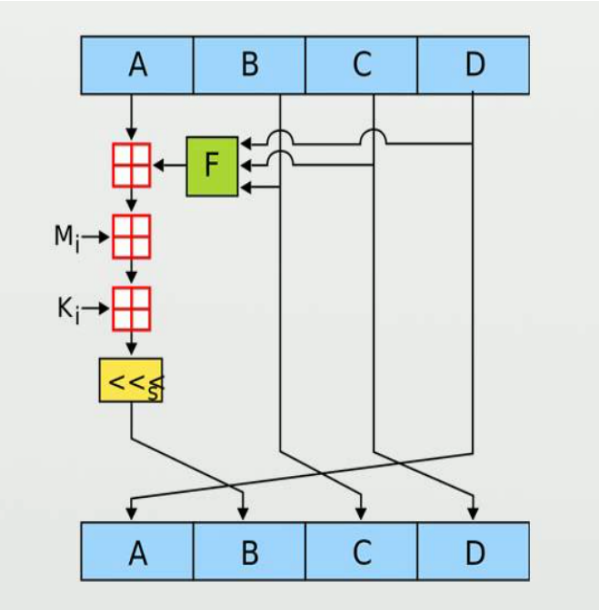


Рисунок 1.5 - Схема одного раунда хешування MD4.

Алгоритм MD4 працює за таким принципом: вважається, що на вхід подається повідомлення, яке складається з n біт, і метою є обчислити його хеш. Значення n може бути будь-яким не від'ємним цілим числом; це число може дорівнювати нулю, не бути кратним восьми, і може мати будь-яку величину[3].

Рівень безпеки, який вбудовано в MD4, спроектований для розробки відносно міцних гібридних систем електронного цифрового підпису. Ці системи використовують MD4 та криптосистему з відкритим ключем[2]. Рональд Рівест вважав, що MD4 може бути використаний для систем, які вимагають високого рівня криптостійкості. Проте, він також зауважував, що MD4 був створений як швидкий алгоритм хешування, тому може виявитися менш ефективним з точки зору криптографічної стійкості.Пізніше дослідження підтвердили його точку зору. Виявилося, що для застосувань, де головним є криптостійкість, віддають перевагу використанню алгоритму MD5.

1.2.2 Безпечний хеш-алгоритм

SHA-1 (Secure Hash Algorithm 1) — це алгоритм криптографічного хешування, описаний у RFC 3174. Він генерує 160-бітове хеш-значення, відоме як дайджест повідомлення, для будь-якого вхідного повідомлення довільної

довжини. SHA-1 — це алгоритм, який використовується в різноманітних криптографічних додатках і протоколах, а також рекомендується для використання урядовими установами у США. Принципи, що стоять в основі SHA-1, схожі на ті, які використовував Рональд Рівест при розробці MD4.

SHA-1 втілює хеш-функцію, що ґрунтується на концепції функції стиснення. Ця функція стиснення приймає блок повідомлення розміром 512 біт та вивід попереднього блоку повідомлення як вхідні параметри. Вихід представляє собою суму всіх хеш-блоків до даного моменту. Хеш-значенням всього повідомлення є результат обчислення для останнього блоку. Алгоритм починається з процесу ініціалізації перед своєю роботою[4]. Оригінальний текст розбивається на блоки, кожен розміром 512 біт. Останній блок доповнюється до довжини, кратної 512 біт. Спочатку додається одиниця, а потім додаються нулі до блоку, щоб його довжина стала рівною$(512 - 64 = 448)$ біт. Останні 64 біти використовуються для запису довжини вихідного повідомлення у бітах. Якщо останній блок має довжину понад 448, але менше 512 біт, то процес доповнення виконується наступним чином: спочатку додається одиниця, потім додаються нулі до кінця 512-бітного блоку. На наступному етапі генерується додатковий блок завдовжки 512 біт, в якому залишаються нулі до досягнення довжини 448 біт, а останні 64 біти використовуються для внесення інформації про довжину вихідного повідомлення у бітах. Навіть у випадку, коли повідомлення вже має необхідну довжину, завжди проводиться доповнення останнього блоку (рис. 1.2).

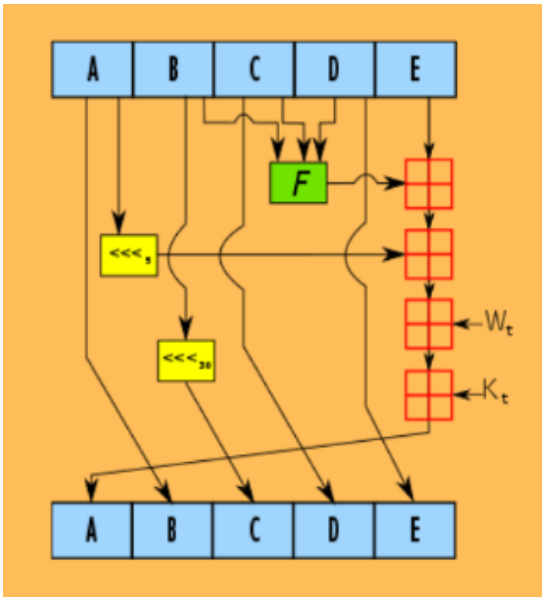


Рисунок 1.6 - Схема одного раунда хешування SHA-1

1.2.3 Хеш-функція RIPEMD-160

RIPEMD-160 (Race Integrity Primitives Evaluation Message Digest) - це хеш-алгоритм, який був створений Хансом Доббертіном, Антоном Босселарсом і Бартом Пренелом в Католицькому університеті Лувена. RIPEMD-160 створює 160-бітне хеш-значення, відоме як дайджест, для будь-якого вхідного повідомлення. Цей хеш-алгоритм є покращеною версією RIPEMD, який використовував принципи MD4 та продемонстрував подібну продуктивність до більш популярного SHA-1.Також існують версії цього алгоритму з розмірами хеш-значень 128, 256 і 320 біт, які мають назви відповідно RIPEMD-128, RIPEMD-256 і RIPEMD-320. 128-бітна версія є просто заміною оригінального RIPEMD, який також мав розмір 128 біт і мав певні вразливості. Версії з розмірами 256 і 320 біт відрізняються подвоєною довжиною хеш-значення, через що ймовірність колізій , але це не підвищує їх криптостійкість (рис. 1.3).

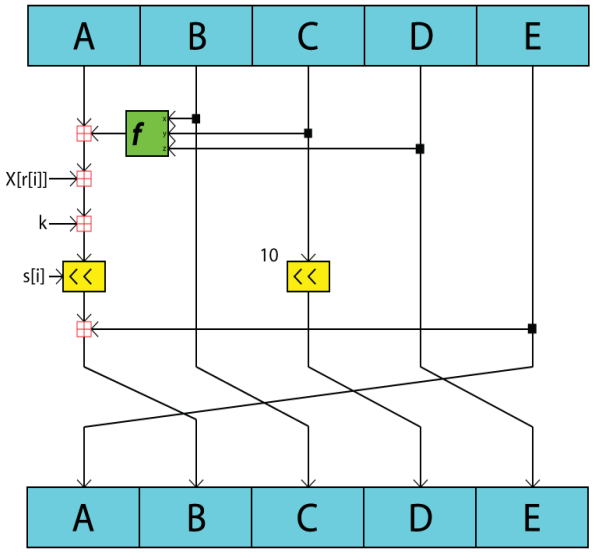


Рисунок 1.7 - Схема одного раунда хешування RIPEMD

RIPEMD був створенийу відкритій академічній громаді але на практиці, використання RIPEMD-160 менше поширене, ніж SHA-1. Важливо відзначити, що RIPEMD-160 може використовуватися без обмежень патентами.

1.2.4 Хеш-функція ДСТУ Р34.11-94

Стандарт ДСТУ Р34.11-94 є українським криптографічним нормативом, який визначає процедуру обчислення хеш-функції. Його розробив Всеукраїнський науково-дослідний інститут стандартизації, а введено його було 23 травня 1994 року.

Розмір отриманого дайджесту в стандарті складає 256 біт, так само як і розмір блоку вхідних даних. Стандарт визначає процедуру та метод обчислення хеш-функції для послідовності символів. Використання цього стандарту являється обов'язковим для алгоритму хешування у деяких комерційних організаціях. Центральний Банк України встановлює вимогу використовувати ДСТУ Р 34.11-94 для електронного підпису документів, які йому надаються.

1.2.5 Хеш-функція MD5

MD5 (Message Digest 5), створений Рональдом Л. Рівестом в 1991 році, є 128-бітним алгоритмом хешування. Його призначенням є створення "відбитків" або "дайджестів" повідомлень будь-якої довжини та подальша перевірка їхньої достовірності. MD5 є покращеною і більш безпечною версією алгоритму MD4[5] (рис. 1.4).

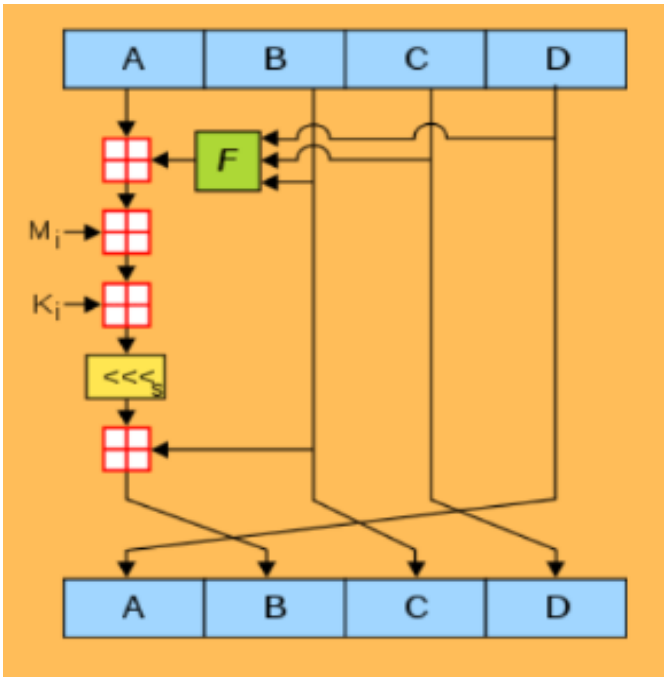


Рисунок 1.8 - Схема одного раунда хешування MD5

Алгоритм MD5 унаслідував багато рис MD4. Однак до нього було додано ще один раунд, що підняло їхню кількість до чотирьох, порівняно з трьома у MD4[2]. Також була внесена модифікація в схему роботи алгоритму та додано нову константу на кожному кроці кожного раунду Це було зроблено з метою мінімізації впливу вхідного повідомлення.[3] Ця константа підсумовується з результатом функції F та блоком даних, що дозволяє досягти більшої стійкості та надійності.

Функція G була змінена на XZ v (Y not (Z)), замість попередньої (XY v XZ v YZ). Це призвело до того, що результат кожного кроку тепер обчислюється на основі результату попереднього кроку, прискорюючи процес зміни результату. Також відбулися зміни в порядку обробки вхідних слів на етапах 2 і 3 раундів. MD5 забезпечує отримання відносно надійного ідентифікатора для блоку даних,

і ця особливість алгоритму широко використовується в різних галузях. Наприклад, в області пошуку дублюючих файлів на комп'ютері, можна порівнювати MD5 файлів, а не їх вміст.

Одним із прикладів є програма DuplicateFinder, яка працює на операційних системах Windows і Linux. Такий самий принцип пошуку може застосовуватися й в Інтернеті. Також MD5 використовується для перевірки цілісності завантажених файлів. Деякі програми, наприклад, інсталяційні диски, можуть іти разом із значенням хеш-кодування, щоб забезпечити можливість перевірки цілісності файлів під час завантаження. Алгоритм MD5 спеціально розроблений для ефективності на 32-бітових системах. За додаткову перевагу можна відзначити той факт, що MD5 не потребує великих таблиць підстановки, що сприяє компактності кодування. Ці характеристики роблять алгоритм MD5 відмінним об'єктом для подальших досліджень в даній роботі.

Алгоритм отримує вхідний потік даних, для якого потрібно знайти хеш. Довжина повідомлення може бути будь-якою, включаючи випадок, коли воно має нульову довжину[5]. Це значення є цілим і не від'ємним. Його кратність будь-якому числу не є обов'язковою. Після отримання даних відбувається процес підготовки потоку для обчислень. Зазначені перетворення включають в себе п'ять основних етапів.

На першому кроці вирівнювання потоку додається бітовий шаблон. Спочатку в кінець потоку дописується одиничний біт, а після цього додається необхідна кількість нульових біт. Це вирівнювання виконується так, щоб новий розмір вхідних даних був кратним 512, при взятті залишку від ділення на 448. На другому етапі в процесі вирівнювання додається 64-бітове представлення довжини повідомлення, яке складається з останніх 64 біт. Спочатку записуємо молодші 4 байти. Якщо довжина перевищує $2^{64} - 1$ то додають лише молодші біти. Після цього довжина потоку стає кратною 512[5]. Обчислення базуються на уявленні цього потоку даних у вигляді масиву слів по 512 біт.

На третьому етапі для проведення обчислень ініціалізуються чотири 32-бітні змінні, і надаються початкові значення у вигляді шістнадцяткових чисел, починаючи з менш значущого байту[5].

$A = 01\ 23\ 45\ 67;\ B = 89\ AB\ CD\ EF;$

$C = FE\ DC\ BA\ 98\ ;\ D = 76\ 54\ 32\ 10.$

Ці змінні призначені для зберігання проміжних результатів обчислень. Початковий стан ABCD визначається ініціалізаційним вектором.

На четвертому етапі виконуються всі інші обчислення, що умовно поділені на чотири раунди, кожен з яких містить 16 перетворень. В кінці кожного раунду результат поточних обчислень підсумовується з результатом попередніх. По завершенні раунду перевіряється наявність ще блоків для обчислень. У випадку наявності блоків, змінюється номер елемента масиву $(n\ ++)$, і процес переходить до початку раунду. На завершальному етапі обчислень, MD5-хеш або дайджест розміщується у буфері ABCD. Якщо розглядати дані на рівні окремих байтів, процес починається з менш значущого байта A і завершується старшим байтом D, утворюючи в результаті MD5-хеш.

