Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

БАКАЛАВРСЬКИЙ ДИПЛОМНИЙ ПРОЄКТ

на тему:

Захист документації на основі технології хешчейну

Пояснювальна записка

Виконав студент 4-курсу, групи 2КІ-18б

Спеціальності 123 - Комп'ютерна інженерія

Бабейко Б.О.

Керівник к.т.н., доц. каф. ОТ

Семеренко В.П.

"20" 06 2022 р.

Рецензент к.т.н., доц. каф.

Карпінець В.В.

"22" 06 2022 р.

Допущено до захисту

д.т.н., проф. Азаров О.Д.

"23" 06 2022 р.

ВНТУ 2022

Вінницький національний технічний університет

Факультет Інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень бакалавр
Напрям підготовки 6.050102
Спеціальність комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
проф., д.т.н. Азаров
"08" 02 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЕКТ СТУДЕНТУ

Бабейку Богдану Олександровичу

1 Тема роботи: «Захист документації на основі технології хеш-чейну», керівник роботи к.т.н., доц. каф.ОТ Семеренко В.П. ,затверджені наказом вищого навчального закладу від "24" березня 2022 року № 66

2 Строк подання студентом проекту 23 червня 2022.

3 Вихідні дані до проекту: розгляд принципів захисту інформації за допомогою хешування, проведення аналізу математичного апарату лінійних послідовнісних схем, багаторівневий захист інформації за допомогою хеш-чейну.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): основні характеристики хеш-функцій, хеш чейн на основі лінійних послідовнісних схем, програмне забезпечення для автоматизованого шифрування документів. тестування додатку, висновки, література, додатки.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

До переліку обов'язкових матеріалів входять блок схеми алгоритмів функцій: PrintMatrix(), MatrixMultiplication(), MatrixAdd(); також входять технічне завдання та протокол перевірки кваліфікаційної роботи на наявність текстових запозичень.

6 Консультанти розділів проекту приведені в Таблиці 1

Таблиця 1 - Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ 1–4	к. т. н., доцент каф. ОТ Семеренко В.П.	09.02.22 дата	12.05.22 дата
		підпис	підпис

7 Дата видачі завдання 12.03.2022

8 Календарний план виконання БДП приведений в Таблиці 2.

Таблиця 2 - Календарний план

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів проекту (роботи)	Примітка
1	Постановка задачі роботи	01.05.2022	виконано
2	Аналіз стану досліджень на сьогоднішній день	02.05.2022 – 06.05.2022	виконано
3	Аналіз наявних методів шифрування	07.05.2022 – 11.05.2022	виконано
4	Розробка логіки програми.	12.05.2022 – 17.05.2022	виконано
5	Програмна реалізація структури програми для хешування документів	17.05.2022 – 21.05.2022	виконано
6	Підготовка матеріалів додатку для багаторівневого шифрування документів	22.05.2022 – 29.05.2022	виконано
7	Оформлення пояснювальної записки	02.06.2022 – 10.06.2022	виконано
8	Перевірка якості виконання бакалаврської роботи	10.06.2022 – 14.06.2022	виконано

Студент Бабейко Б. О.

Керівник роботи Семеренко В. П.

Анотація

Бабейко Б.О. Захист документації на основі технології хеш-чейну. Бакалаврський дипломний проект зі спеціальності 123 — Комп’ютерна Інженерія, Вінниця: ВНТУ, 2022. Пояснювальна записка містить 48 сторінок, 17 рисунків та 15 посилань.

У цьому бакалаврському дипломному проекті розглядається програмний засіб хешування даних за допомогою математичного апарату ЛПС. Проаналізовано особливості захисту інформації на основі хеш-чейна та наведено декілька аналогів. Відповідно до поставленої задачі була розроблена програма для хешування документів в одному ланцюзі. Це рішення дало змогу отримати зручну та швидкодіючу програму для спрощення шифрування документів.

Ключові слова: хешування, хеш-чейн, ЛПС, захист інформації, криптографія.

Abstract

Babeyko B.O. Document protection based on hashchain technology. Bachelor's degree project in specialty 123 - Computer Engineering, Vinnytsia: VNTU, 2022. Explanatory note contains 48 pages, 17 figures and 15 references.

This bachelor's thesis project considers software hashing of data using the mathematical apparatus of LPS. The peculiarities of information protection based on hash-chain are analyzed and several analogues are given. In accordance with the task, a program for hashing documents in one chain was developed. This solution made it possible to obtain a convenient and fast program to simplify the encryption of documents.

Key words: hashing, hash-chain, LPS, information protection, cryptography.

Зміст

Вступ

В наш час захист інформації набуває все більшої **актуальності**. Від рівня захисту залежить надійність збереження даних, їх цілісність та недоторканність. Одним з методів надійного захисту є однонаправлене шифрування. В цьому випадку отримати вхідні дані маючи лише вихідні стає майже неможливо. Критерієм ефективності являються швидкість і легкість обчислення, однонаправленість та мінімальний шанс на однакові вихідні дані.

В максимальній мірі цим вимогам відповідає хешування. Хеш-функція перетворює вхідні дані в унікальний набір символів, зміна хоча б одного з символів у вхідних даних, приведе повної зміни вихідних даних.

Хеш-чейн зазвичай визначається як повторне застосування криптографічної хеш-функції до даного ресурсу даних. Цей тип хеш-криптографії може бути надзвичайно корисним у деяких конкретних налаштуваннях безпеки. Забезпечуючи послідовний зв’язок, хеш-чейни ускладнюють для хакерів, що перевіряють, захоплення об’єкта даних шляхом застосування одного введення.

Ідея хеш-чейну полягає в тому, що користувач надає індивідуальний вхід під час першої взаємодії або сеансу, а потім додає дані аутентифікації під час наступного сеансу. Протягом набору сеансів ці окремі хеш-входи створюють «хеш-чейн», який автентифікує введення окремого користувача більш глибоким чином.

Наприклад, процеси хеш-чейну можуть бути подібними до підходу до реєстру блокчейну для біткойна та іншої криптовалюти, оскільки блокчейн та інші подібні системи аутентифікують вхід із попередніми списками хеш-ключів. Однак інші види хеш-чейну можуть не мати таких же специфічних функцій і деталей, вбудованих в блокчейн, який стає золотим стандартом прозорості бухгалтерської книги у світі глобальних фінансів.

Об’єктом досліджень є процеси захищених потокових хеш-функцій на основі математичного апарату лінійних автоматів.

Предметом дослідження хеш-чейн для захисту важливої документації.

Метою роботи є розробка програмного забезпечення, що дозволяє економити час при створення чейну хеш-функцій,

Для досягнення поставленої мети необхідно вирішити **задачі**:

- аналіз існуючих підходів та програмних засобів для реалізації хешування;
- дослідження основних сфер застосування хеш-функцій;
- дослідження принципу роботи математичного апарату лінійно послідовнісних схем;
- дослідження алгоритмів створення хеш-чейну;
- аналіз криптостійкості хеш-функцій;
- аналіз методів захисту паролів від зловмисників.

Методи дослідження які були використані: криптографія, теорія завадостійкого кодування, теорія інформації, теорія лінійних автоматів, цифрова схемотехніка, об’єктно-орієнтоване програмування.

Практичне значення одержаних результатів - розробка хеш-чейну для захисту документації, яка використовується в вищих навчальних закладах України.

1 Основні характеристики хеш-функцій

1.1 Основні поняття про хешування

Хеш-функція - це математична функція h(X), що перетворює дані будь якого, зазвичай, великого розміру в дані фіксованого розміру. При цьому зміна хоча б одного символу вхідних даних гарантує , що вихідні дані будуть іншими [1].

Хеш створюється функцією h:

- H = h(X);

- де X – є вхідними даними довільної довжини;

- а H – є хешем фіксованої довжини.

Хеш-функція повинна відповідати таким вимогам:

- швидкість та легкість обчислення;

- мінімізація дублювання вихідних значень (колізії);

- однакова довжина для всіх текстів;

- однонаправленість.

Колізія хеш-функції [2] - це коли при вводі двох різних вхідних даних X і Y хеш буде однаковим (h(X) = h(Y)) (рис.1.1). Колізії існують для більшості хеш-функцій, але для «хороших» хеш-функцій частота їх виникнення близька до теоретичного мінімуму.

Так як хеш-функції часто використовують для шифрування паролів, зловмисники можуть скористатись цим. Вміння знаходити колізії дає доступ до різного роду ресурсів користувача. Від стійкості хеш-функції до знаходження колізій залежить безпека електронного цифрового підпису із використанням даного хеш-алгоритму.

Однонаправлена хеш-функція - це хеш-функція , яка працює лише в одному напрямку: легко вичислити значення хеш-функції по вхідним даним , але важко створити вхідні дані , значення хеш-функцій якого рівне значенню заданої величини.

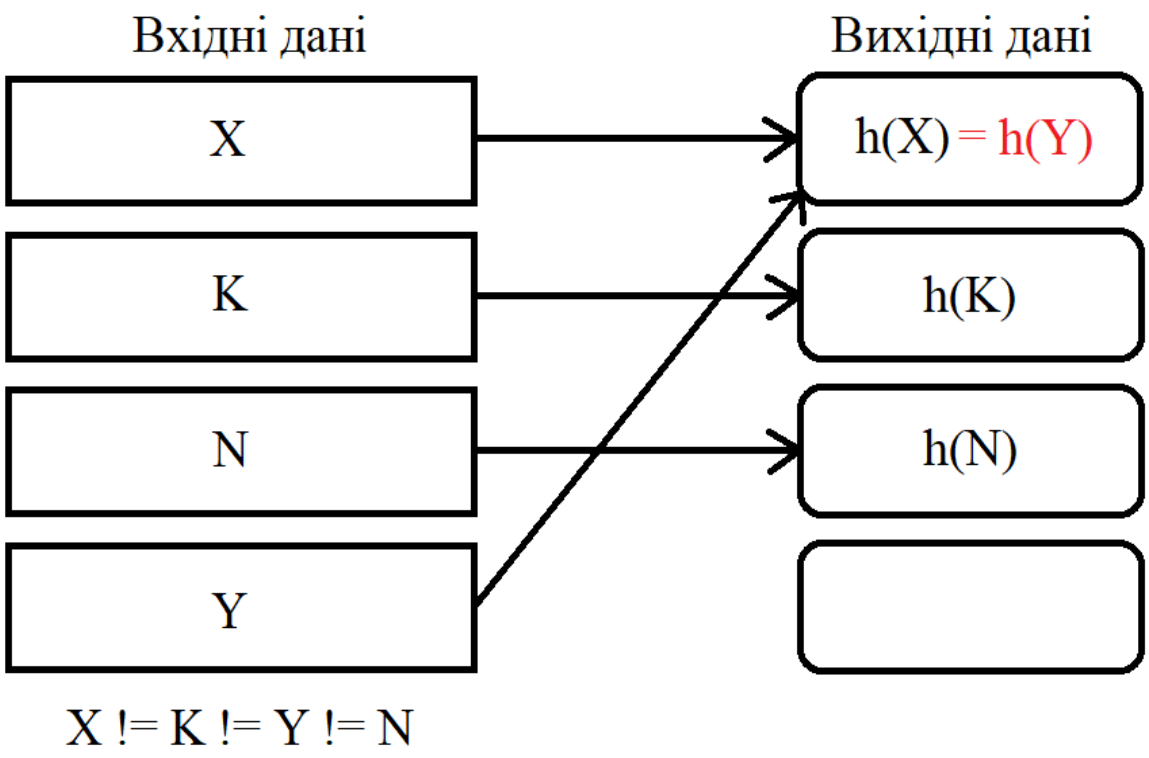


Рисунок 1.1 – Зіткнення хеш-функцій

Безпечність однонаправленої функції полягає саме в її однонаправленості. У вихідних даних немає жодної залежності від вхідних. Зміна одного бігу вхідних даних приведе до кардинальної зміни , як мінімум половини значення хеш-функції, що є чудовим прикладом переваги хешування над звичайним шифруванням, з захистом якого впоратись набагато легше.

Універсальна схема хешування - це рандомізований алгоритм, який вибирає хеш-функцію h серед сімейства таких функцій таким чином, що ймовірність зіткнення будь-яких двох різних ключів дорівнює 1/m, де m - кількість різних хеш-значень. бажаний - незалежно від двох клавіш. Універсальне хешування гарантує (у імовірнісному сенсі), що програма хеш-функції буде вести себе так само, як якщо б вона використовувала випадкову функцію, для будь-якого розподілу вхідних даних. Однак він матиме більше колізій, ніж ідеальне хешування, і може вимагати більше операцій, ніж хеш-функція спеціального призначення.

