Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

РОЗРОБКА ТА ДОСЛІДЖЕННЯ СИСТЕМИ АВТОМАТИЧНОГО КОНТРОЛЮ ТА АРХІВАЦІЇ РЕЗУЛЬТАТІВ ІСПИТІВ
(тема)

Виконав:
студент 2 курсу, групи ІНФм-22-2
Бганцов Є.О.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник проф. Машталір С.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Бганцову Євгенію Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка та дослідження системи автоматичного контролю та архівації результатів іспитів

затверджена наказом по університету від 3 листопада 2023 року № 1280Ст

2. Термін подання студентом роботи до екзаменаційної комісії 25 грудня 2023 р.

3. Вихідні дані до роботи бази даних, перелік використовуваних програмних засобів, теоретичні відомості про методи стиснення даних.

4. Перелік питань, що потрібно опрацювати в роботі

1. Огляд методів стиснення даних.
2. Використання алгоритму симетричного шифрування AES.
3. Математичні моделі методів шифрування.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність проблеми стиснення даних, постановка задачі, модель бази даних, використані методи стиснення, симетричне шифрування AES.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	03.11.2023	
2	Аналіз завдання, підбір літератури	04.11.23-05.11.23	
3	Аналіз літератури з досліджуваної проблеми	05.11.23-06.11.23	
4	Аналіз засобів обробки даних	06.11.23-08.11.23	
5	Розробка методів обробки даних	08.11.23-14.11.23	
6	Програмна реалізація	14.11.23-25.11.23	
7	Оформлення пояснювальної записки	25.11.23-27.11.23	
8	Перевірка на плагіат	03.12.2023	
9	Рецензування	05.12.2023	
10	Підготовка презентації та доповіді	07.12.2023	
11	Занесення роботи в електронний архів	02.01.2024	
12	Попередній захист кваліфікаційної роботи	03.01.2024	

Дата видачі завдання 3 листопада 2023 р.

Студент _____
(підпис)

Керівник роботи _____ проф.Машталір С.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 68 с., 2 табл., 59 рис., 30 джерел.

СИММЕТРИЧНЕ ШИФРУВАННЯ AES, ШИФРУВАННЯ DEFLATE, КОДУВАННЯ RLE, АРХІВУВАННЯ ЧИСЛОВИХ ДАНИХ, ОБРОБКА ЧИСЛОВИХ ДАНИХ.

Об’єктом дослідження є масив числових даних.

Метою дослідження є розробка методів, що базуються на використанні методів шифрування, які дозволяють зменшити обсяг даних та захистити їх від зловмисників.

Використано методи симетричного та лінійного шифрування, а також архівування. Проведено дослідження методів шифрування числових даних та методів архівування масивів чисел, їх зачист та попередня обробка. Проведено аналіз методів шифрування. Розроблено застосунок з іспиту та контролю даних.

У результаті дослідження здійснена програмна реалізація системи автоматичного контролю та архівації даних з іспитів.

AES SYMMETRIC ENCRYPTION, DEFLATE ENCRYPTION, RLE ENCODING, NUMERICAL DATA ARCHIVING, NUMERICAL DATA PROCESSING.

The object of research is an array of numerical data.

The purpose of the reseaech is to develop methods based on the use of encryption methods that allow to reduce the amount of data and protect it from intruders.

The methods used are symmetric and linear encryption, as well as archiving. The methods of encrypting numerical data and methods of archiving arrays of numbers, their cleaning and pre-processing are researched. An analysis of encryption methods is carried out. An application for data examination and control was developed.

As a result of the research, a software implementation of the system for automaticcontrol and archiving of exam data was carried out.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 7
Вступ 8
1 Огляд використаних у дослідженні методів 9
1.1 Метод стиснення Deflate 10
1.1.1 Метод стиснення з втратами LZ77 11
1.1.2 Метод стиснення Хаффмана 14
1.2 Run-Length Encoding 16
1.3 AES шифрування 18
1.4 Постановка задачі дослідження 20
2 Математичні моделі методів шифрування 22
2.1 Математична модель методу шифрування AES 22
2.1.1 Основи шифру 22
2.1.2 Раундове перетворення AES 25
2.1.3 Операція ShiftRows 27
2.1.4 Операция MixColumns 28
2.1.5 Ключовий розклад AES 30
2.1.6 Розшифрування AES 32
2.2 Run Length Encoding 35
2.2.1 Загальні характеристики RLE 35
2.2.2 Приклад роботи RLE 36
3 Комп'ютерна модель фільтрації зображень 40
3.1 Обґрунтування вибору середовища програмної реалізації 40
3.2 Програмна реалізація 42
3.2.1 Робота з базою даних 43
3.2.2 Імпортування даних з файлу 45
3.2.3 Шифрування даних 47
3.2.4 Дешифрування даних 50
3.2.5 Архівація даних 52

3.3 Інструкція користувача.....57
3.4 Тестування розробленої моделі.....61
Висновки.....65
Перелік джерел посилання.....66

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AES – Advanced Encryption Standard (розширений стандарт шифрування)

RLE – Run-Length Encoding (кодування повторюваної довжини)

TLS – Transport Layer Security (протокол захисту транспортного рівня)

SSL – Secure Sockets Layer (рівень захищених сокетів)

ВСТУП

У сучасному цифровому світі безпека та збереження інформації є критичним завданням для організацій та користувачів. У цьому контексті два ключових аспекти виступають на передньому плані: шифрування та архівація.

Шифрування використовується для того, щоб зробити інформацію незрозумілою для будь-якої сторони, крім тих, кому вона призначена. Основною метою шифрування є забезпечення конфіденційності, а також цілісності та автентичності даних. Цей процес необхідний та забезпечує захист від прослуховування, несанкціонованого доступу, зловживання, перехоплення та модифікації інформації під час її передачі чи зберігання.