1.2.6 SHA-2

SHA-2 (Secure Hash Algorithm 2) (рис. 1.5) – це сімейство криптографічних хеш-функцій, розроблених Національним інститутом стандартів і технологій (NIST) США. SHA-2 представляє собою покращення попереднього стандарту SHA-1 і включає в себе ряд алгоритмів з різною довжиною хеш-значень, таких як SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 і SHA-512/256.

SHA-224 та SHA-256 генерують хеші довжиною 224 та 256 біт відповідно.

SHA-384 та SHA-512 генерують хеші довжиною 384 та 512 біт відповідно.

SHA-512/224 та SHA-512/256 є варіантами SHA-512 і генерують хеші довжиною 224 та 256 біт відповідно.

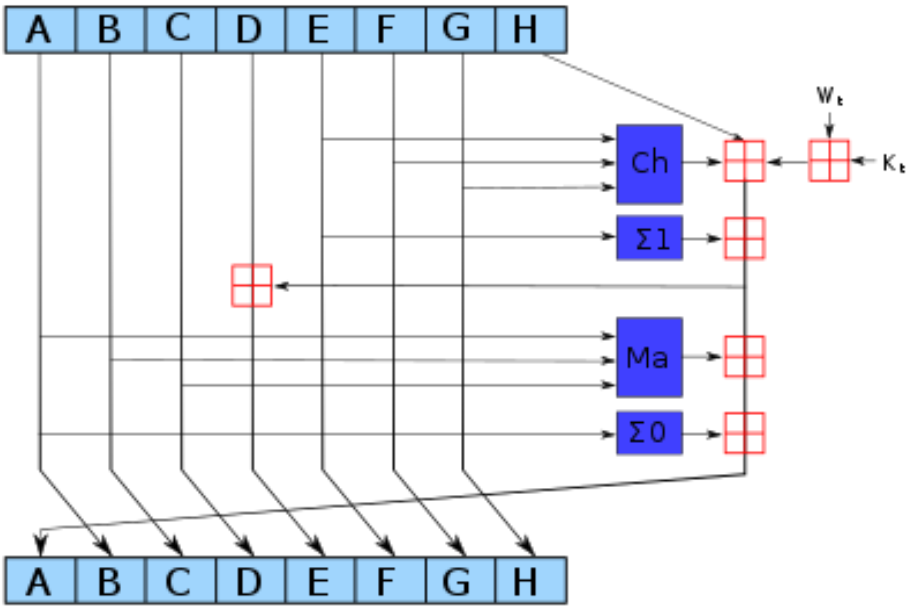


Рисунок 1.9 - Схема одного раунда хешування SHA-2

SHA-2 вважається стійким до колізій, що означає, що ймовірність виявлення двох різних вхідних повідомлень з однаковим хешем є дуже мала[6]. До вересня 2012 року, SHA-2 вважався надійним, але після анонсу практично можливого методу зіткнення для SHA-1, рекомендації перейти на SHA-256 або вище стали ще більш актуальними. SHA-2 широко використовується в криптографічних застосунках, таких як створення цифрових підписів, захист даних та інші області, де важлива інтегрітет і конфіденційність даних.

Ініціалізація: Ініціалізуємо внутрішні стани хеш-функцій (розмір стану залежить від конкретної версії SHA-2). Обробка блоків: Дані розбиваються на блоки фіксованого розміру, які обробляються послідовно. Доповнення даних: Додаємо біти до вхідного повідомлення для вирівнювання довжини. Обчислення хешу: Кожен блок обробляється за допомогою складних раундів, включаючи логічні операції, побітові зсуви, і нелінійні функції. Отримання результату: Хеш отримується як результат обчислень.

Хоча SHA-2 залишається стійким до багатьох атак, важливо слідкувати за оновленнями та рекомендаціями з безпеки, оскільки криптографічні стандарти можуть змінюватися відповідно до еволюції кіберзагроз.

1.2.7 Циклічний надлишковий код

Це метод обчислення чексуми, який використовується для виявлення помилок в переданих даних. CRC базується на поліноміальних арифметичних операціях над бітами даних. Ось загальний принцип обчислення CRC:

1. вибір полінома CRC. Перше, що потрібно зробити, це вибрати поліном CRC, який визначає, які біти даних використовуються для обчислення контрольної суми. Різні стандарти та застосування можуть використовувати різні поліноми CRC;

2. додавання нульових бітів. Для обчислення CRC до вхідних даних (пакету або повідомлення) додаються нульові біти, які визначаються довжиною полінома CRC;

3. ділення поліномів. Далі виконується ділення цього бітового рядка, який складається з вхідних даних та нульових бітів, на поліном CRC. Ділення виконується за правилами бінарної арифметики з використанням операцій XOR (виключне "або").Залишок ділення (CRC): Залишок від ділення стає контрольною сумою, яку додають до вихідних даних для відправки або збереження;

4. перевірка на приймачі. При отриманні даних одержувач обчислює CRC на отриманих даних і порівнює його з CRC, який був включений у передачу. Якщо вони збігаються, це означає, що дані були прийняті без помилок. Якщо CRC не збігається, це вказує на наявність помилок в даних.

Важливо відзначити, що використовується багато різних поліномів CRC для різних застосувань. Вірне обрання полінома CRC дуже важливо, оскільки від нього залежить надійність виявлення помилок в даних. Крім того, довжина CRC (кількість бітів в контрольній сумі) також може варіюватися в залежності від конкретного стандарту або вимог застосування.

1.3 Методи кодування

1.3.1 Код Хемінга

Код Хемінга - це спеціальний тип блочного кодування, який використовується для виявлення та виправлення помилок у цифрових даних. Він був розроблений Річардом Хемінгом у 1950 році і є одним із ранніх і важливих прикладів кодування з корекцією помилок[7].

Базовий принцип роботи полягає в наступному. Визначається кількість контрольних біт, яку необхідно додати до блоку даних. Це визначається за допомогою формули $2^r >\geq m + r + 1$ де m - кількість бітів для даних, r - кількість контрольних біт. Контрольні біти розташовуються на певних позиціях у кодовому слові, які є степенями двійки. Наприклад, біт на позиції
1 - контрольний для 1, біт на позиції 2 - контрольний для 2, біт на позиції
4 - контрольний для 4, і так далі.Контрольні біти додаються на визначені позиції у кодовому слові.

Кожен контрольний біт визначається як парний чи непарний за кількістю бітів, які він контролює.Кодові слова, що містять дані та контрольні біти, передаються чи зберігаються. При передачі, якщо виникає помилка, код Хемінга дозволяє виявити та навіть виправити одну помилку.При отриманні кодованого слова код Хемінга перевіряє контрольні біти. Якщо виявляється одна помилка, код може бути виправлений. Якщо виявляється більше однієї помилки, можна виявити, але не виправити.

1.3.2 Код Соломона-Ріда

Це вид циклічного коду, який використовується для виявлення та виправлення помилок в передачі даних. Цей код отримав свою назву від імені вчених Еррата Соломона та Гордона Ріда, які незалежно один від одного вперше розглянули його в середині 20-го століття.

Код Соломона-Ріда має широке застосування в різних сферах, зокрема, в оптичних засобах зберігання, QR-кодах, бар-кодах, цифрових комунікаціях та інших областях, де важлива надійність передачі даних[8].

Цей код базується на математичних концепціях алгебри, зокрема, на ідеях поля Галуа та алгебраїчних геометричних кривих. Основна ідея коду Соломона-Ріда полягає в тому, щоб вбудовувати в дані додаткові біти (зайві символи), завдяки чому можна виявляти та виправляти помилки[9].

Для зрозуміння процесу корекції помилок за допомогою коду Соломона-Ріда, розглянемо приклад RS(255, 223), що означає, що загальна довжина блоку (N) становить 255 символів, а кількість символів даних (K) дорівнює 223. Таким чином, код має можливість виправлення 16 помилок.

Основні кроки використання коду Соломона-Ріда для виправлення помилок:

Кодування:

- Створення блоку даних з символів, які ви хочете передати.
- Додавання до цього блоку даних контрольних символів, які генеруються кодом Соломона-Ріда.

Передача:

- Передача всього блоку, включаючи контрольні символи, на приймач.

Отримання:

- Отримання блоку на приймачі.

Виявлення помилок:

- Застосування алгоритму виявлення помилок, такого як Syndrome Decoding, для визначення місць та кількості помилок.

Корекція помилок:

- Використання корекційних символів для виправлення помилок, якщо це можливо.

Цей процес дозволяє виправляти помилки при передачі даних і підтримує надійність інформації, навіть якщо під час передачі виникають деякі помилки.

Висновки за розділом 1

З огляду на методи збереження цілісності інформації можна зробити наступні висновки:

Чексума (Checksum): Чексуми є ефективними для виявлення помилок в даних під час їх передачі або зберігання, але вони не здатні виправити помилки. Вони дозволяють швидко виявити порушення цілісності даних, але не надають можливості виправити помилки.

Код Хемінга (Hamming Code): Коди Хемінга - це спеціальні блочні коди, які дозволяють виявити та виправити помилки в даних. Вони широко використовуються в сферах, де важлива надійність передачі даних і збереження інформації.

Криптографічні Хеш-функції: Криптографічні хеш-функції, такі як SHA-256 та MD5, використовуються для забезпечення цілісності та безпеки даних. Вони дозволяють створювати унікальні хеші для наборів даних і виявляти будь-які навіть найменші зміни у вхідних даних.

Захист від помилок і атак: Вибір методу збереження цілісності даних залежить від конкретного застосування. У важливих застосуваннях, де критична цілісність даних, використовуються більш складні та потужні засоби, такі як криптографічні хеш-функції та коди Хемінга.

Обмеження і ресурси: При виборі методу збереження цілісності даних необхідно враховувати обмеження ресурсів, такі як витрати на обчислення та обсяг додаткової інформації, яка потрібна для збереження цілісності.

Висновок полягає в тому, що вибір методу збереження цілісності інформації повинен бути обумовлений конкретними вимогами та контекстом. Для надійних систем, особливо там, де важлива корекція помилок, коди Хемінга і криптографічні хеш-функції є сильними інструментами для забезпечення цілісності даних.

РОЗДІЛ 2
ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДІВ І РОЗРОБКА АЛГОРИТМУ ПРОГРАМНОГО МОДУЛЮ

2.1 Обґрунтування обраних методів

В даному проекті проводиться вдосконалення системи забезпечення цілісності інформації, що передається в мережах передачі за допомогою комбінованого алгоритму перевірки інформації на правильність

Взагалі, система для нашого методу повинна відповідати таким вимогам:

- Система забезпечення цілісності інформації повинна працювати ефективно.
- Система повинна бути економічною, мінімізувати витрати енергії та ресурсів
- Система забезпечення цілісності інформації повинна виявляти, що дані не були змінені під час виконання будь-якої операції над ними, будь то передача, зберігання або подання
- Система забезпечення цілісності інформації має бути вбудованою та легко масштабованою.

Для вирішення завдань проекту були використані наступні засоби та методи:

- алгоритми для хешування MD5, SHA-2;
- алгоритм обчислення контрольної суми CRC, призначений для перевірки цілісності даних;
- код ECC, призначений для виявлення та виправлення помилок.

Ці методи були обрані як найпоширеніші і найпопулярніші на даний час. Також варто відмітити існування такого методу хешування як MD-4. Даний алгоритм все ще використовується для вирішення ряду задач збереження цілісності інформації. Проте цей метод є попередником більш вдосконаленого алгоритму хешування MD-5. Алгоритмічна різниця між цими двома методами не є суттєвою, але в результаті метод MD-5 відзначається збільшенням

надійності збереження даних та зменшенням вразливості до зовнішніх факторів. Саме тому було прийнято рішення не брати до порівняння алгоритм MD-4, так як він значно поступається у вищезазначених критеріях методам, які було обрано до порівняння.

Основною відмінністю між SHA-1 і SHA-2 є довжина хешу. У той час як SHA-1 є базовою версією хешу, що забезпечує коротший код із меншою кількістю унікальних комбінацій, SHA-2 або SHA-256 створює довший і, отже, складніший хеш.

Інші хеш функції не були додані через їх застарілість, непопулярність або копіювання вже представлених функцій.

CRC легко реалізувати і можна швидко виконати навіть для великих обсягів даних. Він надійний і може виявляти помилки, викликані шумом, перешкодами або іншими факторами, які можуть вплинути на точність цифрових сигналів.

Для поставленої задачі було обрано код Хемінга. Код Хемінга має просту структуру та не вимагає великої кількості обчислень. Це робить його легким для реалізації як у апаратурі, так і в програмному забезпеченні.

Також запропоновано використати комбінований метод. Метод полягає в поєднанні контрольної суми MD5 і коду Хемінга. Що повинно забезпечити швидкість і надійність в мережах передачі з частими помилками. Якщо виникне лише одна помилка, то кодування виправить її зекономивши час на пересиланні. І також якщо виникне кількість помилок яку алгоритм кодування не може виправити або виявити, то на останньому етапі алгоритм точно зможе виявити за допомогою контрольної суми.

2.2 Обґрунтування вибору мови програмування

Для вирішення поставленої задачі було обрано мову програмування Type script. Використання мови TS дозволяє розробити застосунок для веб браузеру. За допомогою цього можна встановити застосунок на локальному сервері навчального закладу і простіше надати доступ студентам ніж встановлювати

інструмент окремо на кожен компьютер. Також реалізація застосунку в вигляді веб додатку дозволяє спрощене розширення додатку та підтримку. Додатковими перевагами вибору Typescript можна виділити статичну типізацію яка дозволяє виявляти та виправляти помилки на етапі компіляції, що робить код менш помилковим та більш надійним та те, що TypeScript код можна скомпілювати до різних версій JavaScript, що дозволяє використовувати його в різних середовищах та браузерах[10][11].