Часто бажано, щоб вихід хеш-функції мав фіксований розмір .Але, якщо, наприклад, вихідні дані обмежені 32-бітними цілочисельними значеннями, хеш-

значення можна використовувати для індексування в масив. Таке хешування зазвичай використовується для прискорення пошуку даних. Створення вихідних даних фіксованої довжини з вхідних даних змінної довжини можна досягти шляхом розбиття вхідних даних на шматки певного розміру. Хеш-функції, що використовуються для пошуку даних, використовують певний арифметичний вираз, який ітеративно обробляє фрагменти введення (наприклад, символи в рядку), щоб отримати хеш-значення.

У програмах зберігання та пошуку даних використання хеш-функції є компромісом між часом пошуку та простором для зберігання даних. Якби час пошуку був необмеженим, найкращим середовищем був би дуже компактний невпорядкований лінійний список; якби простір для зберігання був необмеженим, випадково доступна структура, яку можна індексувати за ключем-значенням, була б дуже великою, дуже рідкою, але дуже швидкою. Хеш-функції потрібна скінченна кількість часу, щоб зіставити потенційно великий простір ключів у можливу кількість простору пам’яті, доступний для пошуку за обмежений проміжок часу, незалежно від кількості ключів. У більшості додатків хеш-функція повинна обчислюватися з мінімальною затримкою і, як наслідок, з мінімальною кількістю інструкцій.

Складність обчислень залежить від кількості необхідних інструкцій та затримки окремих інструкцій, причому найпростішими є порозрядні методи (згортання), за ними слідують мультиплікаційні методи, а найскладнішими (найповільнішими) є методи на основі ділення.

У деяких програмах вхідні дані можуть містити функції, які не мають значення для цілей порівняння. Наприклад, під час пошуку особистого імені може бути бажано ігнорувати відмінність між великими та малими літерами. Для таких даних необхідно використовувати хеш-функцію, сумісну з використовуваним критерієм еквівалентності даних: тобто будь-які два вхідні дані, які вважаються еквівалентними, повинні давати однакове хеш-значення. Це може бути досягнуто шляхом нормалізації введення перед хешуванням, як, наприклад, у верхньому регістрі всіх літер.

1.2 Особливості використання хеш-функцій

Хеш - функції використовують в наступних сферах:

- в системах документації Cyclic Redundancy Check (CRC);

- в криптографії захищені хеш-функції (з паролем);

- у високозахищених документальних системах (електронний цифровий підпис);

-для заповнення хеш–таблиць.

CRC має дві розшифровки: Cyclic Redundancy Code (CRC) - циклічний надлишковий код; Cyclic Redundancy Check (CRC) - циклічний надлишковий контроль. Cyclic Redundancy Check являється близьким синонімом, який не змінює суть, до терміну «контрольна сума» [3].

1.2.1 CRC в системах документації

Основною перевагою CRC-кодів є швидке обчислення та простота апаратної та програмної реалізації. По суті, це просто число (або може бути сказано), розраховане з вихідного повідомлення, яке передається самим повідомленням (додається до кінця частини інформації) і служить для контролю його безпомилковою передачею.

Це число обчислюється за певними правилами і завжди має наперед визначену кількість цифр. Дуже зручно заздалегідь знати, скільки біт займає номер перевірки, оскільки інакше вся довжина повідомлення буде невідома, разом з CRC, навіть якщо ми точно знаємо довжину інформаційної частини. Крім того, це дозволяє попередньо виділити CRC-реєстр потрібного розміру.

Це число не несе ніякого додаткового інформаційного навантаження, тому є надмірним з точки зору передачі корисної інформації. Однак, його наявність дозволяє діагностувати ряд помилок, якщо вони виникають під час передачі.

І в основному, вся теорія знаходження CRC базується на таких речах:

- будь-яке повідомлення може бути представлене як одне велике двійкове число і значення цифр цього числа можна припустити для кодування коефіцієнтів деякого многочлена;

- також кількість цифр будь-якого бінарного повідомлення дуже конкретні, кінцеві числа.

Обчислення CRC полягає у знаходженні залишків поділу многочлена, складених за певними правилами на основі вихідного повідомлення на многочлен. Многочлени записуються просто з бітовими послідовностями коефіцієнтів. Отриманий залишок, розрядність якого завжди на одиницю менше за степінь породжувального многочлена, це і є CRC.

1.2.2 Використання хеш-функцій в криптографії

Серед безлічі існуючих хеш-функцій прийнято виділяти криптографічно стійкі, які застосовуються в криптографії. Для того, щоб хеш-функція h вважалася криптографічно стійкою, вона повинна задовольняти трьом основним вимогам, на яких засновано більшість застосувань хеш-функцій в криптографії [4]:

- незворотність: для заданого значення хеш-функції H повинно бути обчислювально нездійсненно знайти блок даних X, для якого h(X) = H;
- стійкість до колізій першого роду: для заданого повідомлення X повинно бути обчислювально нездійсненно підібрати інше повідомлення Y, для якого H(X) = H (Y).
- стійкість до колізій другого роду: має бути обчислювально нездійсненно підібрати пару повідомлень (X, X'), що мають однаковий хеш.

Для криптографічних хеш-функцій також важливо, щоб при найменшій зміні аргументу значення функції сильно змінювалося. Зокрема, значення хешу не повинно давати витоку інформації навіть для окремих бітів аргументу. Ця вимога є запорукою криптостійкості алгоритмів хешування, які хешують пароль користувача для отримання даних [5].

У більшості випадків парольні фрази не зберігаються на цільових об'єктах, зберігаються лише їх хеш-значення. Зберігати парольні фрази недоцільно, тому що у разі несанкціонованого доступу до файлу з фразами зловмисник дізнається всі парольні фрази і відразу зможе ними скористатися, а при зберіганні хеш-значень він дізнається лише хеш-значення, які не оборотні у вихідні дані, в даному випадку в

парольну фразу. У ході процедури аутентифікації обчислюється хеш-значення введеної парольної фрази і порівнюється зі збереженим.

Прикладом у разі можуть бути ОС Linux і Microsoft Windows. Вони зберігаються лише хеш-значения парольних фраз облікових записів користувачів.

1.2.3 Електронний цифровий підпис

Аутентифікація захищає двох учасників, які обмінюються повідомленнями, від впливу деякої третьої сторони. Однак проста аутентифікація не захищає учасників один від одного. У ситуації, коли обидві сторони не довіряють одна одній, необхідно щось більше, ніж аутентифікація. Можливим рішенням подібної проблеми є використання цифрового підпису [6].

Цифровий підпис повинен мати наступні властивості:

- можливість перевірити автора, дату і час створення підпису;
- можливість аутентифікувати вміст під час створення підпису.
- підпис має бути перевіряємий третьою стороною для вирішення суперечок.

Таким чином, функція цифрового підпису включає функцію аутентифікації.

На підставі цих властивостей можна сформулювати наступні вимоги до цифрового підпису:

- підпис має бути двійкового зразка, який залежить від підписаного повідолення;
- підпис має використовувати деяку унікальну інформацію відправника для запобігання підробки або відмови;
- створювати цифровий підпис має бути відносно легко;
- повинно бути обчислювально неможливо підробити цифровий підпис як створенням нового повідомлення для існуючого цифрового підпису , так і створенням помилкового цифрового підпису для деякого повідомлення;
- цифровий підпис повиннен бути досить компактними і не займати багато пам'яті.

Сильна хеш-функція , зашифрована закритим ключем відправника, задовольняє перерахованим вимогам.

Існує кілька підходів до використання функції цифрового підпису. Всі вони можуть бути розділені на дві категорії: прямі і арбітражні.

При використанні прямого цифрового підпису взаємодіють тільки самі учасники, тобто відправник і одержувач. Передбачається, що одержувач знає вхідні данні відправника. Цифровий підпис може бути створений шифруванням усього повідомлення або його хешу.

Кожне підписане повідомлення від відправника А до одержувача В насамперед надходить до арбітра С , який перевіряє підпис для даного повідомлення. Після цього повідомлення датується і надсилається до В із зазначенням того, що воно було перевірено арбітром. Присутність С вирішує проблему схем прямого цифрового підпису , при яких А може відмовитися від повідомлення.

Документи, що підписуються, мають різний обсяг, тому часто підпис ставиться не на сам документ, а на його хеш. Обчислення хешу дозволяє виявити найменші зміни у документі під час перевірки підпису. Хешування не входить до складу алгоритму, тому в схемі може бути використана будь-яка надійна хеш-функція.

1.2.4 Використання хеш-функцій з таблицями

Хеш-таблиця – це контейнер, який використовують, якщо хочуть швидко виконувати операції вставки/видалення/знаходження [7].

Хеш-функції використовуються разом з хеш-таблицями для зберігання та отримання елементів даних або записів даних. Хеш-функція перетворює вхідні дані, у хеш-код, який використовується для індексації хеш-таблиці. Коли елемент потрібно додати до таблиці, хеш може індексувати порожній сегмент, у цьому випадку елемент додається до таблиці (рис. 1.2). Якщо хеш-код індексує заповнений сегмент, потрібне якесь вирішення колізій: новий елемент може бути пропущено (не додано до таблиці), або замінити старий елемент, або його можна додати до таблиці в іншому місці за

допомогою визначену процедуру. Ця процедура залежить від структури хеш-таблиці: у ланцюговому хешуванні кожен сегмент є головою зв'язаного списку або чейна, а елементи, які стикаються в сегменті, додаються до ланцюга. Чейни можна зберігати у випадковому порядку та шукати лінійно, або в послідовному порядку, або у вигляді списку, що самоупорядковується за частотою, щоб прискорити доступ. При хешуванні відкритої адреси таблиця перевіряється, починаючи із зайнятого сегменту, визначеним способом, як правило, шляхом лінійного, квадратичного або подвійного хешування до тих пір, поки не знайдеться відкритий сегмент або не буде перевірена вся таблиця (переповнення). Пошук елемента виконується за такою ж процедурою, доки елемент не буде знайдено, відкрите місце або не буде здійснено пошук у всій таблиці (елементу немає в таблиці) [8].

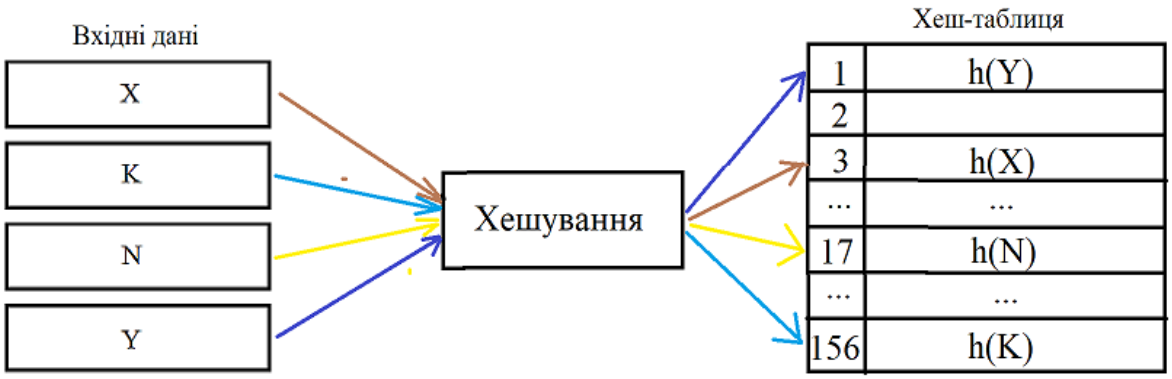


Рисунок 1.2 – Заповнення хеш-таблиці даними

Хороша хеш-функція повинна якомога рівномірніше відображати очікувані вхідні дані в діапазоні вихідних даних. Тобто кожне хеш-значення у вихідному діапазоні має бути згенеровано з приблизно однаковою ймовірністю. Причина цієї останньої вимоги полягає в тому, що вартість методів, заснованих на хешуванні, різко зростає у міру збільшення кількості колізій. Якщо деякі хеш-значення з більшою ймовірністю зустрічаються, ніж інші, більшій частині операцій пошуку доведеться шукати в більшому наборі суперечливих записів таблиці.