Архівація дозволяє зберегти обсяг даних та забезпечити їхню ефективну передачу та збереження. За допомогою методів стиснення та архівації можна зменшити обсяг даних, не втрачаючи важливої інформації.

Крім економії простору, архівація дозволяє полегшити обробку та обмін даними, зменшуючи час передачі та завантаження. Застосування методів стиснення під час архівації може також сприяти економії пропускної здатності мережі під час передачі даних через Інтернет чи локальну мережу.

Основне завдання шифрування та архівації даних полягає у захисті інформації та оптимізації її збереження та передачі.

Актуальність дослідження в області захисту даних визначається сучасними викликами та загрозами, що стикаються з організаціями та користувачами в цифровому світі. З несприятливими зростанням кількості кібератак, витоків даних та інших загроз кібербезпеки, збереження конфіденційності і безпеки даних стає першочерговим завданням для багатьох секторів.

1 ОГЛЯД ВИКОРИСТАНИХ У ДОСЛІДЖЕННІ МЕТОДІВ

Розробка та дослідження системи автоматичного контролю та архівації результатів технічних іспитів є важливою темою у сучасній індустрії та освітній сфері. Особливо ця область є критичною для інженерної та технічної освіти, де вимірювання та оцінювання вимагають специфічних підходів [1].

Методи шифрування та стиснення є критично важливими в сучасному цифровому світі з різними аспектами їх важливості. Вони допомагають зробити інформацію захищеною та компактною.

Шифрування даних важливе для забезпечення конфіденційності і захисту від несанкціонованого доступу. Воно дозволяє зашифровувати дані так, що тільки власник правильного ключа може розшифрувати їх. Це критично важливо для банківських операцій, медичних записів, комерційної інформації та багатьох інших сфер.

Шифрування також використовують для безпеки під час передачі даних через мережу. Протоколи шифрування, такі як TLS/SSL, дозволяють захищати інформацію, яка передається через Інтернет, від перехоплення і зловживання даними [2].

Методи стиснення даних дозволяють зменшити обсяг інформації, що потребується для зберігання та передачі. Це сприяє більш ефективному використанню обчислювальних ресурсів і більш швидкій передачі даних через мережі. Методи стиснення особливо корисні для медіа файлів, таких як зображення, аудіо та відео. Вони дозволяють зберігати ці файли з меншим обсягом, що важливо для швидкого завантаження і зберігання даних на пристроях з обмеженим місцем для зберігання.

Через зменшення об’єму даних справедливо що буде зменшення часу передачі даних: Стиснення даних може значно зменшити час передачі даних через мережу, особливо на мобільних мережах, які мають досі малу швидкість передачі [3]. Це робить інтернет-сервіси більш доступними та швидкими для користувачів.

Загалом, методи шифрування та стиснення даних є невід'ємною частиною сучасної інформаційної безпеки та оптимізації ресурсів, дозволяючи зберігати, передавати та обробляти дані ефективніше та безпечніше.

Дане дослідження являє собою інструмент для роботи з великою кількістю числової інформації. Великі масиви інформації потребують таку ж саму ємність з боку сховища: жорстких дисків, хмарних сервісів, тощо. Також деякі дані мають бути персональні, не повинні попасти в руки іншим особам.

В ході роботи розглядалися декілька методів обробки інформації які вирішують такі проблеми. Серед них методи стиснення та шифрування числових масивів.

1.1 Метод стиснення Deflate

Метод стиснення DEFLATE – це популярний алгоритм стиснення даних, який комбінує два основні під протоколи: LZ77 (згортання з втратами) та алгоритм кодування Хаффмана (без втрат). Введений у 1993 році, цей метод стиснення використовується у різних форматах архівів, вебсторінках, зображеннях та інших застосуваннях, завдяки його ефективності та універсальності. Даний метод має два ступені стиснення. В два кроки через алгоритм стиснення з втратами LZ77 та без втрат Хаффмана [4].

DEFLATE починається зі ступеня згортання, використовуючи алгоритм LZ77. Він працює, знаходячи послідовності байтів, які повторюються в даних, і замінюючи їх короткими вказівниками на попередні входи в послідовності. Це допомагає зменшити обсяг даних.

Після згортання LZ77 дані подаються на вхід в алгоритм кодування Хаффмана для стиснення без втрат. Алгоритм кодування Хаффмана використовується для кодування знаків (символів) в деякому форматі, який

відповідає їхній частоті входження в дані. Знаки з більшими частотами кодуються коротшими бітовими послідовностями, що сприяє подальшій стискальності (рис. 1.1).