2.3 Алгоритм для порівняльного аналізу методів цілісності інформації за допомогою експерименту і тренувального інструменту

Для проведення порівняння були обрані такі методи як:використання алгоритмів хешування MD5 та SHA-2 для розрахунку чек-сум, застосування циклічного надлишкового коду та використання методу кодування з виправленням помилок. У додаток до цього в експерименті також використовується комбінований метод, що поєднує в собі декілька зазначених підходів. Комбінований метод поєднує в собі метод обчислення контрольних сум MD5 та кодування Хемінга. Такий підхід дозволяє більш детально розглянути переваги та недоліки кожного методу в різноманітних сценаріях використання.

Нижче для ознайомлення представлено умови і алгоритм експерименту для методу контрольних сум.

Параметри які можна налаштувати:

- загальна кількість пересилань;
- можлива кількість помилок;
- вхідний текст або текстовий файл;

Алгоритм експерименту:

1. в якості вхідних даних використовується ідентичний текст для кожного алгоритму обчислення контрольних сум і представляємо його у двійковій формі;

2. розраховуємо контрольну суму для вхідної інформації;

3. імітуємо передачу цієї інформації. Під час імітації передачі інформації також імітується помилка в одному, двох або трьох бітах шляхом заміни їх на протилежні. Вірогідність спотворення біту під час імітації передачі — 50% якщо обрано 1 помилку. Якщо обрано 2 помилки 25% - 1 помилка і 25% - 2 помилки. Якщо обрано 3 помилки то ймовірність помилок 16,6% для однієї, двох та трьох помилок відповідно;

4. для отриманої послідовності розраховуємо контрольну сумму;

5. якщо контрольна сума прийнятої послідовності і контрольна сума для початкової послідовності не збігаються, то додаємо 1 в лічільник “повторної передачі” і переходимо на крок №3;

6. якщо контрольна сума прийнятої послідовності і контрольна сума для початкової послідовності збігаються то зараховуємо вдалу передачу і починаємо алгоритм спочатку поки кількість успішних передач не буде дорівнювати N;

7. у вікні результату отримаємо час за який пройшов експеримент, середня кількість повторних надсилань за ітерацію, загальна кількість пересилань, кількість успішних передач.

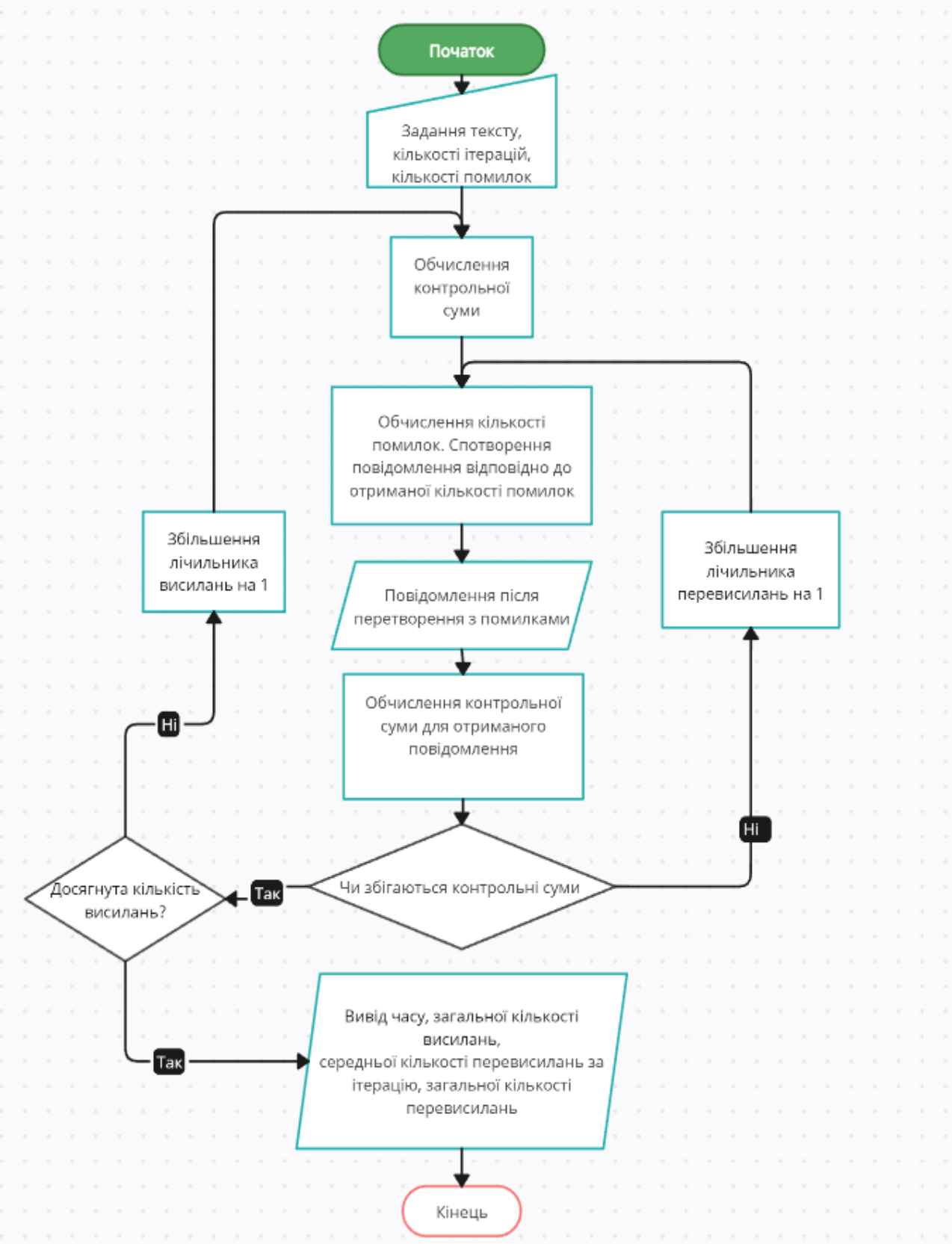


Рисунок 2.1 - Блок-схема алгоритму з використанням контрольної суми

Далі можна ознайомитись з умовами та алгоритмом експерименту для методу кодування Хемінга.

Параметри які можна налаштувати:

- загальна кількість пересилань;
- можлива кількість помилок;
- вхідний текст або текстовий файл;

Алгоритм експерименту:

1. в якості вхідних даних використовується текст і представляється у двійковій формі;
2. кодуємо цю послідовність за допомогою коду Хемінга;
3. імітуємо передачу цієї інформації. Під час імітації передачі інформації також імітується помилка в одному, двох або трьох бітах шляхом заміни їх на протилежні. Вірогідність спотворення біту під час імітації передачі — 50% якщо обрано 1 помилку. Якщо обрано 2 помилки 25% - 1 помилка і 25% - 2 помилки. Якщо обрано 3 помилки то ймовірність помилок 16,6% для однієї, двох та трьох помилок відповідно;
4. якщо була всього одна помилка, то код виправляє її, якщо 2 або 3, то помилка лишається не виправленою;
5. декодуємо отриману послідовність;
6. якщо була одна помилка, то код виправив її і передача вважається успішною. Починаємо алгоритм спочатку поки кількість передач не буде дорівнювати загальній кількості передач. Якщо 2 то код їх знайшов, але не виправив. В такому разі додаємо 1 в лічільник “повторної передачі” і переходимо на крок №3; якщо 3 то починаємо алгоритм спочатку не збільшуючи лічильник успішних передач;
7. у вікні результату отримаємо час за який пройшов експеримент, кількість пересилань, кількість успішних передач, загальна кількість пересилань.

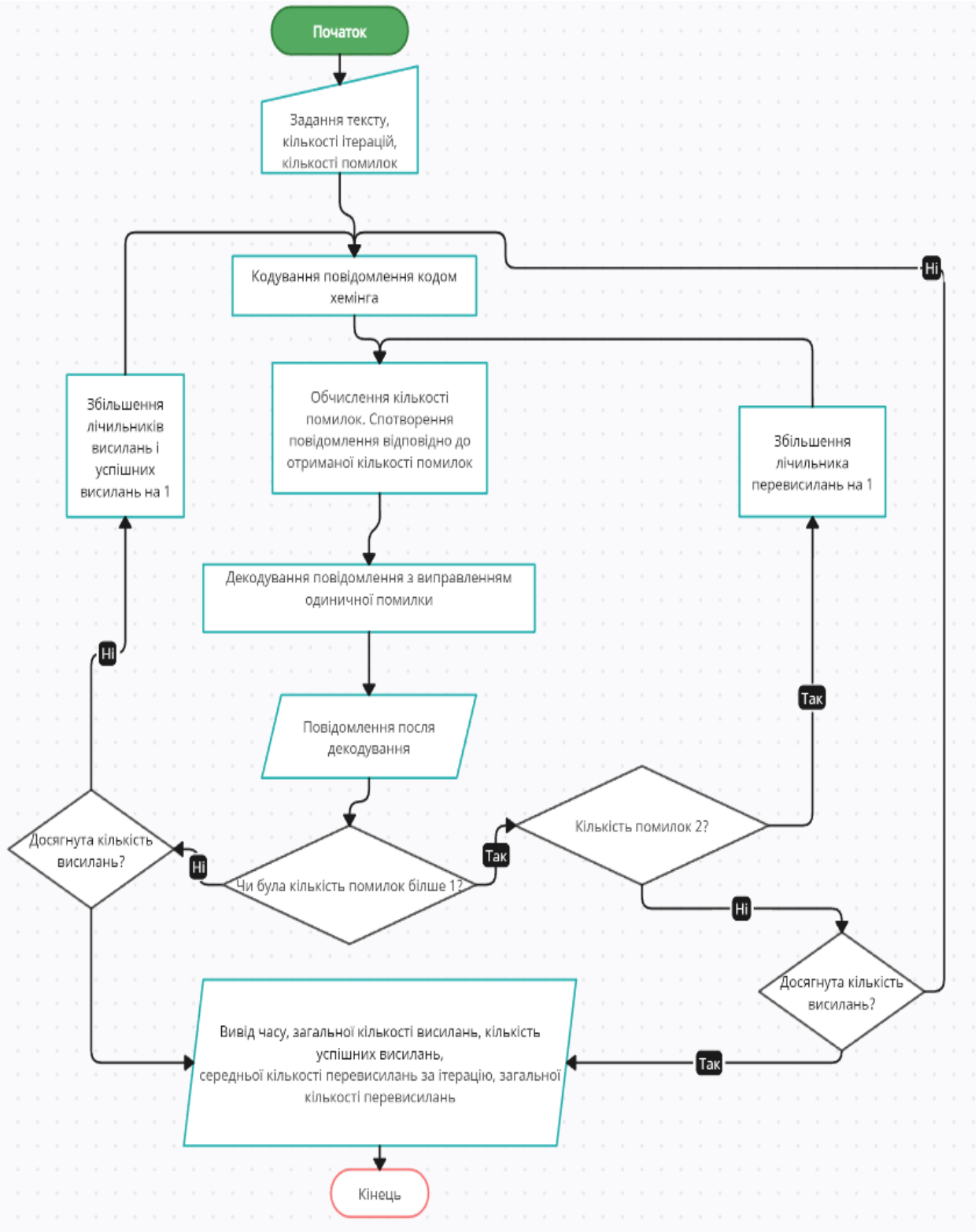


Рисунок 2.2 - Блок-схема алгоритму з використанням кодування

Для запропонованого комбінованого методи нижче вказані умови і алгоритм експерименту.

Параметри які можна налаштувати:

- загальна кількість пересилань;
- можлива кількість помилок;
- вхідний текст або текстовий файл;

Алгоритм експерименту:

1. в якості вхідних даних використовується текст і представляється у двійковій формі;
2. розраховуємо контрольну суму для вхідної інформації;
3. кодуємо вхідну послідовність за допомогою коду Хемінга;
4. імітуємо передачу цієї інформації. Під час імітації передачі інформації також імітується помилка в одному, двох або трьох бітах шляхом заміни їх на протилежні. Вірогідність спотворення біту під час імітації передачі — 50% якщо обрано 1 помилку. Якщо обрано 2 помилки 25% - 1 помилка і 25% - 2 помилки. Якщо обрано 3 помилки то ймовірність помилок 16,6% для однієї, двох та трьох помилок відповідно;
5. якщо була всього одна помилка, то код виправляє її, якщо 2 або 3, то помилка лишається невиправленою;
6. декодуємо отриману послідовність;
7. розраховуємо контрольну суму для отриманої послідовності;
8. якщо контрольна сума прийнятої послідовності і контрольна сума для початкової послідовності збігаються то зараховуємо вдалу передачу і починаємо алгоритм спочатку поки кількість успішних передач не буде дорівнювати N;
9. у вікні результату отримаємо час за який пройшов експеримент, кількість повторних надсилань, кількість успішних передач, загальна кількість пересилань.