1.3 Алгоритми хеш-функцій

MD5 – 128-бітний алгоритм хешування, доволі застарілий, але використовувався для перевірки цілісності інформації і зберігання хешу паролів. Алгоритм отримує на вході вхідні дані довільної довжини і створює в якості виходу вихідні дані довжиною 128 біт.

На кожному з чотирьох циклів алгоритму використовується одна з чотирьох елементарних логічних функцій . Кожна елементарна функція отримує три 32-бітних слова на вході і на виході створює одне 32-бітове слово. Кожна функція є безліччю побітових логічних операцій, тобто n-ий біт виходу є функцією від n-ого біта трьох входів.

SHA-1 або Secure Hash Algorithm 1 - це криптографічна хеш-функція, яка приймає вхідні дані і виробляє 160-бітне (20-байтне) хеш-значення.

SHA-1 зараз вважається небезпечним з 2005 року. Як і MD5, заснований на алгоритмі MD4.

SHA-1 і SHA-2 - це дві різні версії алгоритму. Вони відрізняються як конструкцією (як отриманий хеш створюється з вихідних даних), так і довжиною в бітах підпису. SHA-2 є наступником SHA-1, оскільки це загальне покращення.

Насамперед, люди зосереджуються на довжині біта як важливої відмінності. SHA-1 - це 160-бітний хеш. SHA-2 насправді групою хеш-функцій і буває різної довжини, найпопулярнішим є 256-бітний.

При використанні алгоритму хешування SHA-1 ми будемо отримувати унікальні вихідні дані для кожного варіанту однакової довжини [11]. Тобто при хешуванні «ВНТУ», ми отримаємо: 6d7b01ae5fa9bd057b9c41134b50177a7194dddd, а коли вхідними даними буде виступати «Вінницький Національний Технічний Університет», результатом буде: 93275ffd5fe0429e4b2d9b030542b 73c476718da.

В даному випадку кількість символів у вхідних даних різна, але вихідні дані мають однакову довжину , а саме набір із 40 букв та цифр.

Ось так це виглядає, якщо зобразити у вигляді схеми:

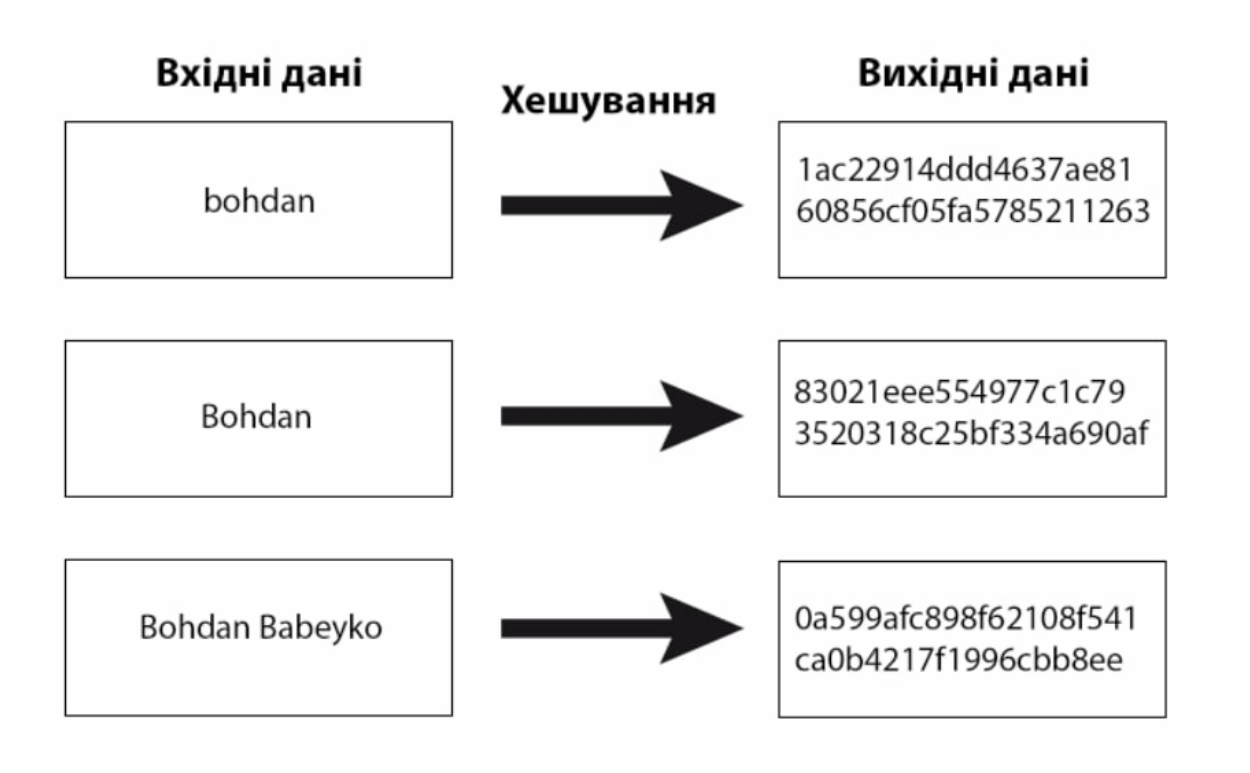


Рисунок 1.3 - Приклад хешування даних за допомогою алгоритму SHA-2

Як легко помітити, зміна одного біта привело до радикальної зміни усіх вихідних даних.

До SHA-2 входять: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256. У кожному випадку з SHA-N, де N – це альтернативна довжина бітів SHA-2.

RIPEMD (RACE Integrity Primitives Evaluation Message Digest) - це група хеш-функцій, розроблена Гансом Доббертіном, Антоном Босселаерсом і Бартом Пренілом у 1992 році. Ідея розробки RIPEMD заснована на MD4, який сам по собі є слабкою хеш-функцією. Він розроблений для роботи з 32-розрядними процесорами.

Перший RIPEMD не вважався хорошою хеш-функцією через деякі недоліки дизайну, які призводять до деяких серйозних проблем із безпекою, одна з яких полягає у розмірі виводу, який становить 128 біт, який занадто малий і його легко зламати. У наступній версії RIPEMD-128 недолік дизайну усунено, але вихідний сигнал все ще 128 біт, що робить його менш безпечним.

RIPEMD-160 - це наступна версія, яка збільшує вихідну довжину до 160 біт і підвищує рівень безпеки хеш-функції. Ця функція призначена для роботи як заміна 128-бітним хеш-функціям MD4, MD5 і RIPEMD-128.

RIPEMD-256 і RIPEMD-320 є розширенням RIPEMD-128, які забезпечують таку ж безпеку, як RIPEMD-160 і RIPEMD-128, що розроблено для додатків, які віддають перевагу більшому хеш-значенню, а не більшому рівню безпеки.

1.4 Захист від дешифрування паролів

Що станеться, якщо група зловмисників вирахує хеш-функції для десятків мільйонів популярних паролів? У такому випадку для злому пароля буде достатньо підрахувати хеш, який досить просто знайти в таблиці. Існує і просунута версія атаки під назвою Rainbow Tables [12].

Rainbow Tables – це реалізація методу time-memory trade-off, за якого немає необхідності зберігати всі можливі хеші. Натомість вони розбиваються на взаємопов'язані ланцюги, від яких зберігаються лише початок і кінець. "Веселкові таблиці" вимагають зусиль з перебору, але перебирати потрібно набагато менше повного діапазону, причому параметри атаки можна підібрати або вирахувати заздалегідь. Обчислення подібних таблиць займає у хакерів тривалий час, після чого самі атаки здійснюються дуже швидко (рис. 1.3).

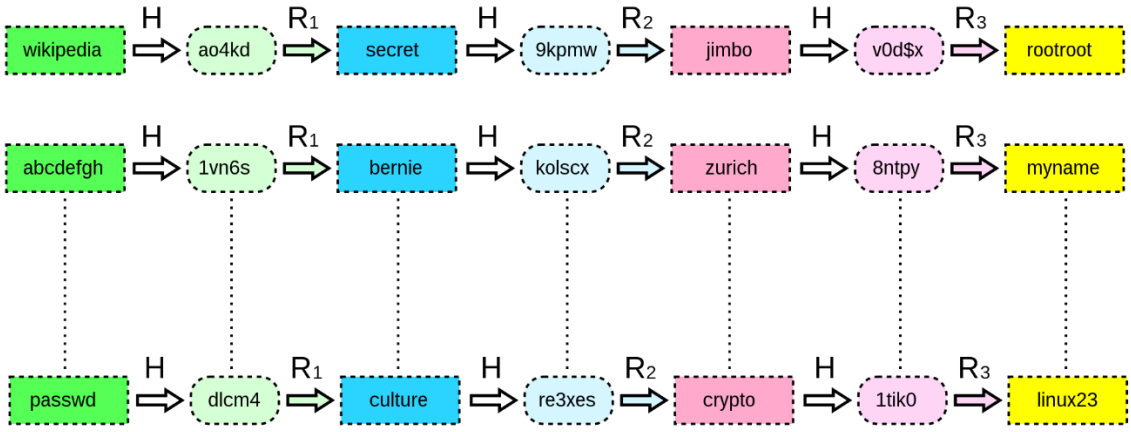


Рисунок 1.3 – Принцип роботи веселкових таблиць

Для протидії подібним атакам було розроблено метод, що застосовує до хешу так звану криптографічну сіль. Тепер хеш обчислюється не від самого пароля, а від комбінації пароля та доданого до нього випадкового рядка (солі). Значення солі (повторимося - це рядок, що складається із випадкових даних) зберігається поруч із хешем, без неї відновити правильний пароль не вдасться. У поодиноких випадках на додаток до солі використовується ще один рядок - Pepper («перець»), значення якого зберігається на окремому фізичному сервері для забезпечення додаткового рівня безпеки.

Для забезпечення адекватного захисту недостатньо використовувати стійкий алгоритм шифрування та криптографічно безпечну хеш-функцію; необхідно правильно реалізувати всю систему захисту. Так, навіть найбезпечніший хеш можна атакувати за допомогою таблиць, у яких містяться як самі паролі, так і результат їх обігу за допомогою популярних хеш-функцій.

Хешування може виконуватися як з використанням деякого секретного ключа, так і без нього. Таке криптографічне контрольне підсумовування широко використовується в різних методах захисту інформації, зокрема, для підтвердження цілісності даних. (рис 1.4).

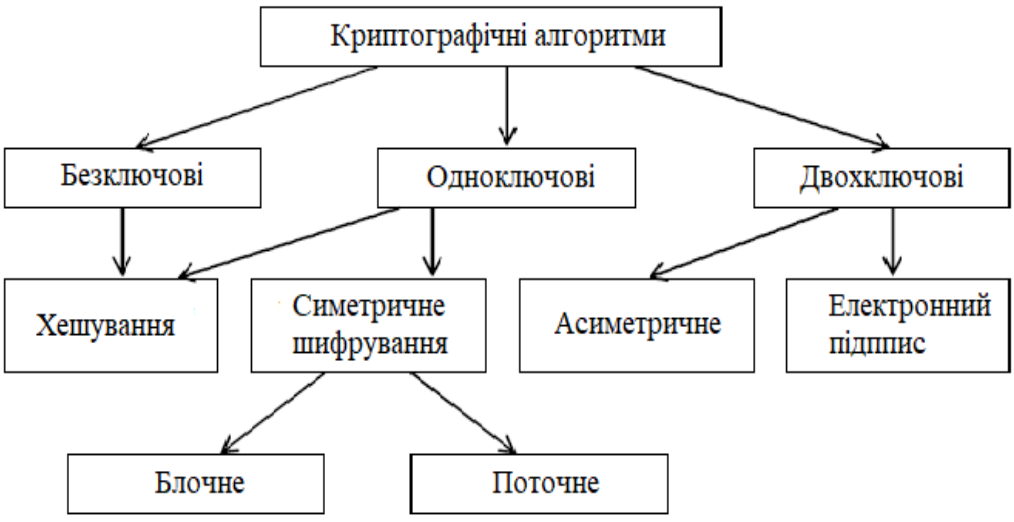


Рисунок 1.4 - Алгоритми шифрування даних

Криптографічний захист може бути організований як програмно, так і з використанням апаратнопрограмних і апаратних засобів. Для забезпечення належного рівня захищеності інформації потрібна криптографічна система (криптосистема) - сукупність засобів криптографічного захисту, необхідної ключової, нормативної, експлуатаційної, а також іншої документації (зокрема й такої, що визначає заходи безпеки). Уразливість криптографічних систем пов’язана з тим, що вони базуються на задачах, які визнані умовно нерозв’язуваними – для жодної з них не знайдено ефективного розв’язання, але й не доведено, що воно не існує. Від добору ключа методом перебирання криптосистема захищена поки що недостатнім рівнем швидкодії комп’ютерів. А численність типів можливих атак на криптографічні системи («на спосіб реалізації», «на паролі», «на користувача», «на моделі довіри» і т. ін.) підтверджує той факт, що захист є надійним і безпечним доти, доки не розпочинаються спроби його зламування.