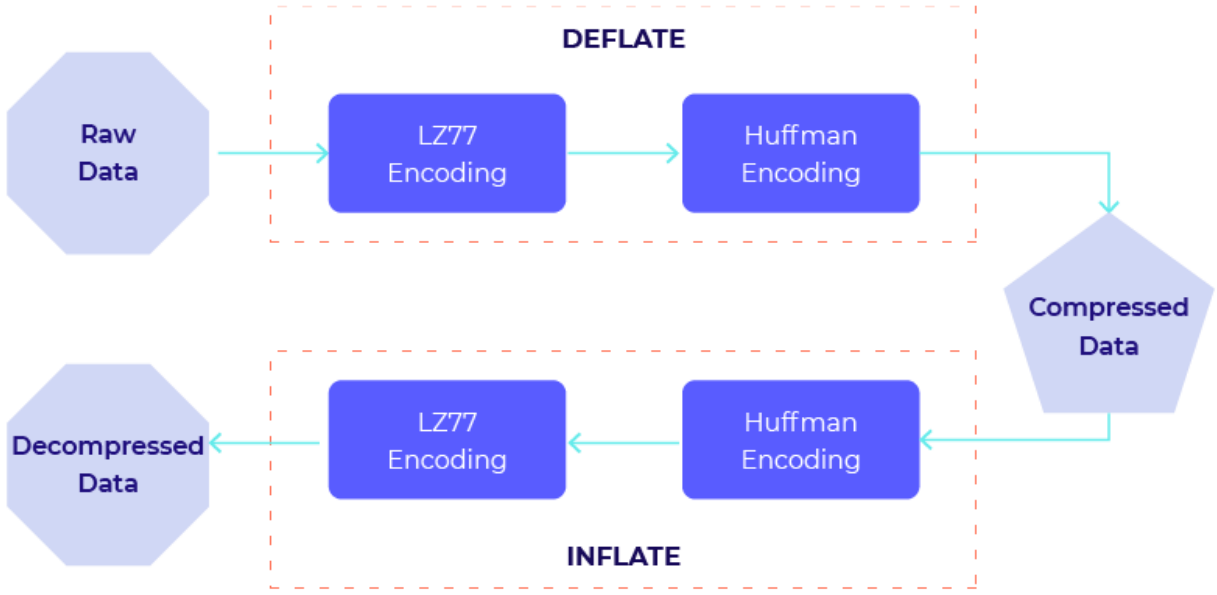

Рисунок 1.1 – DEFLATE стиснення

DEFLATE є ефективним методом стиснення, який використовується для різних типів даних, включаючи тексти, зображення та інші бінарні файли. Він використовує комбінацію методів стиснення без втрат та з втратами, що дозволяє досягти гарних результатів. DEFLATE широко підтримується і використовується в багатьох програмах та стандартах, таких як ZIP-архіви, GZIP, PNG-зображення та інші.

1.1.1 Метод стиснення з втратами LZ77

LZ77 є одним із ключових компонентів методу стиснення DEFLATE та також може використовуватися самостійно в інших контекстах як метод стиснення даних. LZ77 (Lempel-Ziv 1977) – це алгоритм стиснення даних без втрат, який був розроблений Абрахамом Лемпелем і Джейкобом Зівим.

Основний принцип роботи алгоритму LZ77 (рис. 1.2) є пошук повторюваних послідовностей [5]. Алгоритм сканує вхідний потік даних і намагається знайти повторювані послідовності байтів. Він використовує вікно буфера, що обмежується певною довжиною, для зберігання попередніх даних.

Якщо алгоритм виявляє повторювану послідовність то він замінює її вказівниками на попередні входи в послідовність. Ці вказівники складаються з двох чисел, які відповідають за відстань від поточного місця до початку повторюваної послідовності (зсув) та довжину самої послідовності.

Після заміни повторюваної послідовності вказівником на попередні входи, алгоритм видаляє деяку частину даних і додає нові дані до вікна буфера, щоб зберегти його розмір.

Алгоритм повторює весь цей процес, поки весь вхідний потік даних не буде оброблений.

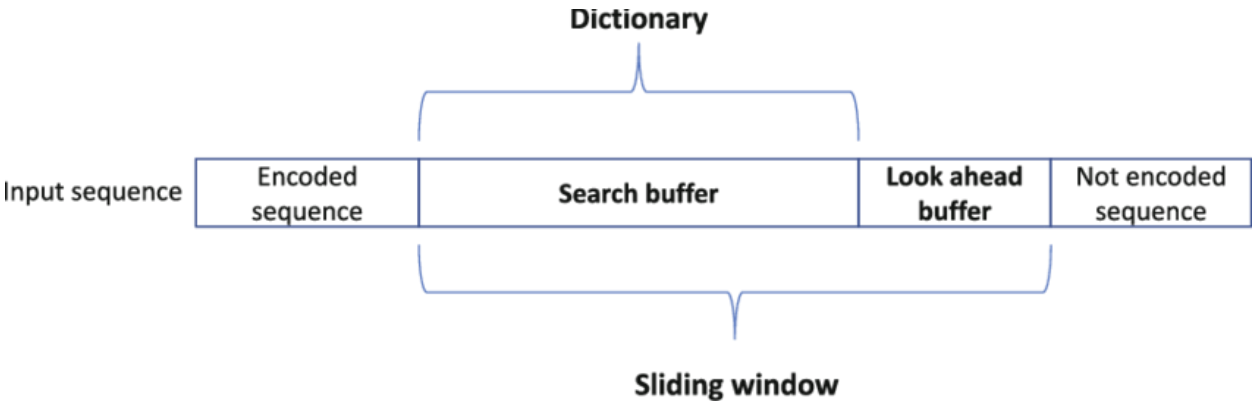

Рисунок 1.2 – Стиснення з втратами LZ77

Процес стиснення виглядає так. Послідовно зчитуються символи вхідного потоку та відбувається перевірка, чи існує у створеній таблиці рядків такий рядок. Якщо такий рядок існує, зчитується наступний символ, а якщо рядок не існує, в потік заноситься код попереднього знайденого рядка, рядок заноситься в таблицю, а пошук починається знову [6].

Наприклад, якщо стискають байтові дані (текст), рядків у таблиці виявиться 256 (від «0» до «255»). Якщо використовується 10-бітовий код, то

під коди для рядків залишаються значення діапазоні від 256 до 1023. Нові рядки формують таблицю послідовно, тобто можна вважати індекс рядка її кодом.

Алгоритму декодування на вході потрібно лише закодований текст, оскільки він може відтворити відповідну таблицю перетворення безпосередньо закодованим текстом. Алгоритм генерує однозначно код, що декодується за рахунок того, що кожного разу, коли генерується новий код, новий рядок додається в таблицю рядків (рис. 1.3). LZW постійно перевіряє, чи є рядок вже відомим, і якщо так, виводить існуючий код без генерації нового [7]. Таким чином, кожен рядок зберігатиметься в єдиному екземплярі і матиме свій унікальний номер. Отже, при дешифруванні при отриманні нового коду генерується новий рядок, а при отриманні вже відомого, рядок вилучається зі словника.