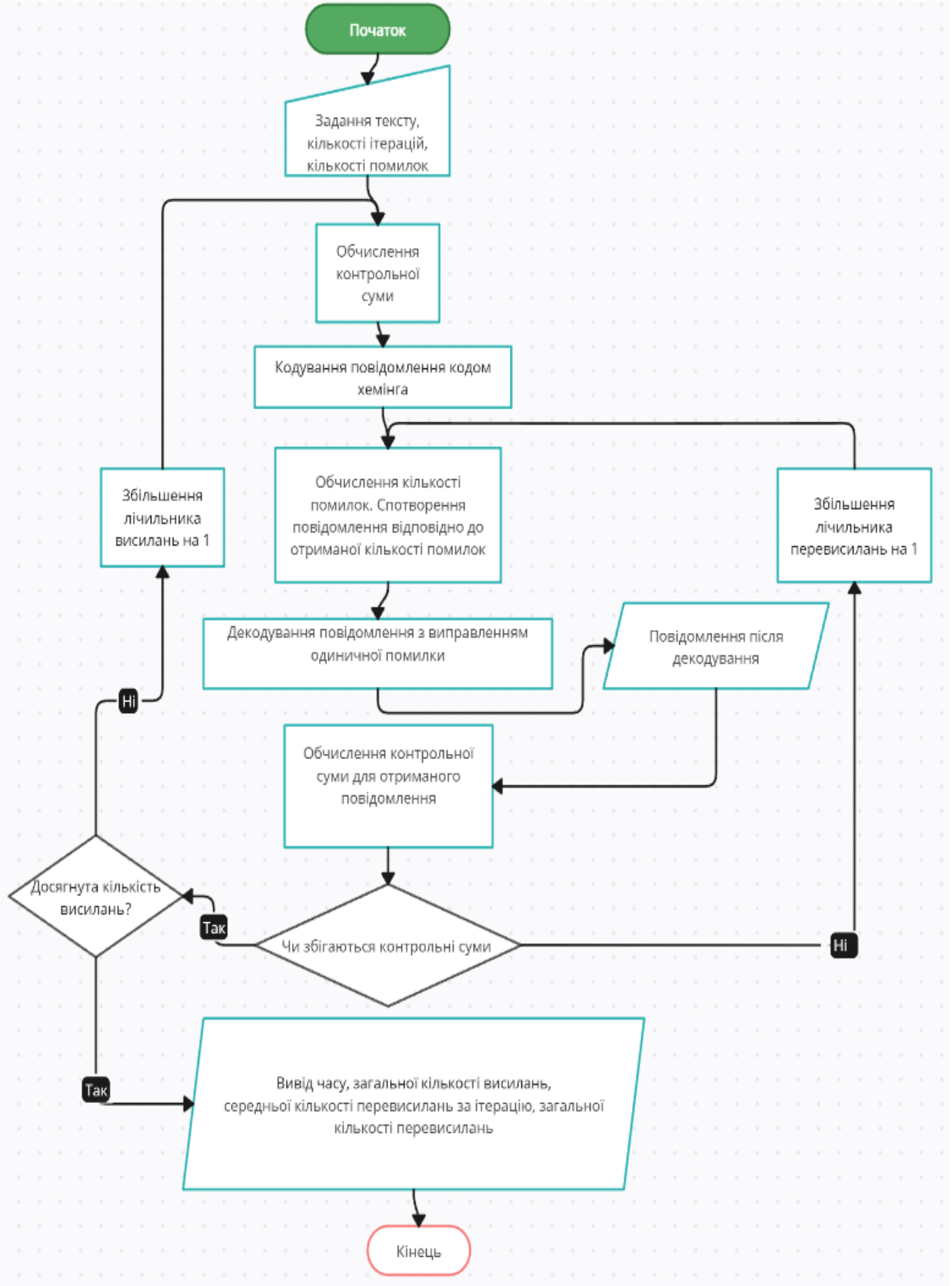


Рисунок 2.3 - Блок-схема алгоритму з використанням комбінованого методу

Висновки за розділом 2

В даному розділі було описано вимоги до системи в якій буде працю комбінований метод:

- повинна працювати ефективно.
- повинна бути економічною
- повинна виявляти, що дані не були змінені
- має бути вбудованою та легко масштабованою.

Було обґрунтовано вибір методів забезпечення цілісності інформації для експерименту та запропоновано комбінований метод. Для порівняльного дослідницького експерименту і побудови навчального інструменту було обрано: MD5, SHA-2, CRC, кодування Хемінга, комбінований метод.

Обґрунтовано вибір мови програмування для побудови програмної моделі з урахуванням важливості оптимізації, ефективності та зручності. Обравши TypeScript для розробки, можна зручно встановити веб-застосунок на локальному сервері навчального закладу. TypeScript дозволяє створювати надійні та розширювані веб-додатки, забезпечуючи статичну типізацію та можливість компіляції до різних версій JavaScript. Подано докладні блок-схеми алгоритмів, які використовуються під час проведення експерименту. Здійснено порівняння різних методів забезпечення цілісності інформації.

РОЗДІЛ 3

РОЗРОБКА ДОДАТКУ ТА ІНСТРУКЦІЙ КОРИСТУВАЧА

3.1 Функціонал додатку та інструкції користувача

Програмна модель являє собою веб-застосунок, в якому кінцевий користувач може вводити власні налаштування для подальшого проведення експерименту, отримати додаткову інформацію про використані методи забезпечення цілісності інформації, а також отримати результати експерименту після його проведення. Структурно застосунок складається з двох варіантів його застосування, для кожного з яких маються окремі вкладки (Рис. 3.1):

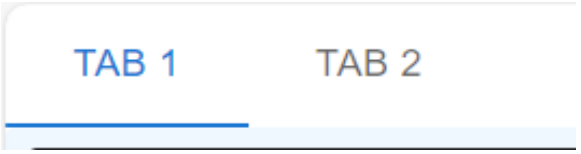


Рисунок 3.1 - Вибір робочої вкладки

Перша вкладка відповідає за ознайомлення кінцевого користувача результатами обробки інформації, що передається, за допомогою хеш-функцій. Пейшовши на першу вкладку, користувачу надається можливість ввести текст у відповідному полі або завантажити текстовий файл. Після цього необхідно натиснути кнопку “Begin” початку обчислень (Рис 3.2). Для початку обчислень обов’язково необхідно ввести текст або завантажити текстовий файл.

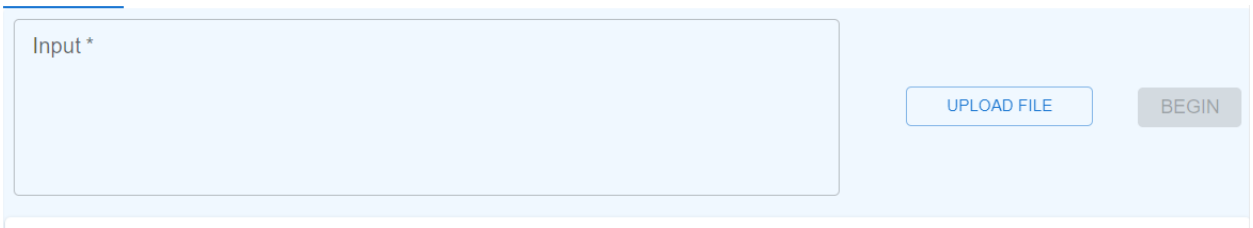


Рисунок 3.2 - Поля вводу вкладка 1

Нижче нижче розташовані поля виводу для кожного з обраних варіантів обчислень контрольної суми (Рис. 3.3). Кнопка “I”, що розташована біля кожної з назв алгоритмів, при натисканні на неї переводить користувача на нову вкладку, де розміщена інформація про обраний алгоритм.

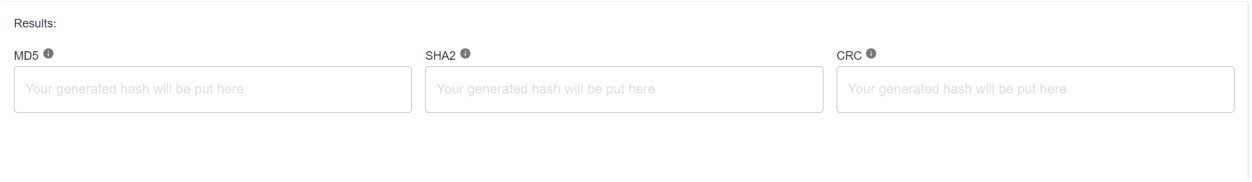


Рисунок 3.3 - Поля виводу вкладка 1

Друга вкладка дає можливість користувачу повноцінно провести експеримент зі згаданими алгоритмами. Вкладка, аналогічно до першої, містить в собі поле для вводу тексту, кнопку для завантаження файлу, та кнопку “Begin” для початку розрахунків. Проте для даного варіанту експерименту існують додаткові поля: поле вводу кількості ітерацій, випадаючий список вибору максимальної кількості помилок (Рис 3.4)

Рисунок 3.4 - Поля вводу вкладка 2

Кількість ітерацій відповідає за число “початково відправлених” пакетів даних для проведення експерименту. Число максимальної кількості помилок надає змогу користувачеві власноруч обрати максимальну кількість помилок для генерації “пошкоджених” даних, що будуть використовуватись в ході експерименту. Для запуску експерименту користувачеві необхідно завантажити текстовий файл, обрати кількість ітерацій та кількість помилок.

Під блоком вхідних даних, аналогічно до першого варіанту, розташовані поля виводу для методів обчислення контрольних сум, коду з виявленням та

виправленням помилок та комбінованого методу (Рис. 3.5). Кнопка “I”, що розташована біля кожної з назв алгоритмів, аналогічно до такої ж кнопки в першому режимі використання програми, при натисканні на неї переводить користувача на нову вкладку, де розміщена інформація про обраний алгоритм.

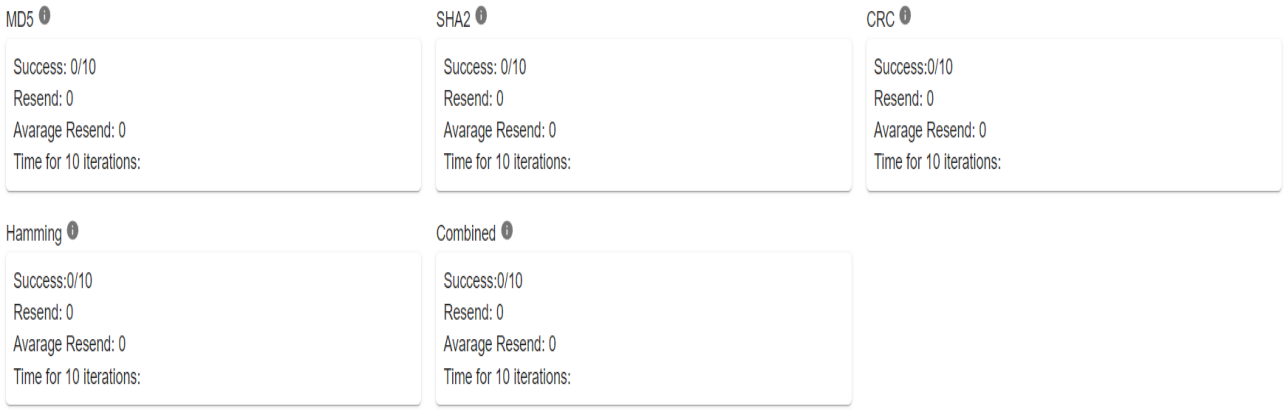


Рисунок 3.5 - Поля виводу вкладка 2

Кожен метод в результаті має такі показники як: кількість успішних передач пакету, кількість повернутих пакетів, та час, що було затрачено на обчислення всіх ітерацій.

3.2 Результати експерименту

Експеримент був розділений на дев’ять раундів. Кожен із раундів повторюється 10 разів. Вхідна інформація, кількість ітерацій і максимальна кількість помилок налаштовуються для кожного раунду індивідуально. Для порівняння використовуються показники кількості вдалих передач, загального часу проходження всіх ітерацій, кількість повторних надсилань та середню кількість повторних надсилань за ітерацію.

Перший раунд:

Кількість ітерацій:1000

Максимальна кількість помилок:1

Вхідна інформація: 1 кб

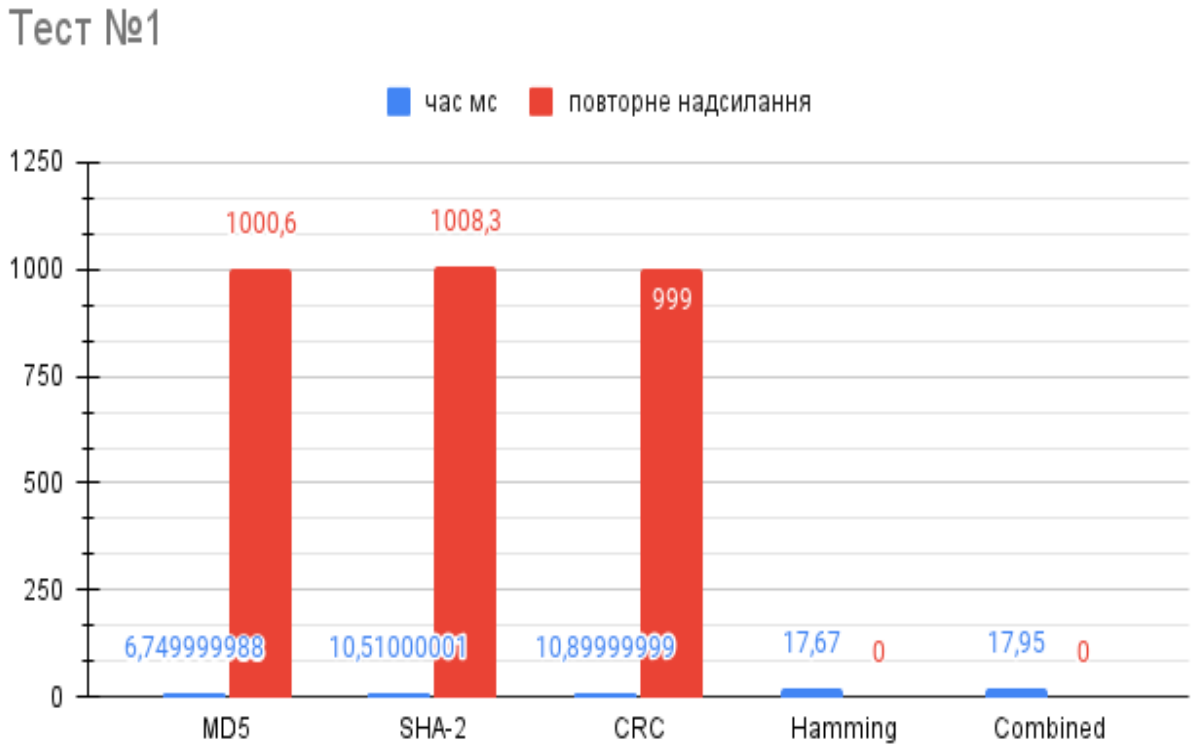


Рисунок 3.6 Результат першого раунду випробувань

В першому раунді тесту найшвидшим є метод контрольної суми MD5. Найдовшим комбінований метод. Також можна прослідкувати що в середньому було здійснено 1000 повторних надсилань для кожного методу крім методів кодування і комбінованого методу, які можуть виправити одинарну помилку і не потребують повторного надсилання даних.

Другий раунд:

Кількість ітерацій:1000

Максимальна кількість помилок:2

Вхідна інформація: 1 кб

Рисунок 3.7 Результат другого раунду випробувань

В другому раунді можна прослідкувати за тим, що метод контрольних сум MD5 все ще є найшвидшим. При майже однаковій кількості повторних надсилань цей метод майже вдвічі швидший ніж варіант з контрольною сумою SHA-2. Методи кодування і комбінований метод працюють майже втричі повільніше, але мають майже вдвічі менше повторних надсилань.