2 Розробка хешчейну на основі лінійних посідовнісних схем

2.1 Створення хеш функцій на основі математичного апарату лінійно послідовних схем

Циклічний надлишковий контроль (CRC – Cyclic Redundancy Check) широко використовується у різних системах передачі даних.

Суть методу полягає в перевірці відсутності помилок в інформаційній послідовності I довільної довжини h(h>>r) за допомогою r-розрядної контрольної суми ∑, отриманої шляхом ділення послідовності I на заданий примітивний поліном g(x) над полем Галуа GF [13].

Така контрольна сума ∑ відповідає усім основним вимогам, які обов'язково відповідають хеш-функції в криптографії: однакова довжина для всіх можливих послідовностей, однонаправленість, стійкість до колізій.

Сигнатурний аналіз обчислює сигнатури шляхом ділення тестової послідовності сигналів на заданий примітивний поліном g(x).

Виходячи з цього, хеш-функція, сигнатура та розглянуті вище CRC – це результат однієї і тієї ж математичної операції, відрізняється лише назва.

Але отримуваний практичний результат на завжди задовольняє кожну з дисциплін.

Наприклад, CRC дозволяє виявити незалежні помилки непарної кратності або пакети помилок довжиною не більше ніж r. Для забезпечення можливості виправлення помилок потрібно додати певні обмеження. Якщо для певного додатного числа $m \geq 4$ довжина n послідовності I не перевищує 2^m - 1, довжина контрольної суми ∑ дорівнює $m + 1$, а поліном g(x) має вигляд:

$$g(x) = (1 + x)p(x),$$

де $p(x)$ - примітивний поліном степені m.

Тоді в послідовності I можна виправити всі поодинокі помилки і всі суміжні пакети помилок довжини.

Розглянутий раніше спосіб формування хеш-функції є вразливим з боку захисту інформації. Для аналізу криптостійкості хеш-функції підходить математичний апарат лінійних послідовних схем (ЛПС), який використовується в завадостійкому кодуванні. В цьому випадку легко довести, що при додаванні до вхідного повідомлення двох r-розрядних секретних ключових послідовностей символів, захист потокових хеш-функцій від криптоатак стає в рази надійнішим.

В даний час стає актуальним захист інформації при передачі її по комп'ютерних мережах та каналів передачі. В цьому випадку необхідно забезпечувати секретність переданої інформації.

Один із способів шифрування повідомлень представлених у виді цифрових груп з однаковим числом знаків, полягає у наступному: перед передачею повідомлення до кожної групи додається «ключове слово». Відновити передане повідомлення є можливим, лише знаючи ключове слово, віднімаючи його від кожної групи. Цей метод шифрування і дешифрування легко реалізується за допомогою лінійної послідовністної схеми з початковим нульовим станом, яка періодично додає або віднімає ключове слово в повідомленні.

Більшість відомих схем вхідних шифрів базується на використанні регістрів зсуву з лінійним зворотнім зв’язком (РЗЛЗЗ). Дуже проста структура яку можна використовувати для їх аналізу алгебраїчну теорію многочленів над полем Галуа. Але РЗЛЗЗ є також окремим випадком кінцевого автомату лінійного типу який ще називають ЛПС. ЛПС може бути описана за допомогою функції переходів (1) та функцій виходів (2) [14].

$$S(t+1) = A \times S(t) + B \times U(t), \quad (1)$$

$$S(t) = A \times S(t) + B \times Y(t), \quad (2)$$

де S(t), U(t) - це вектори станів;

A, B - матриці; t - час.

Розглянемо приклад автоматного методу розрахунку CRC

$$A = \begin{vmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}; \quad B = \begin{vmatrix} 0 \\ 1 \\ 0 \\ 0 \end{vmatrix};$$

У нас є два породжувальних полінома A та B розмірністю 4 x 4 та 4 x 1 відповідно.

L = 01110001

Послідовність n-ної довжини L

$$S(0) = \begin{vmatrix} 0 \\ 0 \\ 0 \\ 0 \end{vmatrix};$$

S(0) - початковий стан автомата;

$$S(1) = A \times S(0) + B \times U(0) = \begin{vmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} 0 \\ 0 \\ 0 \\ 0 \end{vmatrix} + \begin{vmatrix} 0 \\ 1 \\ 0 \\ 0 \end{vmatrix} \times |1| = \begin{vmatrix} 0 \\ 1 \\ 0 \\ 0 \end{vmatrix};$$

Для того, щоб дізнатись S(1) - перший стан автомата, нам потрібно матрицю A перемножити з матрицею попереднього стану автомата , в даному випадку з S(0).

Наступним кроком в обчисленні буде множення матриці B та n-ного символу L- послідовності. Значення U ми отримуємо в зворотному порядку, тобто в даному випадку $U_0 = 1$, $U_1 = 0$, $U_2 = 0$, $U_3 = 0$, $U_4 = 1$, $U_5 = 1$, $U_6 = 1$, $U_7 = 0$.

Після множення ми додаємо дві матриці які отримали, в результаті ми отримаємо матрицю першого стану автомата.

Відповідно до цього алгоритму робимо і наступні кроки. Цикл виконується відповідну до довжини L кількість разів.

$$S(2) = A \times S(1) + B \times U(1) = \begin{vmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} 0 \\ 1 \\ 0 \\ 0 \end{vmatrix} + \begin{vmatrix} 0 \\ 1 \\ 0 \\ 0 \end{vmatrix} \times |0| = \begin{vmatrix} 1 \\ 0 \\ 0 \\ 0 \end{vmatrix};$$

$$S(3) = A \times S(2) + B \times U(2) = \begin{vmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} 1 \\ 0 \\ 0 \\ 0 \end{vmatrix} + \begin{vmatrix} 0 \\ 1 \\ 0 \\ 0 \end{vmatrix} \times |0| = \begin{vmatrix} 0 \\ 1 \\ 0 \\ 0 \end{vmatrix};$$

$$S(4) = A \times S(3) + B \times U(3) = \begin{vmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} 0 \\ 1 \\ 0 \\ 0 \end{vmatrix} + \begin{vmatrix} 0 \\ 1 \\ 0 \\ 0 \end{vmatrix} \times |0| = \begin{vmatrix} 1 \\ 0 \\ 0 \\ 0 \end{vmatrix};$$

$$S(5) = A \times S(4) + B \times U(4) = \begin{vmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} 1 \\ 0 \\ 0 \\ 0 \end{vmatrix} + \begin{vmatrix} 0 \\ 1 \\ 0 \\ 0 \end{vmatrix} \times |1| = \begin{vmatrix} 0 \\ 1 \\ 0 \\ 0 \end{vmatrix};$$

$$S(6) = A \times S(5) + B \times U(5) = \begin{vmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} 0 \\ 1 \\ 0 \\ 0 \end{vmatrix} + \begin{vmatrix} 0 \\ 1 \\ 0 \\ 0 \end{vmatrix} \times |1| = \begin{vmatrix} 1 \\ 1 \\ 0 \\ 0 \end{vmatrix};$$

$$S(7) = A \times S(6) + B \times U(6) = \begin{vmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} 1 \\ 1 \\ 0 \\ 0 \end{vmatrix} + \begin{vmatrix} 0 \\ 1 \\ 0 \\ 0 \end{vmatrix} \times |1| = \begin{vmatrix} 1 \\ 1 \\ 0 \\ 0 \end{vmatrix};$$

$$S(8) = A \times S(7) + B \times U(7) = \begin{vmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} 1 \\ 1 \\ 0 \\ 0 \end{vmatrix} + \begin{vmatrix} 0 \\ 1 \\ 0 \\ 0 \end{vmatrix} \times |0| = \begin{vmatrix} 1 \\ 1 \\ 0 \\ 0 \end{vmatrix};$$

В результаті повного циклу ми отримуємо останній біт кодової послідовності, він і буде хешем.

Завдяки апарату ЛПМ доволі просто реалізувати процедуру хешування. Тоді вхідну послідовність A довільної довжини m можна називати як вхідна послідовність символів, під дією якої ЛПМ з початкового стану S(0) перейде в S(m).

Важливо, що є багато спільного між завданнями технічної діагностики та криптографії. У технічній діагностиці максимальної ступеня стиснення контрольних даних досягається при сигнатурному аналізі, а у криптографії за допомогою хеш-функцій. Тому для вирішення однотипних задач можна застосувати той самий

математичний апарат - теорію ЛПС - і використати деякі отримані теоретичні результати сигнатурного аналізу на дослідження потокових хеш-функцій.

Для 2^m - 1 різних ненульових входів послідовностей довжини m n-мірна ЛПС може сформувати 2^n - 1 різних хеш-функцій довжини n. Таким чином, кожній хеш-функції відповідає 2^{m-n} - 1 однакових вхідних послідовностей. Це означає, що у випадку рівноймовірності вхідних послідовностей , вірогідність отримати при обрахунку колізію буде дорівнювати: $p_0 \approx 2^{-n}$.

Криптостійкі псевдовипадкові генератори існують лише тоді і тільки тоді, коли існують однонаправлені функції. Також, n-мірна ЛПС, яка реалізує формулу (1) буде представляти собою генератор псевдовипадкових чисел, якщо відповідний їй породжуваний поліном у характеристичній матриці A є примітивним. Така ЛПС буде генерувати послідовність символів періоду 2^n - 1 M-послідовності, яка при великих n фактично не відрізняється від випадкової послідовності. Відповідно цьому, функція являється односторонньою для примітивного поліному.

2.2 Аналіз криптостійкості хеш-функції

Існують методи захисту від зловмисників, вони дозволяють обійти повний перебір варіантів. Така можливість існує та базується на властивості ЛПС. Наприклад характеристична матриця A використовує примітивний поліном в n-мірній ЛПС, вона буде n-керованою, тобто для будь-яких внутрішніх станів S(i) та S(j) існує вхідна послідовність довжини не більш ніж n, яка дозволяє переводити ЛПС зі стану S(i) у стан S(j). Аналітична властивість n-керованості ЛПС показується наступним чином:

$$S(j) - A^n \times S(i) = \| A^{n-1}B, A^{n-2}B, AB, B \| \times U$$

Припустимо, що зловмиснику, окрім значення самої хеш-функції відома також структура ЛПС, тобто характеристичні матриці A та B. Завданням зловмисника є отримання для свого повідомлення (послідовності W символів довжини k) такого ж

значення S(m) хеш-функції, що і для вхідного повідомлення (послідовності X символів довжини m). З цією метою зловмисник спочатку подає послідовність W на вхід ЛПС, який знаходиться в початковому стані S(0), та за допомогою формули (1) визначає код останнього стану S(k) ЛПС. Далі за формулою (2) від визначає вхідну послідовність U довжини не більшої ніж n для переводу ЛПС зі стану S(k) у стан S(m). В результаті сумарна послідовність W+U довжини k+n генерує те ж саме значення S(m) хеш-функції, що і послідовність X довжини m.

Простим та надійним способом захисту від такої криптоатаки є додавання до вихідного повідомлення двох n-розрядних секретних ключових послідовностей символів для захисту хеш-функції. З вихідного нульового стану S(0) ЛПС за допомогою встановленої послідовності V спочатку переводиться у стан S(n), а потім під впливом повідомлення X довжини m – у стан S(m+n) і, нарешті, під впливом ключової послідовності E довжини n – у стан S(m+2n), яке і буде вже захищеною хеш-функцією. В якості послідовності V можна використовувати послідовність E, тобто зберегти в секреті лише одне n-розрядне значення ключа.

Оскільки при обчисленні хеш-функції використовуються симетричні ключі, які відомі тільки відправнику та одержувачу, тому така хеш-функція виконує також завдання аутентифікації повідомлення. При цьому швидкість обчислення потокової хеш-функції зростає більш ніж у 10 разів, порівняно з отриманням хеш-функції типу MD5. Ще більшої продуктивності можна досягти при використанні багатоканальної ЛПС.