В результаті стиснення даних за допомогою LZ77 вхідний потік даних може бути представлений у вигляді послідовності вказівників на попередні входи в дані, що допомагає значно зменшити обсяг даних.

	k	<0,k>
k	a	<0,a>
k, a	b	<0,b>
k, a, b	ab	<2,b>
k, a, b, ab	aba	<4,a>
k, a, b, ab, aba	ba	<3,a>
k, a, b, ab, aba, ba	$abab$	<5,b>
k, a, b, ab, aba, ba, $abab$	z	<0,z>

Рисунок 1.3 – Приклад роботи алгоритму

Алгоритм LZ77 використовується у різних форматах архівів, стиснення тексту та інших застосуваннях, де потрібно стиснути дані без втрати якості. Важливим є вибір параметрів, таких як розмір вікна буфера та довжина вікна пошуку, для досягнення оптимальної стискальності в конкретних сценаріях.

1.1.2 Метод стиснення Хаффмана

Стиснення Хаффмана – це метод стиснення даних без втрат, який був розроблений Девідом Хаффманом у 1950 році. Він базується на ідеї використання змінної довжини коду для представлення символів залежно від їхньої частоти входження в дані [8]. Цей метод стиснення даних широко використовується в різних форматах архівів, кодування тексту, компресії зображень та інших застосуваннях.

Префіксні кодування представляються символи з абетки за бітовими послідовностями (кодами), один код для кожного символу, у манері, в якій різні символи можуть бути представлені за допомогою бітових послідовностей різних довжин, але парсер може завжди однозначно розпізнавати закодовані рядки символ-за-символом. Ми визначаємо префіксний код у термінах бінарного дерева (рис. 1.4), у якому дві грані відрізняються від не листового вузла за допомогою мітки 0 і 1 і у якому листовий вузол відповідає один-до-одного з символом абетки; тоді код для символу являється послідовністю нулів і одиниць (рис. 1.5) на гранях від кореня до листового вузла з цим символом.

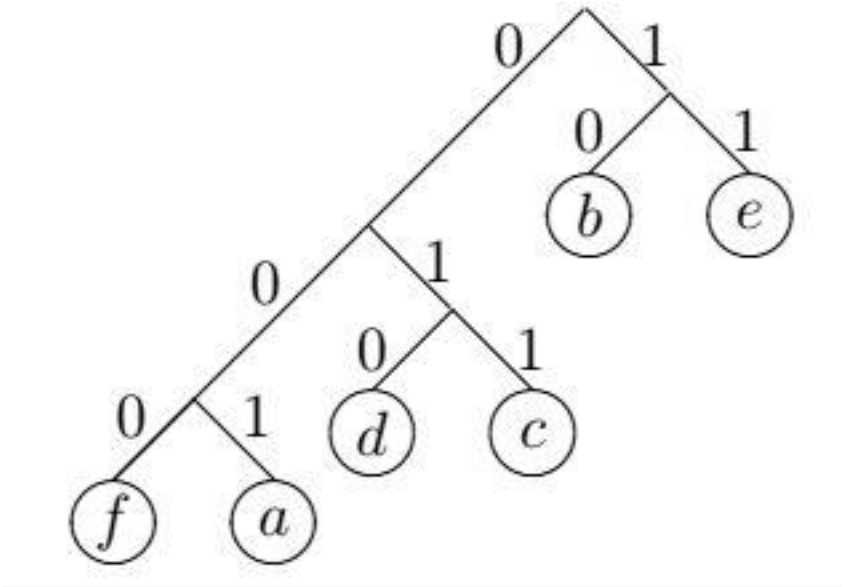

Рисунок 1.4 – Дерево Хаффмана

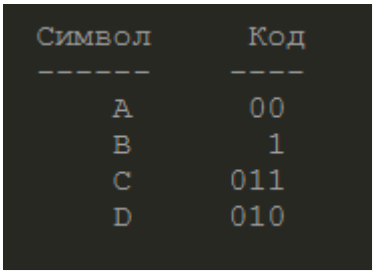

Рисунок 1.5 – Кодування Хаффмана

Парсер може розкодовувати наступний символ з закодованого потоку введення, йдучи вниз по дереву з кореня, на кожному кроці вибираючи грань у відповідності до наступного біту. Отримана абетка з відомою чистотою символів, і алгоритм Хаффмана дозволяє конструкцію з необов'язковим префіксним кодом (який відображає рядок з цими частотами символів, використовуючи декілька бітів будь-якої можливих префіксних кодів цієї абетки). Такі коди називаються кодами Хаффмана Зверніть увагу, що у форматі «deflate» коди Хаффмана для декількох абеток повинні не перевищувати певну максимальну довжину коду. Це обмеження ускладнює алгоритм для обчислення довжин кодів з частот символів.

Коди Хоффмана, які використовуються для кожної абетки у форматі deflate мають два додаткових правила:

– усі коди даної довжини бітів мають лексикографічні послідовні значення, у тому самому порядку символів, які вони представляють;

– короткі коди лексикографічно передують довшим.

Стиснення Хаффмана складається з 5 єтапів:

– підрахунок частоти символів. Спочатку обчислюються частоти входження кожного символу у вихідних даних. Чим частіше символ зустрічається, тим менше бітів він отримує в коді;

– побудова дерева Хаффмана. Символи із відомими частотами використовуються для побудови дерева Гаффмана. Дерево має дві гілки, де листям є символи, а вузли – це поєднання двох символів з найнижчими частотами. Дерево побудоване так, що символи з більшими частотами знаходяться ближче до кореня дерева [9];

– кодування символів. Символам присвоюються бітові коди, причому короткі коди використовуються для символів з високими частотами, а довгі коди – для символів з низькими частотами. Це забезпечує, що часті символи кодуються короткими бітовими послідовностями, що допомагає зменшити обсяг даних;

– кодування даних. Дані кодуються за допомогою отриманих бітових кодів для кожного символу. Замість використання фіксованої кількості бітів на символ, як це робиться у фіксованих кодуваннях, кожний символ кодується за допомогою його власного унікального коду;

– розкодування даних. Для розкодування даних використовується побудоване дерево Гаффмана. Декодер вибирає символи, спускаючись вниз по дереву, відповідно до бітових кодів у вхідних даних.