Третій раунд:

Кількість ітерацій:1000

Максимальна кількість помилок:3

Вхідна інформація: 1 кб

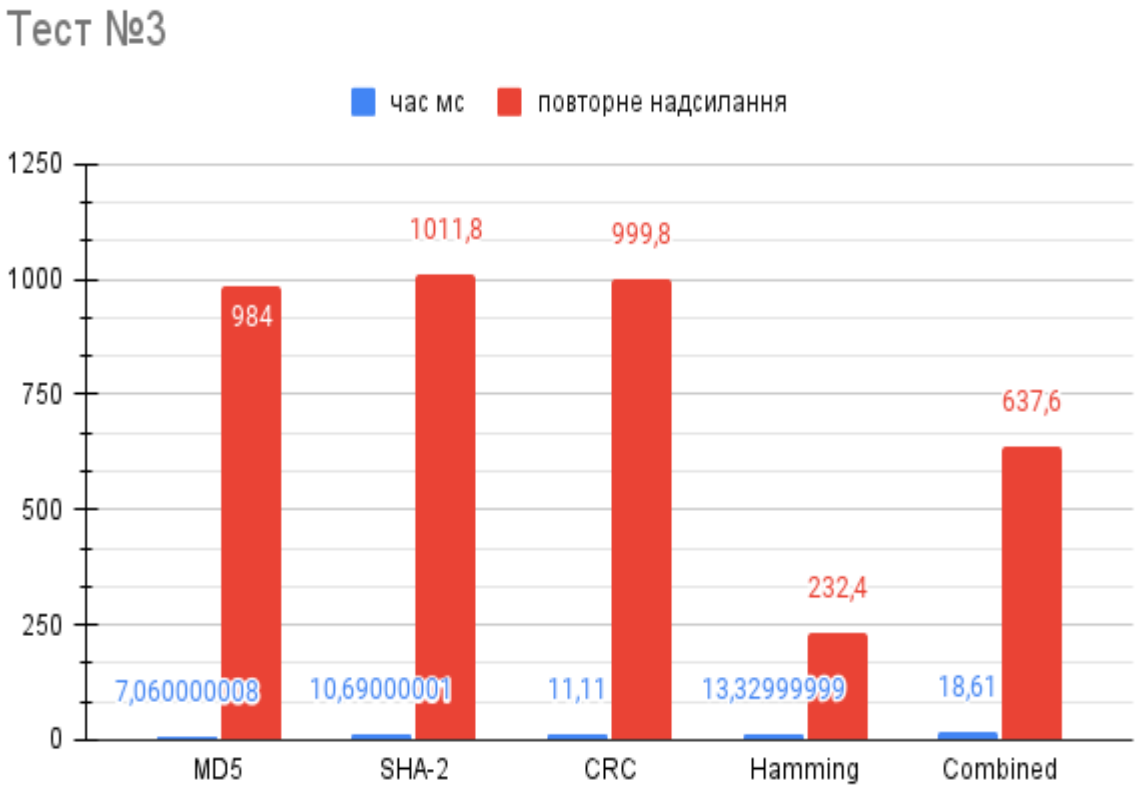


Рисунок 3.8 - Результат третього раунду випробувань

В третьому раунді тесту можна побачити що різниця в кількості повторних надсилань між методами контрольної суми та комбінованим методом скоротилась і тепер комбінований метод має лише на третину менше повторних надсилань. Також потрібно відзначити що метод кодування має майже втричі менше повторних надсилань ніж комбінований і майже в 4,5 разів ніж метод контрольних сум, але слід враховувати що в середньому триста повідомлень з тисячі не були прийняті вірно через те що в даному випадку використовується метод кодування, який не розпізнає 3 помилки.

Четвертий раунд:

Кількість ітерацій:10000

Максимальна кількість помилок:1

Вхідна інформація: 1 кб

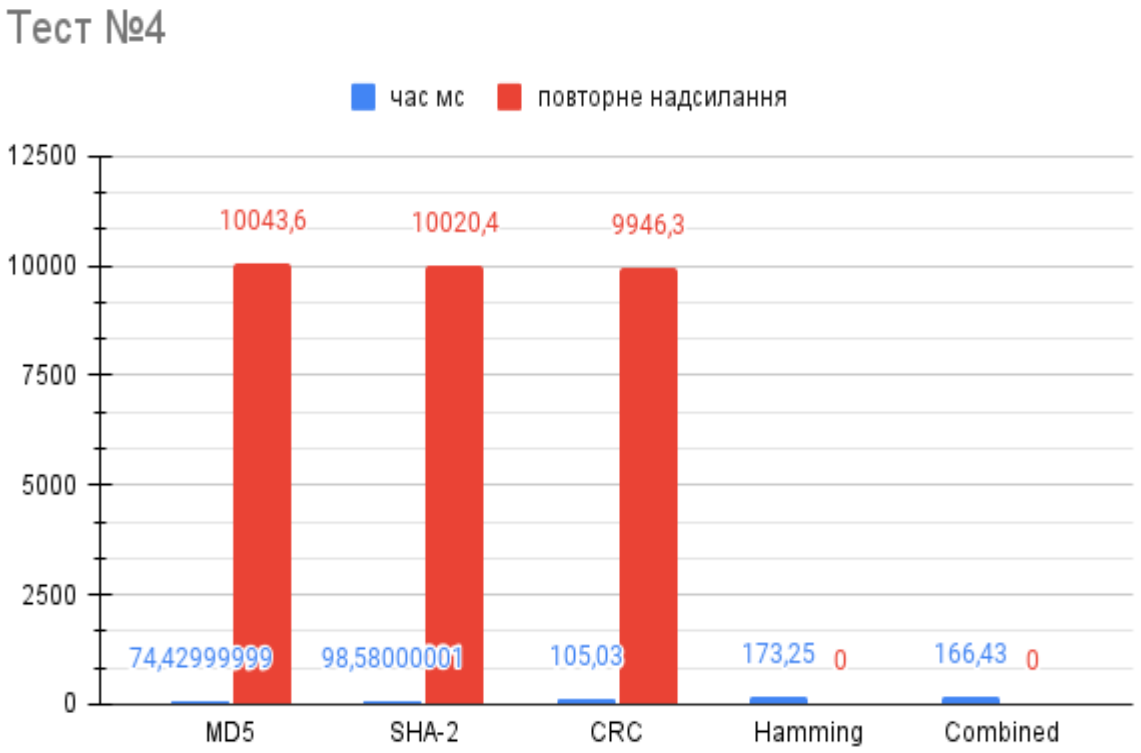


Рисунок 3.9 - Результат четвертого раунду випробувань

В четвертому раунді кількість ітерацій була збільшена до десяти тисяч. Як результат можна побачити скорочення відставання між найшвидшим методом і найповільнішими. При збільшенні ітерацій методом MD5 швидший за комбінований лише в 2,3 рази проти майже 3 при тисячі ітерацій.

П’ятий раунд:

Кількість ітерацій:10000

Максимальна кількість помилок:2

Вхідна інформація: 1 кб

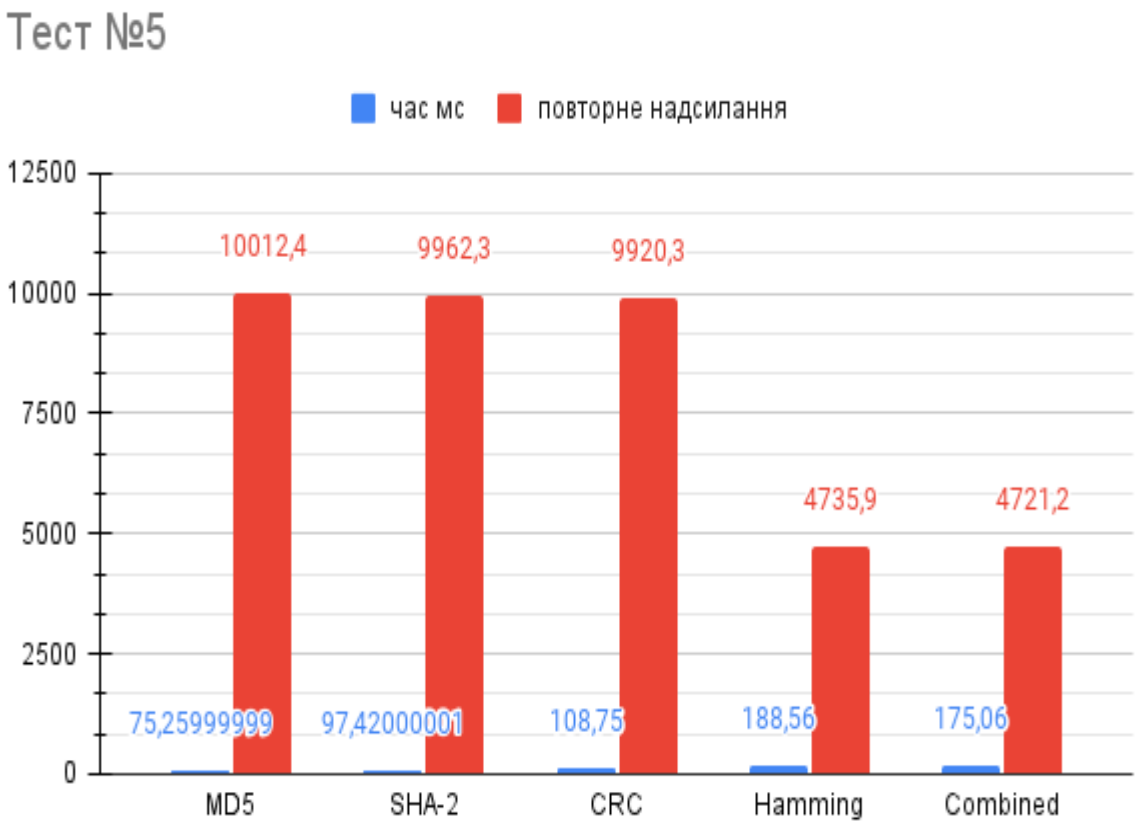


Рисунок 3.10 - Результат п’ятого раунду випробувань

В п'ятому раунді можна відзначити, що використання методу контрольних сум MD5 є найбільш ефективним з точки зору швидкості. Навіть при більшій кількості повторних надсилань цей підхід на третину швидший, ніж альтернативні варіанти із контрольною сумою SHA-2 та CRC . Методи кодування та комбінований метод, хоча працюють майже вдвічі повільніше, відрізняються практично вдвічі меншою кількістю повторних надсилань.

Шостий раунд:

Кількість ітерацій:10000

Максимальна кількість помилок:3

Вхідна інформація: 1 кб

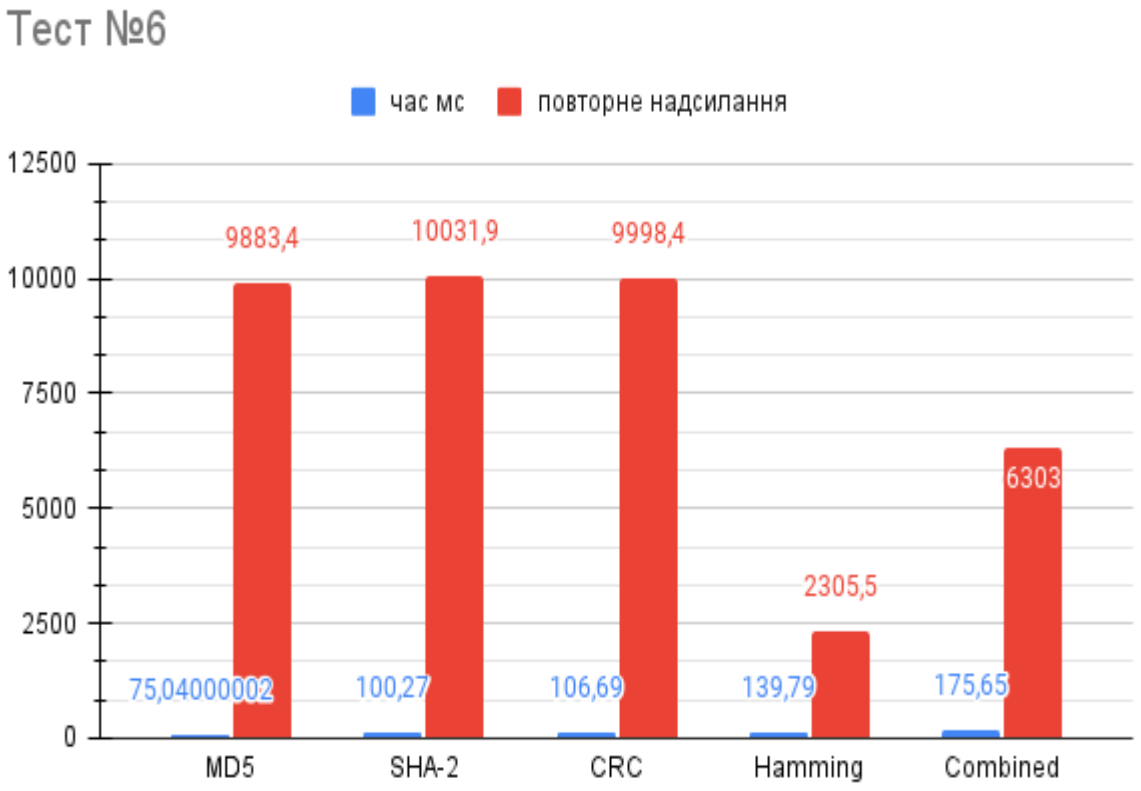


Рисунок 3.11 - Результат шостого раунду випробувань

У шостому тестовому раунді помітно, що різниця в кількості повторних надсилань між методами контрольної суми та комбінованим методом знову зменшилась, і тепер комбінований метод має лише на третину менше повторних надсилань при різниці в часі в трохи більше ніж в 2 рази . Метод кодування знову демонструє найменшу кількість повторних надсилань з урахуванням, що в середньому три тисячі повідомлень з десяти тисяч не були прийняті правильно за цей раунд.