2.3 Принцип роботи хеш-чейну

Хеш-чейн - це послідовність значень, отриманих за допомогою послідовних застосувань криптографічної хеш-функції до початкового входу. Завдяки властивостям хеш-функції відносно легко обчислити послідовні значення в ланцюгу, але з урахуванням конкретного значення неможливо визначити попереднє значення.

Головною особливістю таких баз даних являється організаця у вигляді ланцюга блоків. Кожен блок складається з інформації яку потрібно зашифрувати та хеша.

Хеш кожного блоку обраховується з урахуванням цілого попереднього блоку (рис.2.1). Будь-які, навіть мінімальні зміни в блоці поведуть за собою зміну усіх значень хешу, в результаті чого буде порушена цілісність хеш-чейну [15].

Хеш-чейн з n значень часто позначається $H^n(x)$ де значення і в ланцюзі буде обчислено як $x_i = H(x_i\text{-}1)$. Звідси випливає, що якщо початковий пароль був x, тоді:

$$H^2(x) = H(x), H(H(x)),$$

$$H^3(x) = H(x), H\big(H(x)\big), H\Big(H\big(H(x)\big)\Big),$$

і так далі. Якщо дано значення у ланцюгу x_i, не можливо визначити попереднє значення x_{i-1} через властивості хеш функції. Але, якщо x_{i-1} згодом було задано, то можна обчислити для перевірки $H(x_{i-1}) = x_i$ і впевнитись, що особа яка надала значення, знає хеш-чейн, і таким чином, початкове значення або пароль.

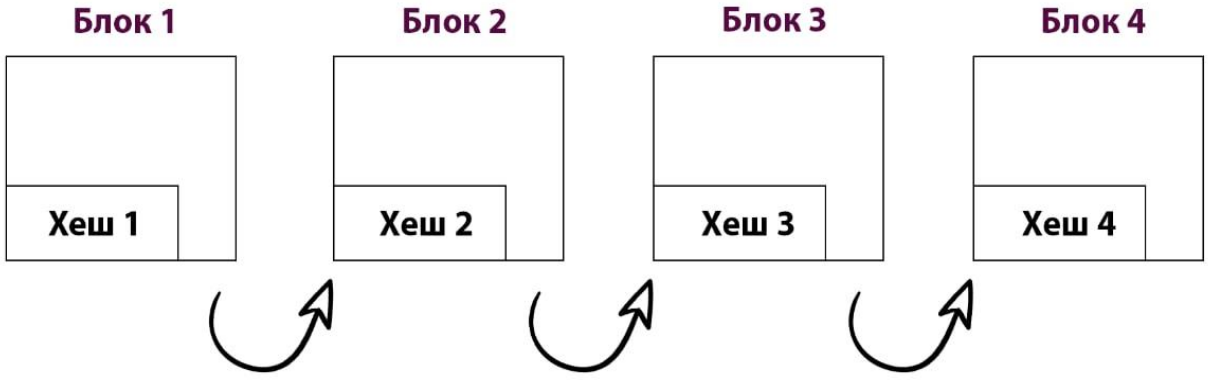


Рисунок 2.1 - Схема роботи послідовного хешчейну

Наприклад, користувач вказує у для першої авторизації. Для аутентифікації користувача система просто обчислить H(y), щоб дізнатись, чи дорівнює результат x_n. Якщо дорівнює, це означає, що користувач правильно вказав первинне значення хеш-чейна (тобто $y = x_{n-1}$), він пройде аутентифікацію. У цьому прикладі , якщо зловмисник повинен був перехопити у або скомпрометувати систему, щоб виявити

x_n, зловмисник не зміг би визначити наступний дійсний пароль (попереднє значення в хеш-чейну) через однонаправлену властивість хеш-функцій. Стійкість до колізій також є важливою для практичного застосування хеш-чейнів. У разі аутентифікації зловмисник, який знав останній використаний пароль x_i , але не знав x_{i-1} або x, міг вказати інший рядок z таким чином, що $H(z) = H(x_{i-1}) = x_i$ за відсутності стійкості до колізій.

Після того як почали використовувати хеш-чейни для аутентифікації комп'ютерної системи, вони швидко стали важливим криптографічним примітивом. Згодом хеш-чейни були застосовані в різних контекстах, наприклад, для безпечних протоколів маршрутизації, мікроплатежів, відкликання сертифікатів у криптосистемах з відкритим ключем (інфраструктура відкритих ключів), прямої безпеки (Perfect Forward Security) та захисту конфіденційності за допомогою радіочастотної ідентифікації (RFID).) системи (RFID Security), а також ефективна аутентифікація з роздільною здатністю адрес, журналами системних подій, багатоадресним мережевим трафіком і бездротовими сенсорними мережами.

Існує безліч можливих варіантів реалізації хеш-чейну, але не багато із них зарекомендували себе так як паралельний метод хешування (рис 2.2). В ньому на вхід подають пари даних, з яких береться хеш, і об'єднується за допомогою конкатенації, міняючи назву на двузначне число, беручи значення з першого та останнього числа з номерів породжувальних хешів.

Наприклад ми маємо x1 та x2, при додаванні їх хешів, ми отримаємо x(1|2)=x12, якщо ж додати x12 до x34, ми бачимо 4 числа x(12|34), в такому випадку першим знаком буде 1, а останнім 4, в результаті отримуємо x14.

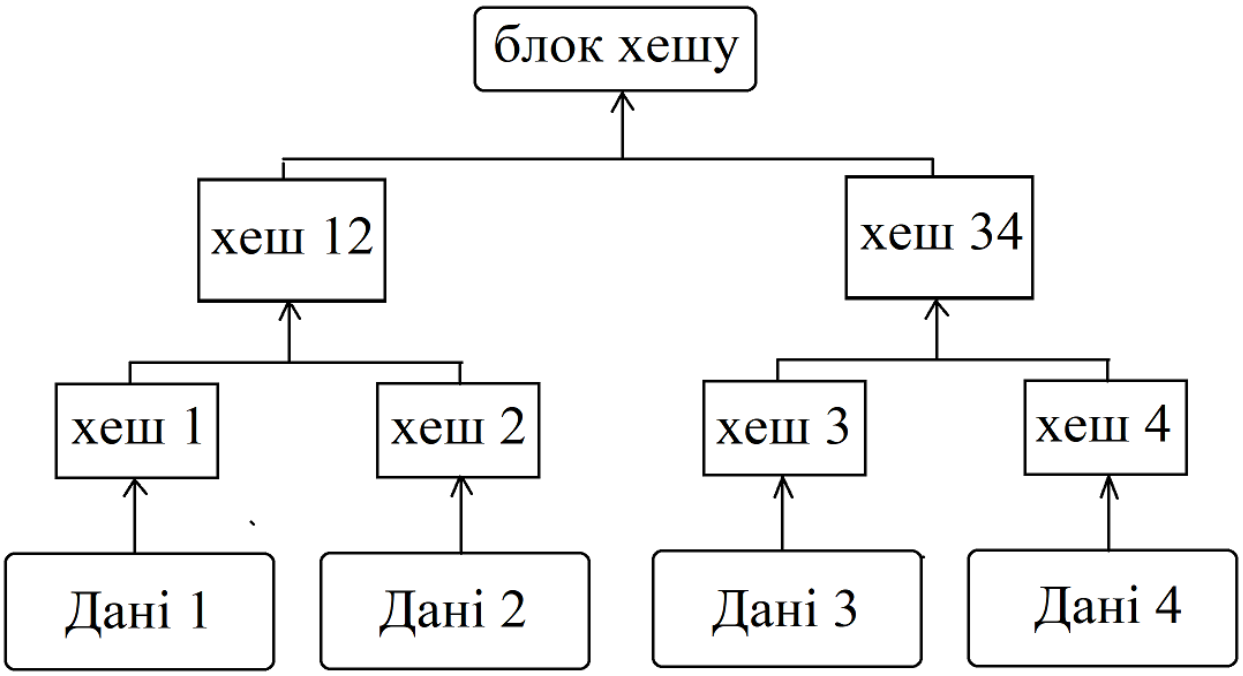


Рисунок 2.2 – Схема роботи паралельного хеш-чейну

3 Розробка програмного забезпечення для автоматизованого шифрування

3.1 Вибір середовища розробки

Для виконання наших задач розглянемо декілька IDE:

- Rider;
-SonarQube;
- VS Code;
- Microsoft Visual Studio

3.1.1 IDE для .NET Rider

Rider допомагає розробляти програми .NET, ASP.NET, .NET Core, Xamarin та Unity на Windows, MacOS або Linux. Він забезпечує широкі можливості редагування та аналізу коду для мов, що використовуються в .NET-розробці, таких як C#, VB.NET, F#, підтримує синтаксис ASP.NET Razor, JavaScript, TypeScript, XAML, XML, HTML, CSS, SCSS, JSON та SQL.

Rider використовує інтерфейс (рис 3.1) та безліч функцій платформи IntelliJ, яка лежить в основі IntelliJ IDEA , WebStorm та інших IDE, розроблених у JetBrains. Платформа IntelliJ знайома мільйонам розробників. Саме вона забезпечує основну функціональність Rider, включаючи інтеграцію із системами контролю версій та підтримку баз даних.

Навігація та пошук, рефакторинги, інспекції коду, швидкі виправлення та багато інших інтелектуальних функцій Rider запозичені з ReSharper.

Середовище розробки Rider не сковане рамками 32-розрядних процесів, тому воно глибоко аналізує код. Rider відкриває більшість рішень майже без затримки. Внесення зовнішніх змін у рішення, перемикання гілок Git – такі завдання не позначаться на швидкості роботи Rider.

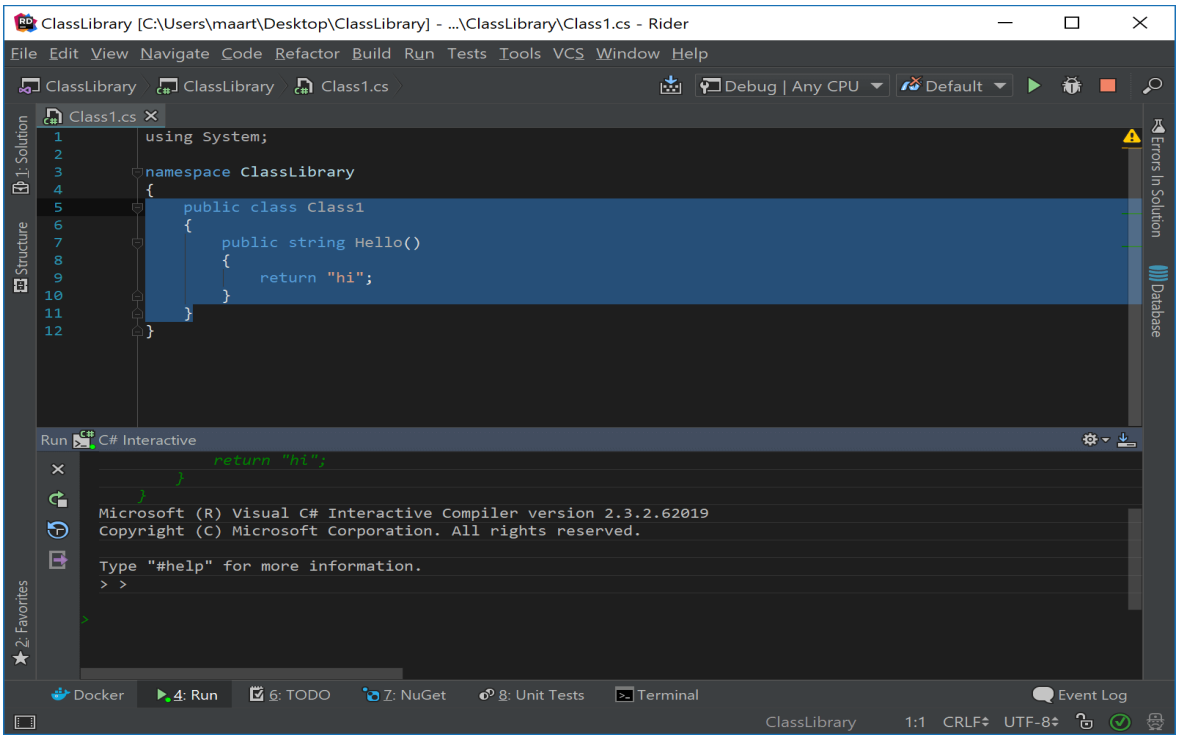


Рисунок 3.1 – Інтерфейс IDE Rider

Інтелектуальний редактор коду надає кілька видів автодоповнення, шаблони для написання типових конструкцій та постфіксні шаблони, посилання на контролери та дії в ASP.NET MVC, що дозволяє редагувати код у кількох місцях одночасно та швидко переміщатися по ієрархії успадкування за допомогою іконок на полях. Rider імпортує відсутні простору імен, вставляє парні дужки, підсвічує межі блоків коду. Для переходу до рефакторингів, генерації коду, команд навігації та контекстних дій, потрібно натиснути пару клавіш.