Сприйняття Хаффмана виглядає наступним чином: часті символи мають короткі коди, що дозволяє зменшити обсяг даних, тоді як рідкі символи мають довгі коди, але це призводить до загального зменшення обсягу даних після стиснення.

Метод стиснення Хаффмана є ефективним для стиснення текстових даних, а також інших видів даних з великими різницями у частоті входження символів. Він використовується в різних стандартах, включаючи формат ZIP, формат JPEG для стиснення зображень та інші.

1.2 Run-Length Encoding

Run-Length Encoding (RLE) – це простий метод стиснення даних, який полягає в заміні послідовних повторюваних символів на один символ, попереджуючи кількість повторень цього символу. RLE використовується для стиснення послідовностей, де деякі символи повторюються багато разів підряд [10].

Є алгоритмом стиснення з втратами та одним з найстаріших і найпростіших алгоритмів архівації графіки. Зображення в ньому витягується в ланцюжок байт по рядках растра. Саме стиснення в RLE відбувається за рахунок того, що в оригінальному документі зустрічаються ланцюжка однакових байт. Заміна їх на пари лічильник повторень, значення зменшує надмірність даних.

В даному алгоритмі ознакою лічильника служать одиниці в двох верхніх бітах ліченого файлу.

Відповідно залишилися 6 біт витрачаються на лічильник, який може приймати значення від 1 до 64. Рядок з 64 повторюваних байтів ми перетворюємо в два байта, тобто стиснемо в 32 рази.

Алгоритм розрахований на ділову графіку – зображення з великими областями повторюваного кольору. Ситуація, коли файл збільшується, для цього простого алгоритму не так вже рідкісна. Її можна легко отримати, застосовуючи групове кодування до оброблених кольоровим фотографіям. Для того, щоб збільшити зображення в два рази, його треба застосувати до зображення, в якому значення всіх пікселів більше довічного «11000000» і поспіль попарно не повторюються.

RLE використовується у різних областях, таких як зображення (наприклад, в форматі BMP), де пікселі одного кольору можуть бути об’єднані в одну групу замість послідовності однакових пікселів. Також він використовується для стиснення текстових даних, де повторення символів можуть бути звичайним явищем.

Основною ідеєю RLE э заміна повторюваних символів: Це значить що коли два або більше однакових символи зустрічаються підряд, вони замінюються одним символом, за яким слідує число, яке вказує на кількість повторень цього символу. Наприклад, «AAAABBBCCDAA» може бути закодовано як «4A3B2C1D2A». Символи, які не повторюються, залишаються без змін.

З цього випливає що даний метод RLE є ефективним при повторах, тобто, коли вхідні дані містять багато повторюваних символів, які послідовно повторюються. Справедливе й навпаки – незручність для даних з низькою повторюваністю. Для даних з низькою частотою повторення RLE може збільшити розмір даних через додавання чисел після кожного символу.

Хоча RLE є дуже простим методом стиснення, він не завжди ефективний для всіх типів даних. У деяких випадках інші методи стиснення, такі як DEFLATE або Huffman Coding, можуть забезпечити кращу стискальність.

1.3 AES шифрування

Advanced Encryption Standard (AES) – це симетричний алгоритм шифрування, який використовується для захисту конфіденційності даних. Він був прийнятий Національним інститутом стандартів і технологій (NIST) у 2001 році як заміна застарілому алгоритму DES (Data Encryption Standard) [11].

Так як це симетричний алгоритм AES використовує один і той же ключ для як шифрування, так і розшифрування даних. Це означає, що ключ потрібно надавати обом сторонам, які спілкуються (рис. 1.6).

Рисунок 1.6 – Симетричне шифрування

Це блочний алгоритм, тому дані розбиваються на блоки фіксованого розміру перед шифруванням. AES підтримує різні розміри блоків, але найпоширеніші – 128 біт. Також AES підтримує ключі різної довжини,

включаючи 128, 192 і 256 біт. Довший ключ зазвичай забезпечує більший рівень безпеки, але вимагає більше обчислювальних ресурсів. AES використовує складні математичні операції, такі як субституція і перестановка байтів, для змішування даних перед шифруванням.

Шифрування AES включає в себе кілька раундів, під час яких застосовуються різні операції для забезпечення високого рівня безпеки.

AES став стандартом для шифрування даних у багатьох галузях, включаючи інтернет-передачу даних, зберігання на дисках, захист безпеки мережі Wi-Fi (WPA2), захист фінансових транзакцій і багато іншого. AES є відкритим стандартом, що означає, що його можна використовувати вільно для різних цілей. Інформація про нього загальнодоступна і перевірена безліччю експертів у галузі криптографії.

Він включає в себе велику кількість математичних операцій і формул, які важко уявити у вигляді окремих формул. Однак основні кроки шифрування і розшифрування можна представити в загальному вигляді.

Основні кроки AES-шифрування: додавання ключа, повторення раундів, підставлення байтів, перемішування рядків, змішування стовпців, додавання раундового ключа, останній раунд (рис. 1.7).