Сьомий раунд:

Кількість ітерацій:5

Максимальна кількість помилок:1

Вхідна інформація: 1 мб

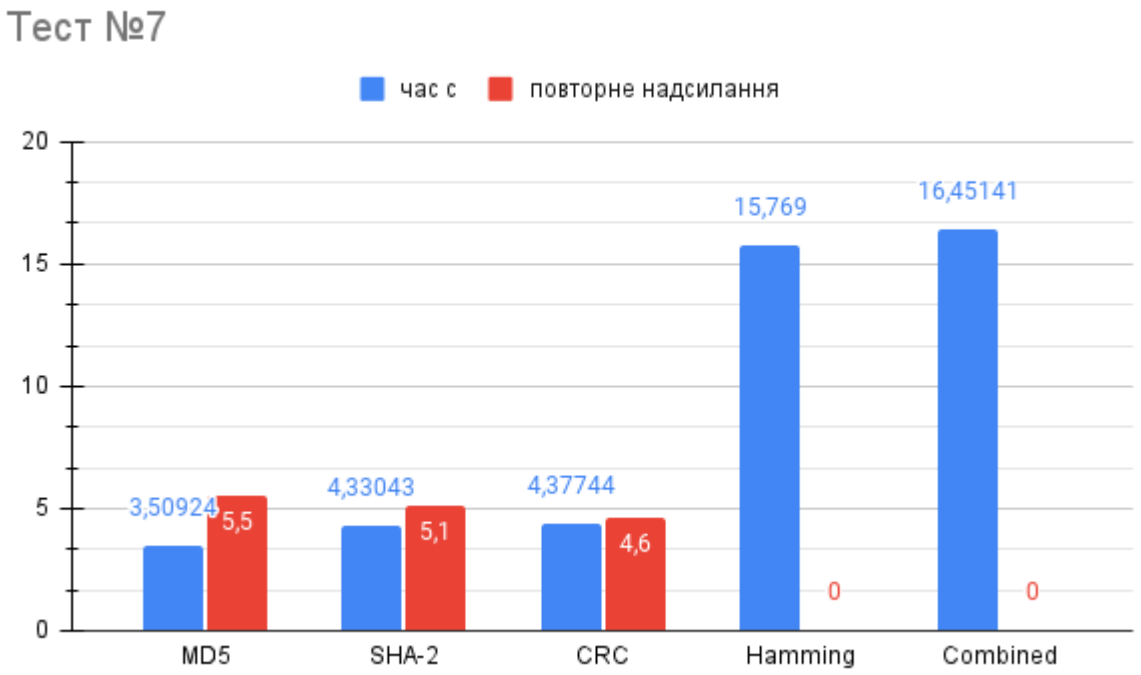


Рисунок 3.12 - Результат сьомого раунду випробувань

В сьомому раунді суттєво збільшилась різниця в часі між методом кодування, комбінованим методом та методами контрольних сум. Так можна побачити що методи контрольних сум не мають сильно велику різницю між собою, хоч MD5 і трохи випереджає конкурентні методи. Метод кодування та комбінований метод працюють майже в 5 разів повільніше в порівнянні з найшвидшим з представлених методом контрольних сум.

Восьмий раунд:

Кількість ітерацій:5

Максимальна кількість помилок:2

Вхідна інформація: 1 мб

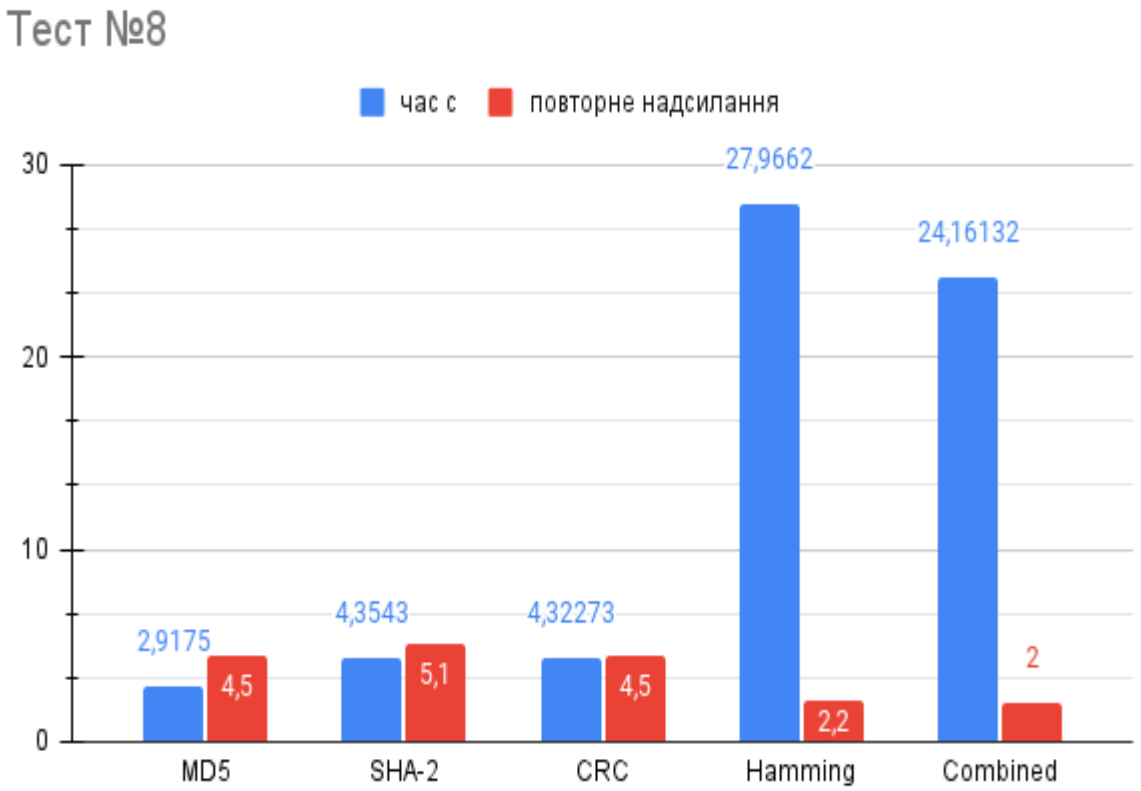


Рисунок 3.13 - Результат восьмого раунду випробувань

Восьмий раунд дає нам змогу прослідкувати за тим як росте різниця в швидкодії методів при наявності повторних надсилань за умови обробки збільшеного об’єму інформації. Так різниця між методом контрольних сум та методом кодування збільшилась до різниці у майже 10 раз і у 8 раз якщо порівнювати комбінований метод та метод MD5.

Дев’ятий раунд:

Кількість ітерацій:5

Максимальна кількість помилок:3

Вхідна інформація: 1 мб

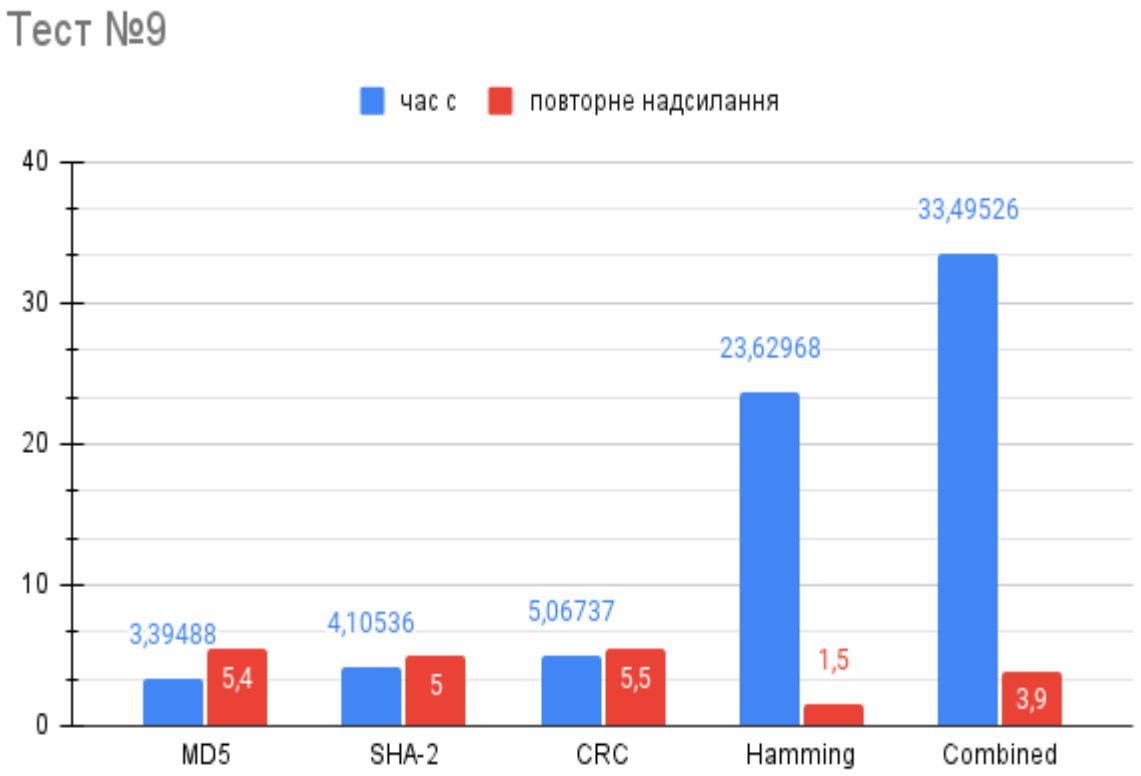


Рисунок 3.14 - Результат дев’ятого раунду випробувань

У дев'ятому тестовому раунді очевидно, що різниця в кількості повторних надсилань між методами контрольної суми та комбінованим методом знову зменшилась, при цьому відмічається різниця в часі збільшилась до різниці в 10 разів. Метод кодування знову показує найменшу кількість повторних надсилань, при умові, що в середньому третина повідомлень не були прийняті правильно у цьому раунді, а часу витрачено майже в шість разів більше ніж в методах контрольних сум та всього майже на третину менше ніж комбінований метод.

Після всіх раундів експерименту можна прослідкувати такі залежності для основних методів. Перша залежність полягає між часом виконання та кількістю повторних надсилань(Рис. 3.15). Друга залежність між часом виконання та розміром даних(Рис. 3.16). Третя залежність між часом виконання та кількістю ітерацій(Рис. 3.17).