Rider включає повнофункціональний відладчик для додатків на .NET Framework, Mono та .NET Core. Ви можете створювати різні конфігурації налагодження, встановлювати точки зупинки та задавати умови їх спрацьовування. Відладчик підтримує покрокове виконання із заходом усередину процедури, обходом коду та виходом із нього. Він також дозволяє приєднуватися до зовнішніх процесів, запускати програму від поточного рядка до рядка з курсором, відстежувати значення змінних, обчислювати вирази і досліджувати потоки.

3.1.2 VS Code

Visual Studio Code — це легкий, але потужний редактор вихідного коду, який працює на вашому робочому столі та доступний для Windows, macOS та Linux. Він має вбудовану підтримку JavaScript, TypeScript і Node.js і має багату екосистему розширень для інших мов і середовищ виконання (наприклад, C++, C#, Java, Python, PHP, Go, .NET).

VS Code має простий та інтуїтивно зрозумілий макет (рис 3.2), який максимально збільшує простір, наданий редактору, залишаючи достатньо місця для перегляду та доступу до повного контексту вашої папки або проекту. Інтерфейс користувача розділений на п’ять областей:

- редактор це основна область редагування файлів. Ви можете відкривати скільки завгодно редакторів поруч по вертикалі та горизонталі;

- бічна панель містить різні види, як-от Explorer, щоб допомогти вам під час роботи над проектом;

- рядок стану представляє собою інформацію про відкритий проект і файли, які ви редагуєте;

- панель активності розташована в дальній лівій частині, вона дає змогу перемикатися між представленнями та надає додаткові індикатори, що залежать від контексту, як-от кількість вихідних змін, коли увімкнено Git;

- можливість відображати різні панелі під областю редактора для виведення або налагоджувальної інформації, помилок і попереджень або вбудованого терміналу. Панель також можна перемістити вправо, щоб збільшити простір по вертикалі.

Кожного разу, коли ви запускаєте VS Code, він відкривається в тому ж стані, в якому був, коли ви востаннє його закривали. Папка, макет і відкриті файли зберігаються.

Відкриті файли в кожному редакторі відображаються із заголовками вкладок (Tabs) у верхній частині області редактора.

Підтримка C# у Visual Studio Code оптимізована для міжплатформної розробки .NET. VS Code підтримує налагодження програм C#, що працюють на .NET або Mono.

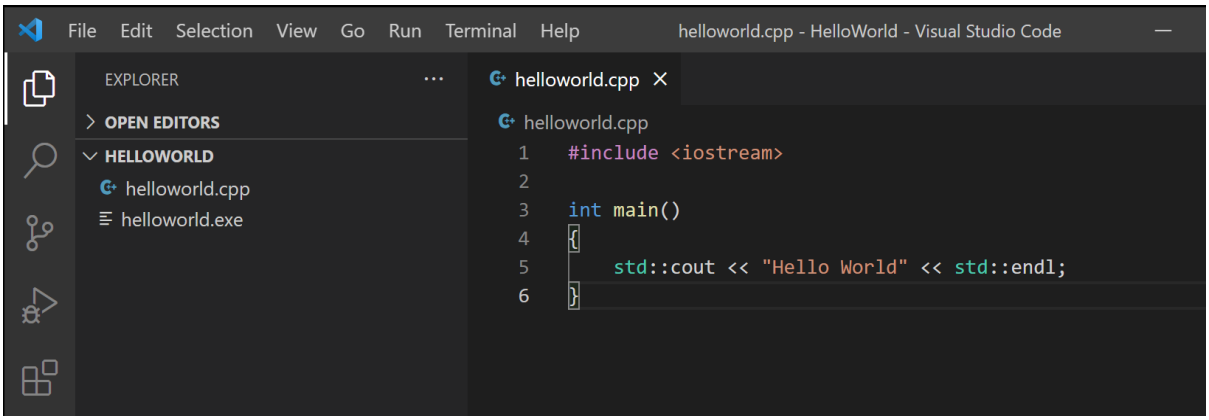


Рисунок 3.2 Інтерфейс VS Code

Visual Studio Code – це передусім редактор і містить функції, необхідні для високопродуктивного редагування вихідного коду.

3.1.3 Microsoft Visual Studio

Visual Studio 2022 - найкраща версія Visual Studio. -розрядне середовище IDE спрощує роботу з великими проектами та складними робочими навантаженнями. При виконанні повсякденних справ, таких як написання коду та перемикання гілок, система реагує швидше та плавніше. А що з приводу помилок нестачі пам'яті? Вони скоро залишаться лише у спогадах.

Visual Studio максимізує зручність та продуктивність розробки за допомогою зручного та інтуїтивного інтерфейсу.

Інтегрована налагодження - найважливіша складова всіх продуктів Visual Studio. Можна проводити розбір коду, вивчати значення, що зберігаються в змінних, налаштовувати контрольні значення змінних, щоб відстежувати зміну значень, вивчати шлях виконання вашого коду та інші особливості роботи програмного забезпечення.

Visual Studio 2022 має вбудовану підтримку керування версіями Git для клонування, створення та відкриття власних репозиторіїв. Вікно інструментів Git

містить все необхідне для фіксації та надсилання змін до коду, управління гілками та вирішення конфліктів злиття.

Після аналізу усіх середовищ розробки я зупинився саме на Visual Studio. Після завантаження Visual Studio та його налаштування, відкриється вікно із можливістю створити новий проект або ж відкрити вже готовий. Після вибору «Create new project», відкриється вікно налаштувань проекту, зображене на рис.3.3

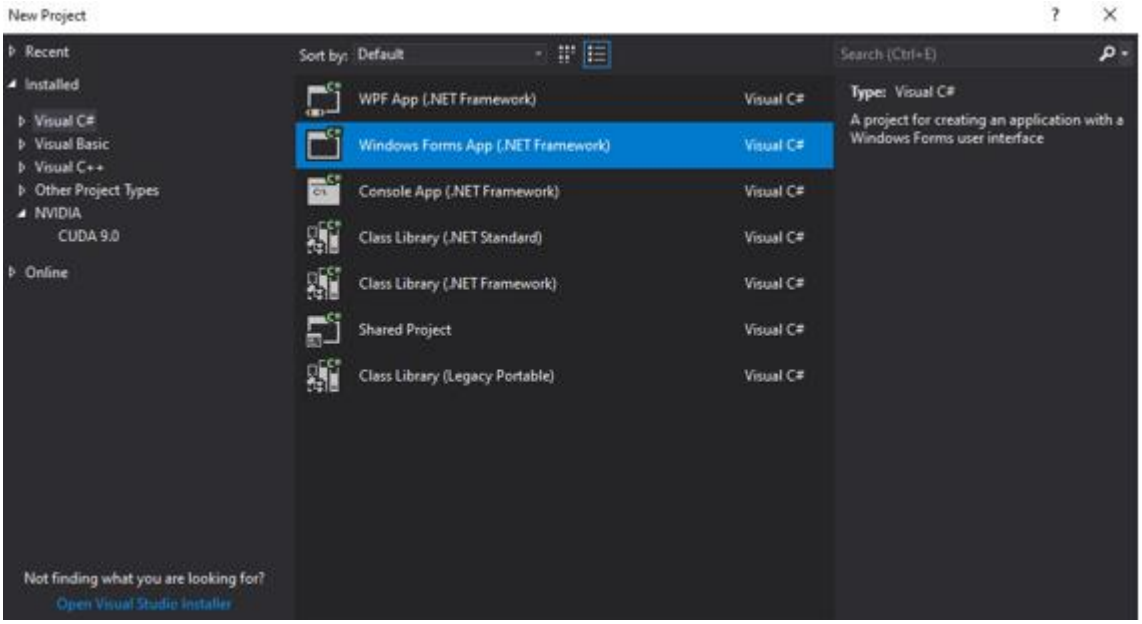


Рисунок 3.3 – Створення нового проекту в Visual Studio

Нам буде доступний вибір між мовами програмування, типом проекту тощо. Але так як розробляється конмольне програмне забезпечення на мові C#, потрібно вибрати «Console App (.Net Framework)». Після вибору та натиснення клавіші «OK», відкривається середовище розробки із базовим програмним кодом.

3.2 Налаштування логіки програми хешування

Для створення захищеного хеш-чейну будемо використовувати формулу математичного апарату ЛПС

$$S(t+1) = A \times S(t) + B \times U(t)$$

Задаємо породжувальні поліноми A розмірності 4x4 (рис.3.4) та B розмірності 4x1 (рис.3.5)

```
private static readonly int[,] A = new int[,]
{
    { 0, 0, 0, 1 },
    { 1, 0, 0, 0 },
    { 0, 1, 0, 1 },
    { 0, 0, 1, 0 }
};
```

Рисунок 3.4 - Матриця A.

```
private static readonly int[,] B = new int[,]
{
    { 1 },
    { 0 },
    { 0 },
    { 0 }
};
```

Рисунок 3.5 - Матриця B.

Вводимо секретне значення S(0) (рис. 3.6) без знання якого зловмиснику доведеться вручну перебирати всі варіанти значень.

Задаємо послідовності n та m (рис. 3.7) для унікальності нашого шифрування

```
var s = new int[,]
{
    { 0 },
    { 0 },
    { 0 },
    { 0 }
};
```

Рисунок 3.6 - Матриця S(0).

```
private static readonly int[] Un = new int[] { 0, 1, 1, 1, 0, 0, 0, 0 };
private static readonly int[] Um = new int[] { 1, 1, 0, 0, 0, 0, 0, 0 };
```

Рисунок 3.7 - Послідовності n та m.

При обрахуванні алгоритму підстановку даних з послідовності починаємо з кінця, тобто у випадку U_n послідовності: $U_0 = 0$, $U_1 = 0$, $U_2 = 0$, $U_3 = 0$, $U_5 = 1$, $U_6 = 1$, $U_7 = 1$, $U_8 = 0$. Аналогічно беремо дані з U_m послідовності.

Для надійного хешування даних ми виконуємо алгоритм тричі, наш перший крок це обрахування за формулою підставляючи дані з U_n послідовності, в настовному кроці ми підставляємо дані з U_m послідовності, і так як використання однакових послідовностей не підряд ніяк не зашкоджує корректній роботі алгоритму третім кроком обраховуємо ще один раз алгоритм з послідовністю U_n, тобто повний цикл алгоритму це - S(0) > S(n) > S(m+n) > S(m+2n). Далі розглянемо детальніше.

Шифрування відбувається в декілька етапів, а саме:

- $S(n1) = A \times S(0) + B \times Un(0)$;

- $S(n2) = A \times S(n1) + B \times Un(1)$;

- $S(n3) = A \times S(n2) + B \times Un(2)$;

- $S(n4) = A \times S(n3) + B \times Un(3)$;

- $S(n5) = A \times S(n4) + B \times Un(4)$;

- $S(n6) = A \times S(n5) + B \times Un(5)$;

- $S(n7) = A \times S(n6) + B \times Un(6)$;

- $S(n8) = A \times S(n7) + B \times Un(7)$.

Після одного повного циклу ми переходимо до другої фази орбахування, виходячи з останнього результату функції попередньої послідовності ми розпочинаємо обраховувати алгоритм знову, підставляючи нові значення, в даному випадку буде виглядати так:

$$S(m1) = A \times S(n8) + B \times Um(0).$$

Коли обрахунок циклу з U_m послідовності завершається , знову повертаємось до U_n послідовності.

$$S(n1) = A \times S(m8) + B \times Un(0).$$

Отримане значення S(n8) і є хешем нашого документу, яке ми використовуємо в обчисленні наступних блоків як S(0).

В консоль виводиться весь чейн обрахунків алгоритму крок за кроком.