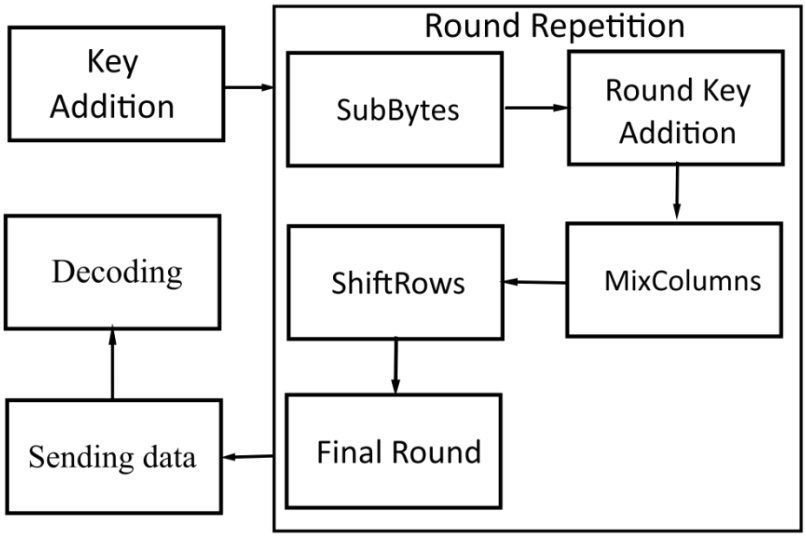

Рисунок 1.7 – Схема методу AES-шифрування

Кроки AES-шифрування:

– додавання ключа (Key Addition). Оригінальні дані комбінуються з ключем за допомогою операції XOR;

– повторення раундів (Round Repetition). Основний процес шифрування повторюється декілька разів (зазвичай 10 для AES-128, 12 для AES-192 і 14 для AES-256), і кожен раунд включає в себе такі операції [12];

– підставлення байтів (SubBytes). Застосовує заміну байтів з таблиці заміни (S-Box);

– перемішування рядків (ShiftRows). Переміщення байтів у кожному рядку;

– змішування стовпців (MixColumns). Лінійне перетворення стовпців матриці байтів;

– додавання раундового ключа (Round Key Addition). Кожен раунд використовує свій унікальний раундовий ключ;

– останній раунд (Final Round). Останній раунд не включає MixColumns;

– завершення. Зашифровані дані готові до відправлення або зберігання;

– розшифрування. Розшифрування виконується у зворотньому порядку, починаючи з останнього раунду і закінчуючи першим.

1.4 Постановка задачі дослідження

Таким чином, задача полягає у розробці застосунку для контролю результатів технічних іспитів у виді чисельної інформації, шифрування та архівування. алгоритму шифрування чисельних значень з використанням методів криптографії та аналізу сигналів.

Об’єктом дослідження є масив числових даних.

Метою дослідження є розробка методів, що базуються на використанні методів шифрування, які дозволяють зменшити обсяг даних та захистити їх від зловмисників.

Для досягнення цієї мети необхідно вирішити наступні завдання:

– аналіз існуючих методів шифрування чисельних даних: Дослідити існуючі методи та алгоритми шифрування чисел з метою вибору найбільш підходящого для конкретного використання;

– розробити алгоритм, який забезпечує конфіденційність чисельних значень, тобто перетворення даних в нечитабельну форму, яку можна розшифрувати лише з використанням певного ключа;

– реалізація алгоритму шифрування: Реалізувати розроблений алгоритм шифрування в програмному коді для подальшого використання;

– забезпечити безпеку ключів, розробити методи для збереження та керування ключами шифрування, оскільки вони є важливою частиною шифрування;

– провести тестування розробленого алгоритму шифрування для забезпечення його ефективності та безпеки.

2 МАТЕМАТИЧНІ МОДЕЛІ МЕТОДІВ ШИФРУВАННЯ

2.1 Математична модель методу шифрування AES

2.1.1 Основи шифру

$$m(x) = x^8 + x^4 + x^3 + x + 1. \quad (2.1)$$

$$b_7x^7 + b_6x^6 + b_5x^{5+}b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0. \quad (2.2)$$

$$b(x) = b_7x^7 + b_6x^6 + b_5x^{5+}b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0. \quad (2.3)$$

Для опису алгоритму використовується кінцеве поле Галуа $GF(2^8)$, побудоване як розширення поля $GF(2) = \{0,1\}$ за модулем многочлена (2.1). Елементами поля $GF(2^8)$ є многочлени виду (2.2), Ступінь яких менший за 8, а коефіцієнти $b^7, b^6,\ldots,b^0 \epsilon\{0, 1\}$ [13].

Операції у полі виконуються за модулем m(x). Всього в полі $GF(2^8)$ налічується $2^8 = 256$ многочленів. Подання двійкового числа b^7, b^6, b^5, b^4, b^3, b^2, b^1, b^0 у вигляді многочлена з коефіцієнтами b^7, $b^6,\ldots,b^0$ дозволяє інтерпретувати байт як бітовий многочлен у кінцевому полі $GF(2^8)$ (2.3).

Наприклад, байт 63 задає послідовність бітів 01100011 та визначає конкретний елемент поля $x^6 + x^5 + x + 1 \leftrightarrow 01100011$. Розглянемо основні математичні операції у полі $GF(2^8)$.

Складання байт можна виконати будь-яким із трьох способів: подати байти бітовими многочленами та скласти їх за звичайним правилом підсумовування многочленів з наступним приведенням коефіцієнтів суми по модулю 2 (операція XOR над коефіцієнтами); підсумовувати за модулем 2 відповідні біти в байтах; скласти байти у шістнадцятковій системі обчислення.

$$(x^6+x^4+x^2+x+1)+(x^7+x+1)=x^7+x^6+x^4+x^2. \quad (2.4)$$

$$\{01010111\}+\{10000011\}=\{11010100\}. \quad (2.5)$$

$$\{57\}+\{83\}=\{D4\}. \quad (2.6)$$

Наприклад, такі три записи еквівалентні [14]:

– подання у вигляді многочленів (2.4);

– бітове подання (2.5);

– шістнадцяткове подання (2.6).