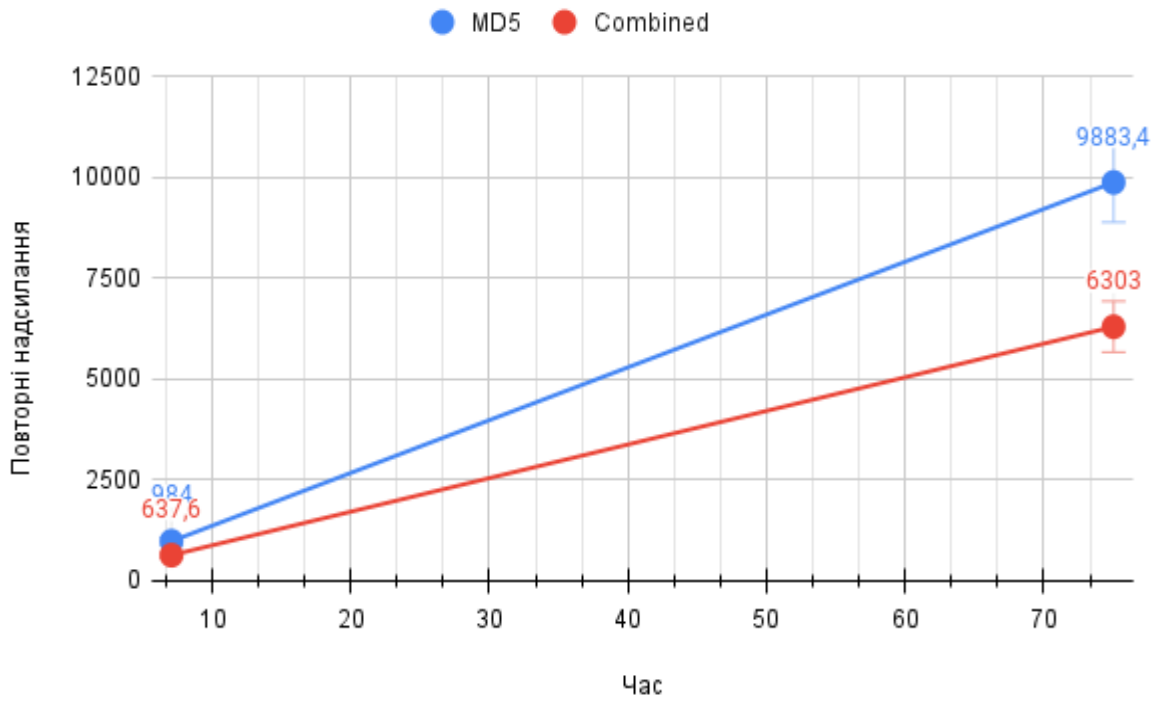


Рисунок 3.15 - Залежність часу від від кількості повторних надсилань

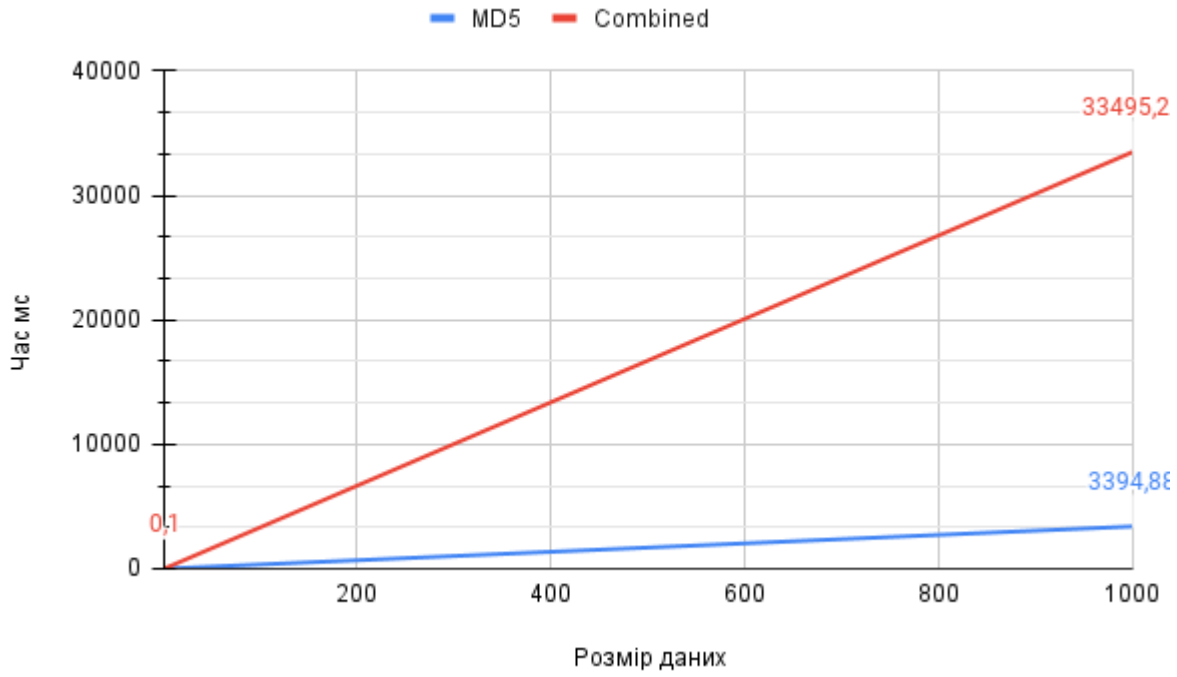


Рисунок 3.16 - Залежність часу від розміру даних

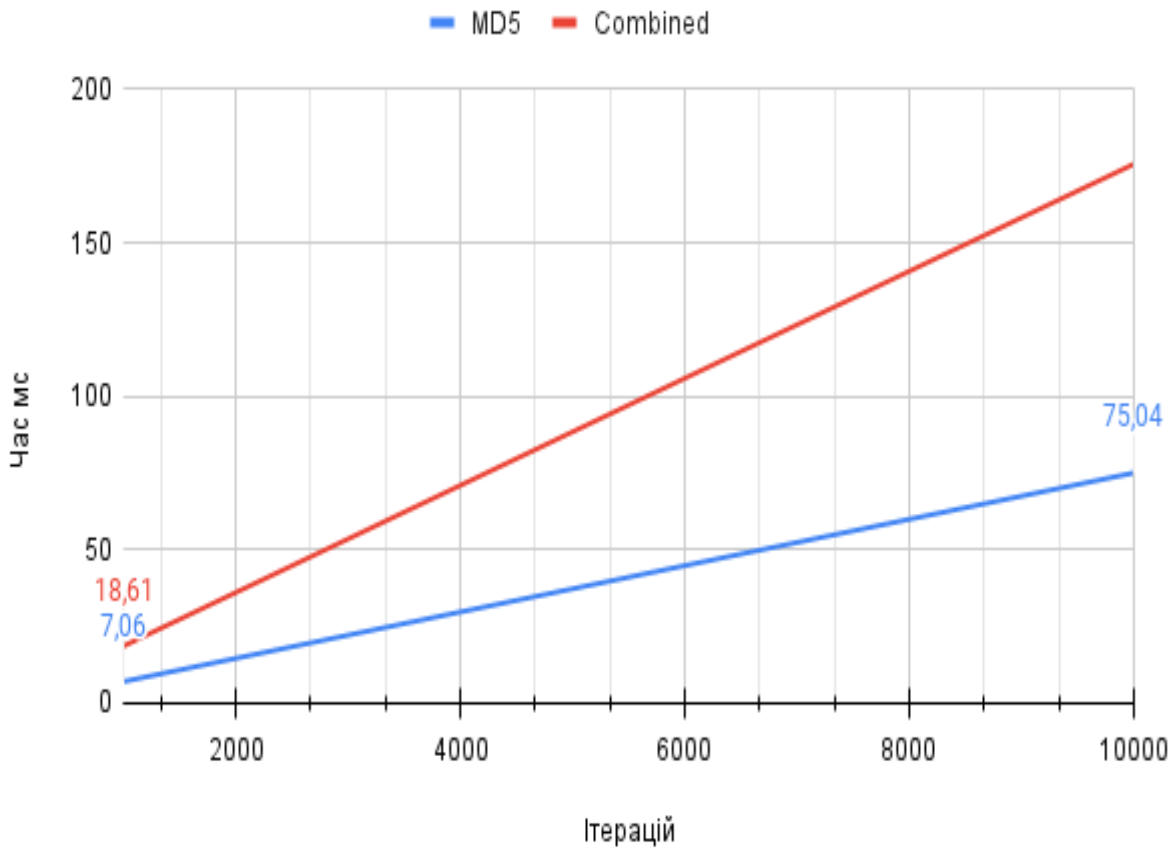


Рисунок 3.17 - Залежність часу виконання від кількості ітерацій

Як результат можна чітко виділити швидкодію методу контрольних сум з використанням алгоритму MD5. Метод показує кращу швидкість в усіх раундах випробувань, але потрібно також відмітити що для вдалої передачі всіх повідомлень також потрібна велика кількість повторних надсилань якщо ймовірність помилки досить висока. В даному експерименті не бралось до уваги сама швидкість передачі, яка суттєво змінює результат. Припустимо для кожної передачі потрібна одна секунда. Так якщо взяти результати з раунду №3 то для передачі з методом MD5 нам знадобиться 1984.007 секунд, а для комбінованого методу 1637,618. Якщо ж брати до уваги роботу з великим обсягом інформації за раз, то спираючись на дев’ятий раунд експерименту можна побачити, що для методу контрольної суми операція передачі інформації для 5 ітерацій зайняла б 13,79 секунд, а для комбінованого методу 42,39 секунди.

Аналіз результатів дослідження дозволяє визначити як переваги, так і недоліки комбінованого методу у порівнянні з простим методом виправлення помилок та методом контрольних сум. У сценарії великої кількості помилок комбінований метод виявився більш ефективним завдяки використанню повторних надсилань при будь-якій кількості помилок та здатності виправляти одиночні помилки на льоту. Це сприяє зменшенню кількості повторних надсилань, оптимізації використання пропускної здатності та прискоренню передачі даних. Також слід відзначити, що комбінований метод виявився швидшим в порівнянні з методом контрольних сум при передачі невеликої кількості інформації. У випадку, коли потрібно обмежити кількість повторних надсилань, різниця між ними стає особливо відчутною, підкреслюючи перевагу комбінованого методу. Проте, при значному розширенні обсягу переданих даних за один раз, виграш у часі, отриманий завдяки економії на повторних надсиланнях, компенсується збільшеним часом обробки цієї інформації. У такому випадку, використання методу контрольних сум може стати більш вигідним

Висновок до розділу 3

У третьому розділі дослідницької роботи наведено докладні результати проведених досліджень, а також надані вичерпно описи можливостей програмної моделі та детальні інструкції для користувачів.

Програмна модель, яку було створено для проведення дослідження, має в собі ряд корисних та практичних засобів для ознайомлення користувача з різними алгоритмами для збереження цілісності інформації в мережі, а також інструменти для запуску експериментів та проведення обчислень. В результаті проведених досліджень та експериментів було виявлено як переваги, так і недоліки кожного з методів.

Зокрема, комбінований метод продемонстрував свою ефективність в умовах великої ймовірності виникнення помилок та при передачі невеликої

кількості інформації за один раз. В інших сценаріях метод контрольних сум виявився більш кращим вибором. Експеримент також підтвердив, що сам метод контрольних сум MD5 виявився швидшим за будь-яких умов.

Ці результати стають важливим джерелом інформації для користувачів, які розглядають використання різних методів в області передачі та обробки даних.

ВИСНОВОК

В даній роботі було проведене вдосконалення системи забезпечення цілісності інформації, що передається в мережах передачі за допомогою комбінованого алгоритму перевірки інформації на правильність. Було проведено обгрунтування використання комбінованого методу. Також в ході роботи була проведена розробка моделі тренувального програмного інструментарію для подальшого ознайомлення студентів з роботою методів забезпечення цілісності передачі інформації.

Для досягнення поставленої мети були вирішені наступні задачі:

- проведений аналіз методів забезпечення цілісності інформації;
- обрано програмне забезпечення для побудови застосунку;
- визначено набір методів забезпечення цілісності інформації;
- написання статті для публікації у науковому виданні;
- розробка веб застосунку
- проведено порівняння методів забезпечення цілісності за допомогою розробленого застосунку

У даному дослідженні визначено та обґрунтовано вибір методів, які використовуються для проведення експериментів і порівняльного аналізу. Серед обраного інструментарію входять такі методи як MD5, SHA-2, CRC, кодування Хемінга та комбінований метод. Цей вибір зроблено з урахуванням потреб дослідження та необхідності побудови навчального інструменту, спрямованого на аналіз цих методів.

Обґрунтовано вибір мови програмування для побудови програмної моделі. Розглянуто алгоритм проведення експерименту порівняння методів забезпечення цілісності інформації. В результаті аналізу якості та швидкодії випробувань розробленої моделі можна зробити висновок, що розроблена модель однозначно поліпшує процес розуміння роботи методів забезпечення цілісності інформації, а також дозволяє провести ряд випробувань для

демонстрації переваг і недоліків комбінованого методу в порівнянні з іншими методами і їх обгрунтування

Результати експерименту показали переваги і недоліки комбінованого методу в порівнянні з простим методом виправлення помилки та методом контрольних сум. Так за рахунок повторної передачі інформації при виникненні великої кількості помилок, якщо нам важливо отримати інформацію повністю комбінований метод показав себе краще ніж метод виправлення помилки, за рахунок того, що він має повторні надсилання при будь-якій кількості помилок, а також може виправити одиничні помилки що знижує загальну кількість повторних надсилань. Також перевагу над методом контрольних сум при умові передачі невеликої кількості інформації. При нижчій кількості повторних надсилань різниця стає дуже відчутною. Однак якщо збільшувати об’єм надісланих даних за раз, то зекономлений час на повторних надсиланнях повністю нівелюється часом обробки цієї інформації і в такому випадку ліпше використовувати метод контрольних сум.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bonaventure O. Computer Networking: Principles, Protocols and Practices : textbook Creative Commons Attribution : October 31, 2011. 278 p.
2. Ferguson N., Schneier B. Practical Cryptography: textbook: John Wiley & Sons 2003. 432 p.
3. Stallings W. Cryptography and Network Security: Principles and Practice : textbook 7th ed. Pearson Education 2020. 767 p.
4. Stallings W. Network Security Essentials: Applications and Standards : textbook 6th ed Pearson Education 2017. 432 p.
5. Rivest R. The MD5 Message-Digest Algorithm : веб-сайт. URL: https://datatracker.ietf.org/doc/html/rfc1321 (дата звернення: 08.10.2023).
6. Sklavos N. Koufopavlou O. On the hardware implementations of the SHA-2 (256, 384, 512) hash functions : proceedings of IEEE International Symposium on Circuits & Systems, Vol. V, pp. 153-156, Thailand, May 25-28, 2003. 4 p.
7. Hamming Richard W. The Art of Doing Science and Engineering : textbook Taylor & Francis e-Library, 2005. 227 p.
8. Andrew S. Tanenbaum, David J. Wetherall. Computer science : textbook 5th ed. Pearson Education, Inc 2011. 962 p.
9. Shu Lin Error control coding : textbook Prentice-Hall, Inc., Englewood Cliffs, N.J. 07632 1983 624 p.
10. Richard E. Blahut Algebraic Codes for Data Transmission : textbook Cambridge university press 2003. 473 p.
11. J. H. van Lint Introduc-tion to Coding Theory : textbook 3rd rev. and expanded ed. springer-Verlag Berlin Heidelberg 1999. 243 p.
12. EtherCAT Technology Group: веб-сайт. URL: https://www.ethercat.org/default.htm (дата звернення: 08.10.2023).
13. FIPS 180-2, Secure Hash Standard : веб-сайт. URL: https://csrc.nist.gov/files/pubs/fips/180-2/final/docs/fips180-2.pdf (дата звернення: 09.10.2023).

14. Getting Started – React : веб-сайт. URL: https://legacy.reactjs.org/docs/getting-started.html (дата звернення: 11.10.2023).
15. TypeScript: JavaScript With Syntax For Types : веб-сайт. URL: https://www.typescriptlang.org/ (дата звернення: 01.10.2023).
16. JavaScript Tutorial (w3schools.com) : веб-сайт. URL: https://www.w3schools.com/js/ (дата звернення: 02.11.2023).
17. Visual Studio Code – Code Editing. Redefined : веб-сайт. URL: https://code.visualstudio.com/ (дата звернення: 01.10.2023).
18. React Native Tutorial: Build app with JavaScript : веб-сайт. URL: https://www.raywenderlich.com/126063/reactnative-tutorial (дата звернення: 12.10.2023).
19. Офіційний сайт Stack Overflow : веб-сайт. URL: https://insights.stackoverflow.com/survey/2015#tech-super. (дата звернення: 12.11.2023).
20. Creately | Visual Collaboration & Diagramming Platform : веб-сайт. URL: https://app.creately.com/ (дата звернення: 13.11.2023).
21. GitHub: Let's build from here – GitHub : веб-сайт. URL: https://github.com/ (дата звернення: 02.11.2023).

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Факультет комп'ютерних наук
Кафедра теоретичної та прикладної системотехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) **Магістр**
Галузь знань: **15 – Автоматизація та приладобудування**
Спеціальність: **151 – «Автоматизація та комп'ютерно-інтегровані технології»**

ЗАТВЕРДЖУЮ
Завідувач кафедри теоретичної та прикладної системотехніки
_____д.т.н., проф. Шматков С. І.