3.3 Реалізація програми для хешування даних за допомогою мови C#

Для зручності відслідковування процесу обчислення виводимо в консоль текстові позначення. За допомогою циклу виводимо на екран результат кожного етапу та контрольну суму першого циклу обчислення з використанням вхідної послідовності Un.(рис. 3.8)

```
Console.WriteLine("Canculate Un");
Console.WriteLine("S0:");
PrintMatrix(s);

for (int i = 0; i < Un.Length; i++)
{
    s = CalcS(A, B, s, Un[i]);
    Console.WriteLine($"S{i + 1}:");
    PrintMatrix(s);
}
```

Рисунок 3.8 - Результат обчислень першої послідовності

Повторюємо дії з попереднього етапу, але з використанням вхідної послідовності Um. Приймаючи S(n8) з попереднього циклу, як S(0) у новому. (рис. 3.9)

Виводимо на екран результат кожного етапу та контрольну суму першого циклу обчислення з використанням вхідної послідовності Uz, в ролі якої може виступати послідовність Un, яку ми використовували в першому циклі алгоритму, це ніяк не завадить захисту. (рис. 3.10)

```
Console.WriteLine("Canculate Um");
Console.WriteLine("S0:");
PrintMatrix(s);

for (int i = 0; i < Um.Length; i++)
{
    s = CalcS(A, B, s, Um[i]);
    Console.WriteLine($"S{i + 1}:");
    PrintMatrix(s);
}
```

Рисунок 3.9 - Результат обчислень другої послідовності

```
Console.WriteLine("Canculate Un");
Console.WriteLine("S0:");
PrintMatrix(s);

for (int i = 0; i < Un.Length; i++)
{
    s = CalcS(A, B, s, Un[i]);
    Console.WriteLine($"S{i + 1}:");
    PrintMatrix(s);
}
```

Рисунок 3.10 - Результат обчислень третьої послідовності

Створюємо функція PrintMatrix() яка нараховує в собі два цикла, завдяки яким обраховується розмір та виводиться на екран матриця згідно з заданою розмірністю, як показано на рисунку 3.11.

За допомогою циклу від 0 відповідно до розмірності нашого масиву в консольпослідовно виводяться його елементи. (рис. 3.12)

```
public static void PrintMatrix(int[,] matrix)
{
    for (int i = 0; i < matrix.GetLength(0); i++)
    {
        for (int j = 0; j < matrix.GetLength(1); j++)
        {
            Console.Write($"{matrix[i, j]} ");
        }
        Console.WriteLine();
    }
}
```

Рисунок 3.11 - Виведення матриці

```
public static void PrintArray(int[] array)
{
    for (int i = 0; i < array.Length; i++)
    {
        Console.Write($"{array[i]} ");
    }
    Console.WriteLine();
}
```

Рисунок 3.12 - Виведення елементів масиву

Створюємо функцію MatrixMultiplication(), у ній ініціалізуємо майбутні розміри матриці C, а далі циклічно перемножуємо елементи матриць A та B на основі кількості їх рядків та стовпців, аж поки не отримаємо значення розмірності нашої матриці C (рис. 3.13)

Далі створюємо ще одну функцію MatrixMultiplication() для наступного множення однієї матриці на елемент послідовності , який відповідає наявному порядковому номеру в послідовності. Вичислення порядкового номеру починається з кінця послідовності.

Створюємо функцію MatrixAdd() для циклічного додавання елементи першої матриці, до елементів другої матриці, для визначення суми їх елементів, і знаходження S(n) (рис. 3.15)

```
public static int[,] MatrixMultiplication(int[,] matrixA, int[,] matrixB)
{
    var matrixC = new int[matrixA.RowsCount(), matrixB.ColumnsCount()];

    for (var i = 0; i < matrixA.RowsCount(); i++)
    {
        for (var j = 0; j < matrixB.ColumnsCount(); j++)
        {
            matrixC[i, j] = 0;

            for (var k = 0; k < matrixA.ColumnsCount(); k++)
            {
                matrixC[i, j] += matrixA[i, k] * matrixB[k, j];
            }
        }
    }

    return matrixC;
}
```

Рисунок 3.13 - Множення матриць

```
Ссылок: 1
public static int[,] MatrixMultiplication(int[,] matrixA, int scalar)
{
    var matrixC = new int[matrixA.RowsCount(), matrixA.ColumnsCount()];

    for (var i = 0; i < matrixA.RowsCount(); i++)
    {
        for (var j = 0; j < matrixA.ColumnsCount(); j++)
        {
            matrixC[i, j] = matrixA[i, j] * scalar;
        }
    }

    return matrixC;
}
```

Рисунок 3.14 - Множення матриці та значення послідовності

```
Ссылок: 1
static int[,] MatrixAdd(int[,] matrixA, int[,] matrixB)
{
    var matrixC = new int[matrixA.RowsCount(), matrixB.ColumnsCount()];

    for (var i = 0; i < matrixA.RowsCount(); i++)
    {
        for (var j = 0; j < matrixB.ColumnsCount(); j++)
        {
            matrixC[i, j] = matrixA[i, j] + matrixB[i ,j];
        }
    }

    return matrixC;
}
```

Рисунок 3.15 - Додавання матриць

Створюємо функцію CalcS() , яка додає результати множення функцій MatrixMultiplication() відповідно до формули (1), для знаходження S(n), що в нашому

випадку є головною задачею, тому що останній елемент після усіх проведених матетатичних операцій і буде хешем (рис.3.16)

```
Ссылок: 3
static int[,] CalcS(int[,] a, int[,] b, int[,] s, int u)
{
    var p1 = MatrixMultiplication(a, s);
    var p2 = MatrixMultiplication(b, u);
    return MatrixAdd(p1, p2);
}
```

Рисунок 3.16 - Математика матриці станів

Створює дві функції RowsCount() та ColumnsCount() для визначення розмірності матриць, вони повертають значення довжини , як показано на рисунку 3.17 та рисунку 3.18.

```
Ссылок: 6
public static int RowsCount(this int[,] matrix)
{
    return matrix.GetLength(0);
}
```

Рисунок 3.17 - Визначення розмірності рядків

```
Ссылок: 7
public static int ColumnsCount(this int[,] matrix)
{
    return matrix.GetLength(1);
}
```

Рисунок 3.18 - Визначення розмірності стовбців

Висновки

В даному дипломнопу проекті було розроблено систему автоматичного шифрування документів.

Даний додаток можна використовувати для створення захищених чейнів документів для використання у різних навчальних закладах, або ж підприємствах.

У першому розділі було проаналізовано переваги та недоліки різних способів захисту інформації. Розглянуто основні параметри, яким повинні відповідати методи шифрування. Розглянуто сфери в яких використовуються дані методи шифрування. Проаналізовано методи захисту паролів від зловмисників.

У другому розділі були розглянуті можливості створення хеш-функцій на основі математичного апарату лінійних послідовнісних схем, а також наведено приклад його роботи. Проаналізовано переваги та недоліки захисту даного методу. Було розглянуто принципи роботи хеш-чейну та криптостійкійсть хеш-функцій. Проаналізовано теоретичні основи багаторівневого захисту від зловмисників.

У третьому розділі були виконані всі поставлені задачі, а саме: створено програму для автоматичного шифрування документів з власним секретним ключем, з максимальною швидкодією. Використано усі переваги захисту з допомогою хеш-чейну. Розроблене програмне забезпечення було протестовано на можливі несправності, або неполадки у виконанні.

Отже, ми отримали працездатний метод захисту документації за допомогою хеш-чейну на основі математичного апарата лінійно послідовнісних схем. Звісно ця робота буде і надалі розвиватись, в планах створити комфортний інтерфейс для звичайного користувача, щоб покращити практичність та зручність використання.

Перелік джерел посилання

1. Хеш-функції і аутентифікація повідомлень. Частина 1 [Електронний ресурс]. Режим доступу: https://pns.hneu.edu.ua/pluginfile.php/40812/mod_resource/content/1.pdf

2. Колізія хеш-функції [Електронний ресурс]. Режим доступу: https://dic.academic.ru/dic.nsf/ruwiki/647243

3. Семеренко В.П. Теорія і практика CRC кодів: нові результати на основі автоматних моделей //К:2015. С. 38-48.

4. Брюс Шнайєр Прикладна криптографія. Друге видання. Протоколи, алгоритми, вихідні тексти на мові С // С. 610

5. Види хеш функцій. Хеш – що це таке? [Електронний ресурс]. Режим доступу: https://bitserv.ru/uk/vidy-hesh-funkcii-hesh-chto-eto-takoe-opredelenie-znachenie-perevod/.

6. Технічний аспект поняття електронного підпису [Електронний ресурс]. Режим доступу: https://buklib.net/books/25233.

7. Хеш-таблиці [Електронний ресурс]. Режим доступу: http://algolist.ru/ds/s_has .php.

8. Хеш-таблиці: теорія і практика [Електронний ресурс]. Режим доступу: http://rus-linux.net/MyLDP/algol/hash_tables_theory_and_practice.html.

9. Що таке алгоритм хешування MD5 і як він працює? [Електронний ресурс]. Режим доступу: https://www.avast.com/c-md5-hashing-algorithm

10. Що таке SHA-2 і як він працює? [Електронний ресурс]. Режим доступу: https://www.comparitech.com/blog/information-security/what-is-sha-2-how-does-it-work/

11. Чудеса хешування [Електронний ресурс]. Режим доступу: https://www.kaspersky.ru/blog/the-wonders-of-hashing/3633/.

12. Як шифруються документи та захищаються паролі [Електронний ресурс]. Режим доступу: https://blog.elcomsoft.ru/2021/10/shifrovanie-heshirovanie-kak-shifruyutsya-dokumenty-i-zashhishhayutsya-paroli/.

13. Семеренко В.П. Про взаємозв'язки завадостійкого кодування криптографії та технічної діагностики // УДК 681.325 С. 2

14. Семеренко В.П., Ширшова П.В. Розробка хеш-функції на основі поточного шифрування // С. 5

15. Хеш-чейн [Електронний ресурс]. Режим доступу: https://link.springer.com/-referenceworkentry/10.1007/978-1-4419-5906-5_780

Додаток А

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри _____

д.т.н., проф.

_____О. Д. Азаров

“___” _____ 2022р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання дипломного проекту

«Захист документації на основі технології хеш-чейну»
08-23.БДП.020.00.000 ТЗ

Науковий керівник к.т.н., доц. каф. ОТ

_____ Семеренко В. П.

Студента групи 2КІ-18б

_____ Бабейко Б. О.

Вінниця 2022

1 Підстава для виконання дипломного проекту (ДП)

1.1 Підставою для виконання дипломного проекту

2 Мета і призначення ДП

2.1 Мета проекту - процес розробки автоматизованого шифрування для полегшення та пришвидшення процесу шифрування документів, для успішного використання к навчальних заходах та на виробництві.

2.2 Призначення розробки - виконання бакалаврського дипломного проекту із подальшим розвитком та підтримкою продукту.

3 Вихідні дані для виконання БДП

3.1 Розгляд принципів захисту інформації за допомогою хешування.

3.2 Проведення аналізу математичного апарату лінійних послідовнісних схем.

3.3 Багаторівневий захист інформації за допомогою хеш-чейну.

4 Технічні вимоги до виконання ДП

4.1 Зручна програма для надійного хешування даних з максимальною швидкодією.

5 Етапи ДП та очікувані результати. (Таблиця А.1)

6 Матеріали, що подаються до захисту ДП

До захисту БП подаються пояснювальна записка БДП, презентація, протокол попереднього захисту БДП на кафедрі, відгук наукового керівника, відгук рецензента, анотації до БДП українською та іноземною мовами, нормоконтроль про відповідність оформлення БДП діючим вимогам.

7 Порядок контролю виконання та захисту ДП

Виконання етапів графічної та розрахункової документації БДП контролюється науковим керівником згідно зі встановленими термінами. Захист БДП відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

Таблиця А.1 - етапи дипломної роботи та очікувані результати

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Аналіз стану досліджень на сьогоднішній день	01.05.2022	04.05.2022	Огляд літературних джерел, задачі досліджень, пошук інформації
2	Аналітичний огляд існуючих підходів та програмних засобів для реалізації хешування	05.05.2022	12.05.2022	Розділ 1
3	Особливості створення програмних засобів для багаторівневого захисту документів	13.05.2022	17.05.2022	Розділ 2
4	Розробка програми для автоматизованого хешування документів	18.05.2022	22.05.2022	Розділ 3
№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
5	Практична реалізація	23.05.2022	29.05.2022	Програмний продукт
6	Тестування додатку. Виправлення недоліків	30.05.2022	02.06.2022	Розділ 3
7	Оформлення пояснювальної записки, презентації	03.06.2022	10.06.2022	Пояснювальна записка, презентація

8 Вимоги до оформлення ДП

При оформлюванні БДП використовуються:

- ДСТУ 3008 : 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

- ДСТУ 8302 : 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

- ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

- документами на які посилаються у вище вказаних.