Множення байт виконується за допомогою представлення їх многочленами та перемноження за звичайними правилами алгебри.

Результат приведення дорівнює залишку від поділу твору на *m(x).*

$$m(x) = x8+x4+x3+x+1. \quad (2.7)$$

Отриманий результат необхідно навести за модулем многочлена (2.7)

$$x(b_7x^7+b_6x^{6+}b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x+b_0)=b_7x^{8+}b_6x^7+$$
$$+b_5x^6+b_4x^5+b_3x^4+b_2x^3+b_1x^2+b_0. \quad (2.8)$$

Перемноження многочленів у полі можна спростити, ввівши операцію множення бітового многочлена (2.2) на *x* (2.8).

Для будь-якого ненульового бітового многочлена *b(x)* у полі *GF(2⁸)* існує многочлен $b^{-1}(x)$ зворотний щодо нього по множенню $b(x)b^{-1}(x)=1 mod\ m(x)$. Для знаходження зворотного елемента використовують розширений алгоритм Евкліда, за допомогою якого знаходять такі многочлени *a(x)* і *c(x)*, що $a(x)b(x)+c(x)m(x)=1$. Отже, $b^{-1}(x) \equiv a(x) mod m(x)$.

Многочлени з коефіцієнтами, що належать полю *GF(2⁸)*. Многочлени третього ступеня з коефіцієнтами кінцевого поля a і є *GF(2⁸)*.

$$a(x) = a_3x^4 a_2x^2 a_1x + a_0. \quad (2.9)$$

Таким чином, у цих многочленах у ролі коефіцієнтів при невідомі задіяні байти замість біт. Далі многочлени (2.9) представлятимемо у формі слова [a_0, a_1, a_2, a_3]. У стандарті AES при множенні многочленів виду використовується приведення за модулем іншого многочлена $x^4 + 1$.

Для вивчення арифметики аналізованих многочленів введемо додатково многочлен $b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$, де $b_i \in GF(2^8)$.

Тоді:

$$a(x) + b(x) = (a_3 \oplus b_3)x^3 + (a_2 \oplus b_2)x^2 + (a_1 \oplus b_1)x + (a_0 \oplus b_0). \quad (2.10)$$

– додавання

$$a(x) = a^3x^3 + a^2x^2 + a^1x + a^0\ b,$$
$$b(x) = b^3x^3 + b^2x^2 + b^1x + b^0; \quad (2.11)$$

– множення. Це складніша операція. Нехай, перемножується два многочлена (2.11).

$$a(x)b(x) = c^6x^6 + c^5x^5 + c^4x^4 + c^3x^3 + c^2x^2 + c^1x + c^1. \quad (2.12)$$

Результатом множення буде многочлен (2.12), де:

$$c_0 = a_0b_0;$$

$$c_1 = a_1b_0 \oplus a_0b_1;$$

$$c_2=a_2b_0 \oplus a_1b_1 \oplus a_0b_2;$$

$$c_3=a_3b_0 \oplus a_2b1 \oplus a_0b_3;$$

$$c_4=a_3b_1 \oplus a_2b_2 \oplus a_1b_3;$$

$$c_5=a_3b_2 \oplus a_2b_3;$$

$$c_6=a_3b_3.$$

2.1.2 Раундове перетворення AES

Операція SubBytes виконує нелінійну заміну байтів, що виконується незалежно з кожним байтом матриці State. Заміна оборотна і побудована шляхом комбінації двох перетворень над вхідним байтом:

– знаходження зворотного (інвертованого) елемента щодо множення в полі $GF(2^8)$ (вважається, що нульовий байт {00} переходить сам у себе);

– виконання якогось афінного перетворення: множення інвертованого байта на многочлен $a(x)=x^4+x^3+x^2+x+1$ і підсумовування з многочленом $b(x) = x^6 +x^5+x+1$ у полі $F_2[x]/x^8+1$;

– $a^{-1}(x) = x^6+x^3+x$ та $a^{-1}(x)b(x) = x^2 + 1$ у матричній формі процедура SubBytes записується як на рисунку 2.1.

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}^{-1} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix},$$

Рисунок 2.1 – SubBytes у матричній формі

Де через x позначені вхідні біти, а через *y* – вихідні. Якщо на вхід функції потрапляє нульовий байт, то результатом заміни буде число $y = b$. Процес заміни байтів за допомогою таблиці підстановки ілюструє рисунок 2.2. Нелінійність перетворення обумовлена нелінійністю інверсії x^{-1}, а оборотність – оборотністю матриці [15].

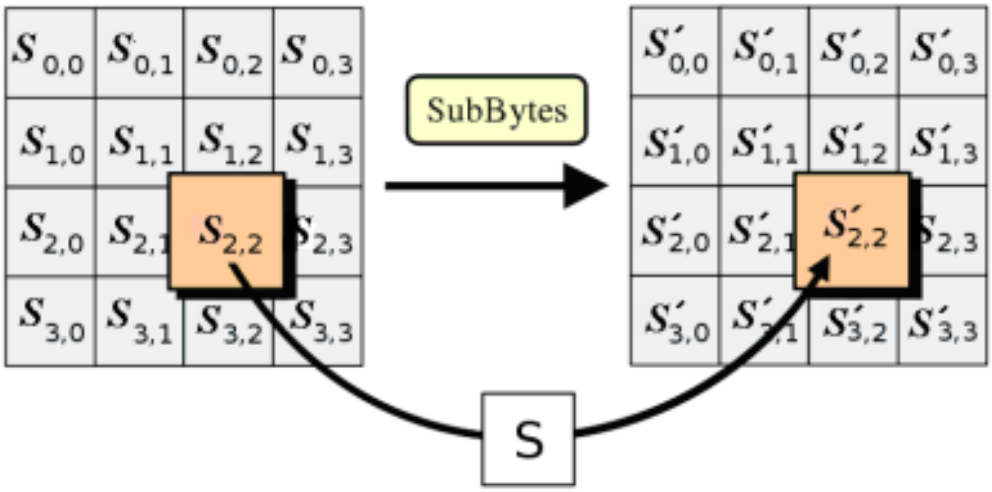

Рисунок 2.2 – Процес заміни байтів

Створену на основі цієї операції спеціальну таблицю замін байтів у шістнадцятковій системі називають *S*-боксом (табл. 2.1).