«08 » грудня 2022 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Маницький Сергій Сергійович
(прізвище, ім'я, по батькові студента)

1. Тема роботи «Система забезпечення цілісності інформації, що передається в мережах передачі даних» _____

керівник роботи Мірошник Марина Анатоліївна д.т.н. , професор _____,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «10» листопада 2023 року № 4101-5/3197

2. Строк подання студентом роботи 28 .11.2023

3. Перелік питань, які потрібно розробити)
1. Огляд та аналіз існуючих методів забезпечення цілісності інформації _____
2. Розробка моделі візуалізації _____
3. Дослідження моделі _____
4. Підготовка і оформлення пояснювальної записки _____

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1.	Аналіз сучасних методів забезпечення цілісності даних	08.12.2022-10.01.2023
2.	Вибір методів забезпечення цілісності даних	11.01.2023-13.02.2023
3.	Розробка моделі візуалізації системи забезпечення цілісності даних	14.02.2023-01.08.2023
4.	Налаштування та тестування моделі	02.08.2023-04.09.2023
5.	Коригування після тестування	05.09.2023-10.10.2023
6.	Підготовка наукової доповіді	26.05.2023-17.10.2023
7.	Доповідь на студентській науково-технічній конференції в ХНУ ім. Каразіна	27.10.2023-28.10.2023
8.	Підготовка пояснювальної записки до перевірки на наявність запозичень	03.09.2023-28.11.2023
9.	Представлення кваліфікаційної роботи науковому керівнику кваліфікаційної роботи та рецензенту	29.11.2023-25.12.2023

5. Дата видачі завдання _08.12.2022_____

Студент __Манищький С.С.___ _____
ініціали, прізвище підпис

Керівник роботи __ Мірошник М.А.___ _____
ініціали, прізвище підпис

Додаток Б

ІНДИВІДУАЛЬНЕ ТЕХНІЧНЕ ЗАВДАННЯ.

Технічне завдання на розробку програмного виробу «Система забезпечення цілісності інформації, що передається в мережах передачі даних»

1.	Введення	Назва: Система забезпечення цілісності інформації що передається в мережах передачі даних Область застосування:забезпечення цілісності інформації в мережах передачі інформації.
2.	Підстава для розробки	1) Навчальний план ФКН за фахом 151 – «Автоматизація та комп'ютерно-інтегровані технології» 2) Завдання на дипломну роботу магістра №04101-5/3197 від 10.11.2023 (представити як Додаток А до пояснювальної записки до кваліфікаційної роботи);
3.	Призначення розробки	1) Створення web додатку для подальшого ознайомлення студентів з роботою методів забезпечення цілісності передачі інформації. 2) Модель призначена для використання в навчальному процесі 3) Вихідні дані розробки: web додаток для подальшого ознайомлення студентів з роботою методів забезпечення цілісності передачі інформації
4.	Технічні вимоги до програмного виробу	Програма повинна: 1) Представляти з себе web-додатку з візуалізацією роботи методів забезпечення інформації 2) Надавати можливість вводити інформацію для обробки шляхом ручного вводу 3) Надавати можливість вводити інформацію через завантаження файлу 4) Надавати можливість налаштовувати “експеримент” шляхом встановлення кількості ітерацій і максимальної кількості помилок 5) Забезпечувати захист від некоректних дій; 6) Забезпечувати вивід результатів обчислень 7) Забезпечити вивід додаткової інформації 8) Вимоги до маркування та упаковки (не висуваються); 9) Вимоги до транспортування і зберігання (не висуваються); 10) Спеціальні вимоги (не пред’являються).

5.	Вимоги до програмної документації	Програмною документацією щодо розроблюваного програмного продукту вважати: 1) Справжнє технічне завдання на розробку програми (представити як Додаток Б до пояснювальної записки до кваліфікаційної] роботи); 2) Програму і методику випробувань розробленої програми (представити як Додаток В до пояснювальної записки до кваліфікаційної роботи); 3) Опис програмного виробу (представити в Розділі 3 пояснювальної записки до кваліфікаційної роботи).	
6.	Вимоги до техніко-економічних показників	1) Оцінка економічної ефективності – непотрібна. 2) Визначення економічних переваг моделі в порівнянні з вітчизняними та зарубіжними аналогами – непотрібно.	
7.	Стадії і етапи розробки	Дата	Назва етапу

		від 2 грудня 2022 до 16 грудня 2022	Написання технічного завдання.
		від 2 грудня 2022 до 2 січня 2023	Аналіз літератури, щодо існуючих засобів ознайомлення студентів з роботою методів забезпечення цілісності передачі інформації
		від 3 січня 2023 до 30 березня 2023	Дослідження теорії та обґрунтування актуальності
		від 31 березня 2023 до 27 травня 2023	Вибір і аналіз методів методів забезпечення цілісності даних.
		від 28 травня 2023 до 31 серпня 2023	Розробка web додатку для подальшого ознайомлення студентів з роботою методів забезпечення цілісності передачі інформації.
		від 1 вересня 2023 до 21 вересня 2023	Тестування та апробація комп'ютерної моделі.
		від 22 вересня 2023 до 15 жовтня 2023	Розробка інструкцій для користувача.
		від 16 квітня 2023 до 25 листопада 2023	Розробка пояснювальної записки.
8.	Порядок контролю і приймання програмного продукту (моделі)	1) Перевірку ходу розробки додатку виконувати раз в 3 тижні. 2) Випробування програмного продукту провести відповідно до програми та методики випробувань на базі комп'ютерного класу. 3) Захист розробленої моделі провести на засіданні Атестаційної комісії. 4) Пояснювальну записку подати на паперових носіях в 1 примірнику і в електронному вигляді в 1 примірнику на CD-R компакт-диску.	

Виконавець
студент групи КУ-61
Маницький С.С.

___ _____

Замовник
д.т.н. , професор
Мірошник М.А.

_____ _____

Додаток В

ПРОГРАМА І МЕТОДИКА ВИПРОБУВАНЬ

програмного виробу «Система забезпечення цілісності інформації, що передається в мережах передачі даних»

1. Об'єкт випробувань

Об'єктом випробувань є реалізація web додатку для подальшого ознайомлення студентів з роботою методів забезпечення цілісності інформації що передається в мережах передачі даних.

2. Мета випробування

Перевірка відповідності функціональності програмної реалізації заявленим функціональним можливостям в технічному завданні (Додаток Б до пояснювальної записки до кваліфікаційної роботи).

3. Вимоги до програми

Програма повинна:

1) Представляти з себе web-додатку з візуалізацією роботи методів забезпечення інформації
2) Надавати можливість вводити інформацію для обробки шляхом ручного вводу
3) Надавати можливість вводити інформацію через завантаження файлу
4) Надавати можливість налаштовувати "експеримент" шляхом встановлення кількості ітерацій і максимальної кількості помилок
5) Забезпечувати захист від некоректних дій;
6) Забезпечувати вивід результатів обчислень
7) Забезпечити вивід додаткової інформації
8) Вимоги до маркування та упаковки (не висуваються);
9) Вимоги до транспортування і зберігання (не висуваються);
10) Спеціальні вимоги (не пред'являються).

4. Вимоги до програмної документації

Склад програмної документації, що пред'являється на випробуванні:

Програмною документацією щодо розроблюваного програмного продукту вважати:

1) Справжнє технічне завдання на розробку програми (представити як Додаток Б до пояснювальної записки до кваліфікаційної] роботи);
2) Програму і методику випробувань розробленої програми (представити як Додаток В до пояснювальної записки до кваліфікаційної роботи);
3) Опис програмного виробу (представити в Розділі 3 пояснювальної записки до кваліфікаційної роботи)

4. **Засоби випробувань**

Необхідна наявність ПК з сучасним програмним забезпеченням, необхідним для роботи програмного продукту та мобільний пристрій на базі ОС iOS.

5. **Програма і методика випробувань**

1. Перевірка програмної документації

1.1 Перевірка складу програмної документації. Перевірку здійснювати за критерієм наявності, представленої в ТЗ документації.

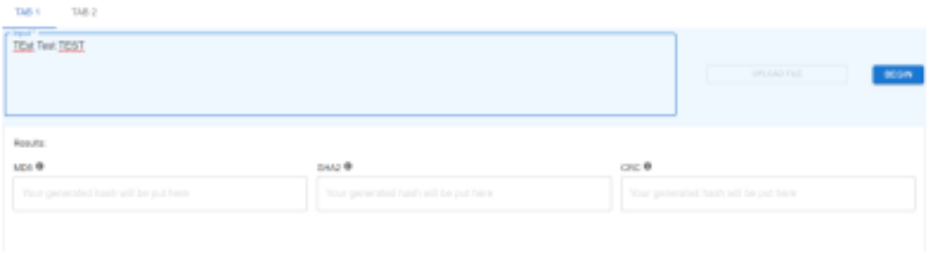
1.2 Перевірка якості програмної документації. Перевірку здійснювати за критерієм відповідності вимогам ГОСТ 19.301-79 ЕСПД. «Програма і методика випробувань».

2. Перевірка працездатності програми

Тест 1

2.1 Запуск програми

2.1.1 Перевірку здійснювати за критерієм відкриття головної вкладки, та можливості введення необхідних значень;



Тест 2

2.2Перевірка виконання програми

2.2.1 Перевірку здійснювати за критерієм можливості введення та вибору необхідних значень

2.2.2 Перевірку здійснювати за критерієм отримання результатів обчислень

Тест 3

2.3Перевірка забезпечення захисту від некоректних дій

2.3.1 Перевірку здійснювати за критерієм неможливості початку обчислень якщо не введені всі необхідні значення

Висновки: випробування вважаються успішно пройденими, якщо були виконані всі перевірки з пунктів 1,2.

Виконавець студент групи КУ-61 Маницький С.С. ___ __

Замовник д.т.н. , професор Мірошник М.А. _

Додаток Г

Лістинг 1. Фрагмент генерації та обробки помилки тексту

```
  const textError = (error: number, prob: number, text: string) =>
{
    switch (error) {
      case 1: {
          const newText = prob >= 50 ? generateFail(text, false,
false) : text

        return newText
      }
      case 2: {
        const newText =
          prob >= 50
            ? prob < 75
              ? generateFail(text, false, false)
              : generateFail(text, true, false)
            : text


        return newText
      }
      case 3: {
        const newText =
          prob >= 50
            ? prob < 67
              ? generateFail(text, false, false)
              : prob < 83
              ? generateFail(text, true, false)
              : generateFail(text, true, true)
            : text


        return newText
      }
      default:
        return text
    }
  }


  const handleClick = useCallback(
    (text: string, int: number, errors: number) => {
      setIsLoading(true)
      clearAll()
```

```
const ifThereLetters = text
  .split('')
  .find((char) => char !== '1' && char !== '0')


const textFormated = ifThereLetters ? convert(text) : text
const initCRC2 = CRC32Function(textFormated)
const initSHA2 = SHA256Function(textFormated)
const initMD5 = MD5Function(textFormated)
const initHamming = Hamming(textFormated)


let successCount = 0
let resendCount = 0
let startTime = performance.now()


for (let index = 0; index < int; index++) {
  let success = false
  while (!success) {
    const probl = getProbability()
    const newText = textError(errors, probl, textFormated)
    setTimeout(() => {}, 2000)
    const crc = CRC32Function(newText)
    if (initCRC2 === crc) {
      success = true
      successCount++
    } else {
      resendCount++
    }
  }
}
let endTime = performance.now()


setCRC32({
  success: successCount,
  resend: resendCount,
  time: `${endTime - startTime} ms`,
})
successCount = 0
resendCount = 0
startTime = performance.now()
```

```
for (let index = 0; index < int; index++) {
  let success = false
  do {
    const prob2 = getProbability()
    const newText = textError(errors, prob2, textFormated)


    const sha = SHA256Function(newText)
    setTimeout(() => {}, 2000)


    if (initSHA2 === sha) {
      success = true
      successCount++
    } else {
      resendCount++
    }
  } while (!success)
}
endTime = performance.now()


setSHA256({
  success: successCount,
  resend: resendCount,
  time: `${endTime - startTime} ms`,
})
successCount = 0
resendCount = 0


startTime = performance.now()
for (let index = 0; index < int; index++) {
  let success = false
  while (!success) {
    const prob2 = getProbability()
    const newText = textError(errors, prob2, textFormated)


    const md = MD5Function(newText)
    setTimeout(() => {}, 2000)


    if (initMD5 === md) {
      success = true
```

```
      successCount++
    } else {
      resendCount++
    }
  }
}
endTime = performance.now()
setMD5({
  success: successCount,
  resend: resendCount,
  time: `${endTime - startTime} ms`,
})


successCount = 0
resendCount = 0


startTime = performance.now()
for (let index = 0; index < int; index++) {
  let success = false
  while (!success) {
    const prob2 = getProbability()
    const ham = Hamming(textFormated, prob2, errors)


    setTimeout(() => {}, 2000)
    if (errors !== 3) {
      if (initHamming.decodedData === ham.decodedData) {
        success = true
        successCount++
      }
      if (errors === 2 && prob2 >= 75) {
        resendCount++
      }
    }
    if (errors === 3) {
      if (prob2 >= 67 && prob2 < 83) {
        resendCount++
      } else if (prob2 >= 83) {
        success = true
      } else {
        if (initHamming.decodedData === ham.decodedData) {
          success = true
          successCount++
        }
```

```
      }
    }
  }
}
endTime = performance.now()
setHamming({
  success: successCount,
  resend: resendCount,
  time: `${endTime - startTime} ms`,
})
successCount = 0
resendCount = 0
startTime = performance.now()
//combined
for (let index = 0; index < int; index++) {
  let success = false
  while (!success) {
    const prob2 = getProbability()


    const ham = Hamming(textFormated, prob2, errors)
    const md = MD5Function(ham.decodedData)
    if (errors !== 3) {
      if (initMD5 === md) {
        success = true
        successCount++
      }
      if (errors === 2 && prob2 >= 75) {
        resendCount++
      }
    }
    if (errors === 3) {
      if (prob2 >= 67 && prob2 < 83) {
        resendCount++
      } else if (prob2 >= 83) {
        resendCount++
      } else {
        if (initMD5 === md) {
          success = true
          successCount++
        }
      }
    }
  }
}
endTime = performance.now()
```

```
    setCombined({
      success: successCount,
      resend: resendCount,
      time: `${endTime - startTime} ms`,
    })
  },
  []
)
```