Технічне завдання до виконання отримав _____ Бабейко Б. О.

Додаток Б

Лістинг програми

```
using System;

namespace Hashchain
{
    internal class Program
    {
        private static readonly int[,] A = new int[,]
        {
            { 0, 0, 0, 1 },
            { 1, 0, 0, 0 },
            { 0, 1, 0, 1 },
            { 0, 0, 1, 0 }
        };

        private static readonly int[,] B = new int[,]
        {
            { 1 },
            { 0 },
            { 0 },
            { 0 }
        };

        private static readonly int[] Un = new int[] { 0, 1, 1, 1, 0, 0, 0, 0 };
```

```
        private static readonly int[] Um = new int[] { 1, 1, 0, 0, 0, 0, 0, 0 };


static void Main(string[] args)
{
   Console.WriteLine("Matrix A:");
                PrintMatrix(A);
                Console.WriteLine("Matrix B:");
                PrintMatrix(B);
                Console.Write("Un:");
                PrintArray(Un);
                Console.Write("Um:");
                PrintArray(Um);

                var s = new int[,]
                {
                    { 0 },
                    { 0 },
                    { 0 },
                    { 0 }
                };

                Console.WriteLine("Canculate Un");
                Console.WriteLine("S0:");
                PrintMatrix(s);
```

```
            for (int i = 0; i < Un.Length; i++)
{
                s = CalcS(A, B, s, Un[i]);
                Console.WriteLine($"S{i + 1}:");
                PrintMatrix(s);
            }

            Console.WriteLine("Canculate Um");
            Console.WriteLine("S0:");
            PrintMatrix(s);

            for (int i = 0; i < Um.Length; i++)
            {
                s = CalcS(A, B, s, Um[i]);
                Console.WriteLine($"S{i + 1}:");
                PrintMatrix(s);
            }

            Console.WriteLine("Canculate Un");
            Console.WriteLine("S0:");
            PrintMatrix(s);

            for (int i = 0; i < Un.Length; i++)
            {
                s = CalcS(A, B, s, Un[i]);
```

```
                Console.WriteLine($"S{i + 1}:");
                PrintMatrix(s);
            }
        }

        public static void PrintMatrix(int[,] matrix)
        {
            for (int i = 0; i < matrix.GetLength(0); i++)
            {
                for (int j = 0; j < matrix.GetLength(1); j++)
{
                    Console.Write($"{matrix[i, j]} ");
                }
                Console.WriteLine();
            }
        }

        public static void PrintArray(int[] array)
        {
            for (int i = 0; i < array.Length; i++)
            {
                Console.Write($"{array[i]} ");
            }
            Console.WriteLine();
        }
```

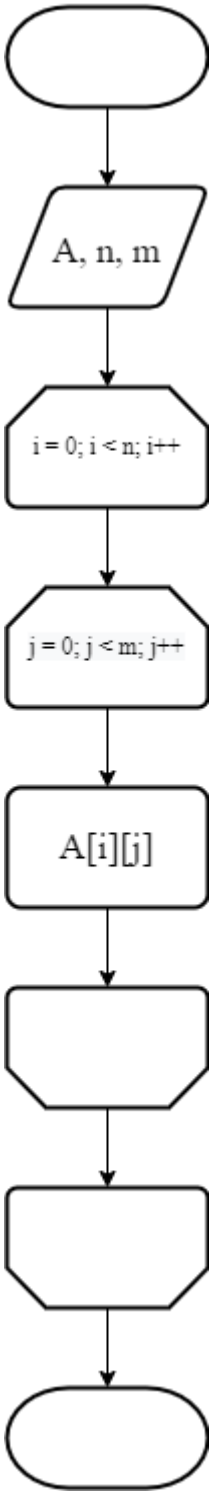
```
        public static int[,] MatrixMultiplication(int[,] matrixA, int[,] matrixB)
        {
            var matrixC = new int[matrixA.RowsCount(),
matrixB.ColumnsCount()];

            for (var i = 0; i < matrixA.RowsCount(); i++)
            {
                for (var j = 0; j < matrixB.ColumnsCount(); j++)
                {
                    matrixC[i, j] = 0;

                    for (var k = 0; k < matrixA.ColumnsCount(); k++)
                    {
                        matrixC[i, j] += matrixA[i, k] * matrixB[k, j];
                    }
                }
            }

            return matrixC;
        }

        public static int[,] MatrixMultiplication(int[,] matrixA, int scalar)
        {
            var matrixC = new int[matrixA.RowsCount(),
matrixA.ColumnsCount()];
```

```
            for (var i = 0; i < matrixA.RowsCount(); i++)
            {
                for (var j = 0; j < matrixA.ColumnsCount(); j++)
                {
                    matrixC[i, j] = matrixA[i, j] * scalar;
                }
            }

            return matrixC;
        }

        static int[,] MatrixAdd(int[,] matrixA, int[,] matrixB)
        {
            var matrixC = new int[matrixA.RowsCount(),
matrixB.ColumnsCount()];

            for (var i = 0; i < matrixA.RowsCount(); i++)
            {
                for (var j = 0; j < matrixB.ColumnsCount(); j++)
                {
                    matrixC[i, j] = matrixA[i, j] + matrixB[i ,j];
                }
            }

            return matrixC;
```

```
        }

        static int[,] CalcS(int[,] a, int[,] b, int[,] s, int u)
    {
            var p1 = MatrixMultiplication(a, s);
            var p2 = MatrixMultiplication(b, u);
            return MatrixAdd(p1, p2);
    }
}

static class MatrixExt
{
        public static int RowsCount(this int[,] matrix)
        {
            return matrix.GetLength(0);
        }

        public static int ColumnsCount(this int[,] matrix)
        {
            return matrix.GetLength(1);
        }
}
}
```

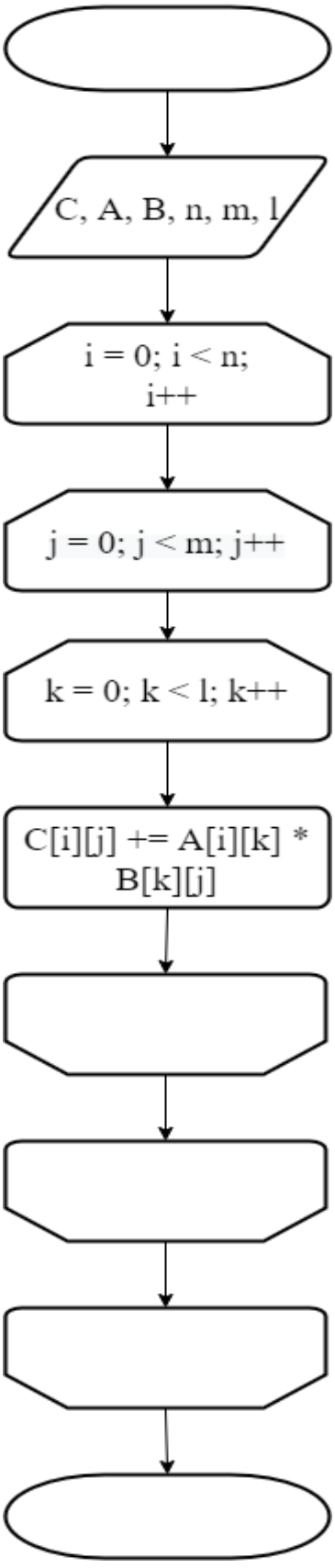
Додаток В

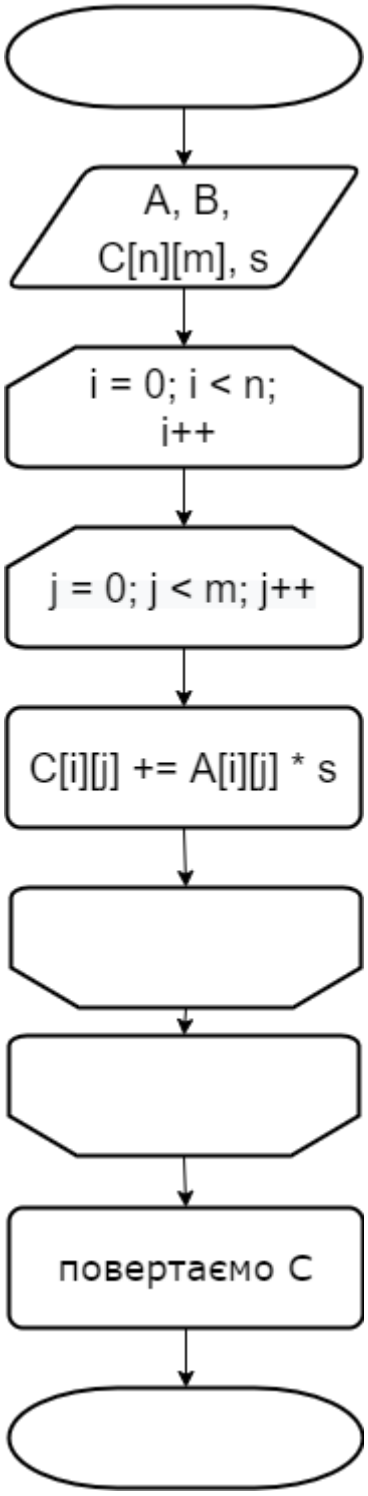
Блок-схема алгоритму функції PrintMatrix()



Додаток Г

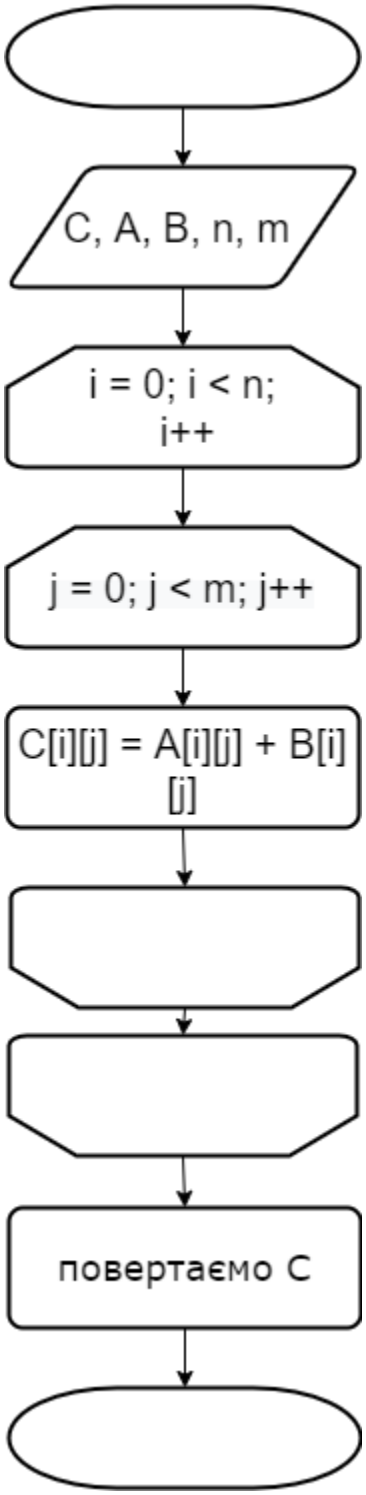
Блок-схема алгоритму функцій MatrixMultiplication()




A, B,
C[n][m], s
i = 0; i < n;
i++
j = 0; j < m; j++
C[i][j] += A[i][j] * s
повертаємо C

Додаток Д

Блок-схема алгоритму функції MatrixAdd()



Додаток Е

ПРОТОКОЛ

ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Захист документації на основі технології хеш-чейну

Тип роботи: бакалаврська дипломна робота

Підрозділ кафедра обчислювальної техніки

Показники звіту подібності Unicheck

Оригінальність 79,5% Схожість 20,5%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ 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.
- □ 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 _____ Захарченко С.М.

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

Автор роботи _____ Бабейко Б.О.

Керівник роботи _____ Семеренко В.П.