Наприклад, якщо $s_{1,1}$={8A}, то результат заміни цього байта слід шукати на перетині рядка з індексом 8 та стовпця з індексом *A*, тобто *SubBytes(8A)={7E}*.

SubByte є важливим елементом забезпечення конфіденційності та стійкості AES. Цей крок також включає в себе інші операції, такі як зсуви байтів, міксовані константи та додавання ключа, що додає додатковий рівень безпеки до шифру. У великому контексті ці кроки працюють разом для забезпечення ефективного та безпечного шифрування даних.

Таблиця 2.1 – Таблиця заміни байтів

	Таблиця перетворень SubBytes															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	FO	AD	D4	A2	AF	9C	A4	72	CO
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	AO	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	DO	EF	AA	FB	43	4D	33	85	45	F9	02	F7	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DE
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	CB	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	IF	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	OE	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	OD	BF	E6	42	68	41	99	2D	OF	BO	54	BB	16

2.1.3 Операція ShiftRows

Операція застосовується до рядків матриці State (рис. 2.3) – її перша рядок нерухомий, а елементи нижніх трьох рядків циклічно зсуваються вправо на 1, 2 і 3 байти відповідно.

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} \rightarrow \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \end{pmatrix}.$$

Рисунок 2.3 – Матриця State

По суті, це перестановка елементів матриці, в якій беруть участь лише елементи рядків, тому перетворення (рис. 2.4) оборотне [16].

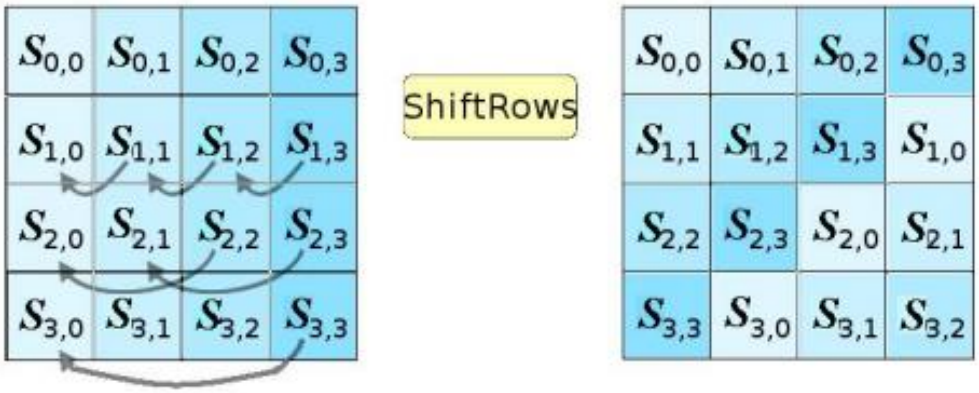

Рисунок 2.4 – Перестановка матриці State

2.1.4 Операция MixColumns

За допомогою цієї операції виконується перемішування байтів у стовпцях матриці State (рис. 2.5).

$$c(x) = c_3x^3 + c_2x^2 + cx + c_0 = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}. \quad (2.13)$$

Кожен стовпець цієї матриці приймається за многочлен над полем $GF(2^8)$ і множиться фіксований многочлен (2.13) по модулю многочлена x^4+1.

Як показано вище, таку операцію можна записати в матричному вигляді як на рисунку 2.6.

$$\begin{pmatrix} c_0 & c_3 & c_2 & c_1 \\ c_1 & c_0 & c_3 & c_2 \\ c_2 & c_1 & c_0 & c_3 \\ c_3 & c_2 & c_1 & c_0 \end{pmatrix} \cdot \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix} = \begin{pmatrix} s_0' \\ s_1' \\ s_2' \\ s_3' \end{pmatrix}$$

Рисунок 2.5 – Операція MixColumns

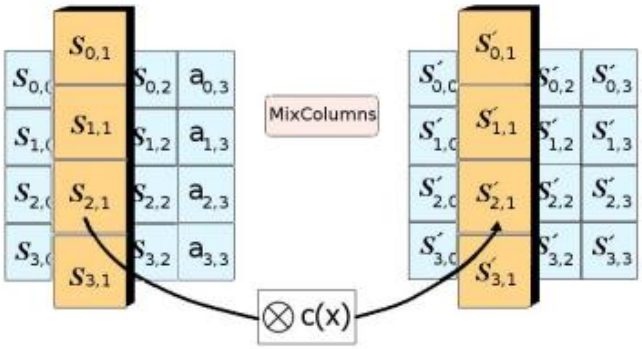

Рисунок 2.6 – Перемішування стовбців матриці State

Многочлен *c(x)* – взаємно простий із многочленом x^4+1 над полем *GF(2)*, тому в полі існує зворотний многочлен $c_{-1}(x)(mod x^4+1)$ матриця в цій формулі оборотна.

Функція AddRoundKey (State, RoundKey) (рис. 2.7) побітово складає елементи змінної RoundKey та елементи змінної State за принципом: *i*-й стовпець даних *(i = 0,1,2,3)* складається з певним 4-байтовим фрагментом розширеного ключа $W[x^4+1]$, де *r* – номер поточного раунду алгоритму. Під час шифрування перше складання ключа раунду відбувається до першого виконання операції SubBytes.

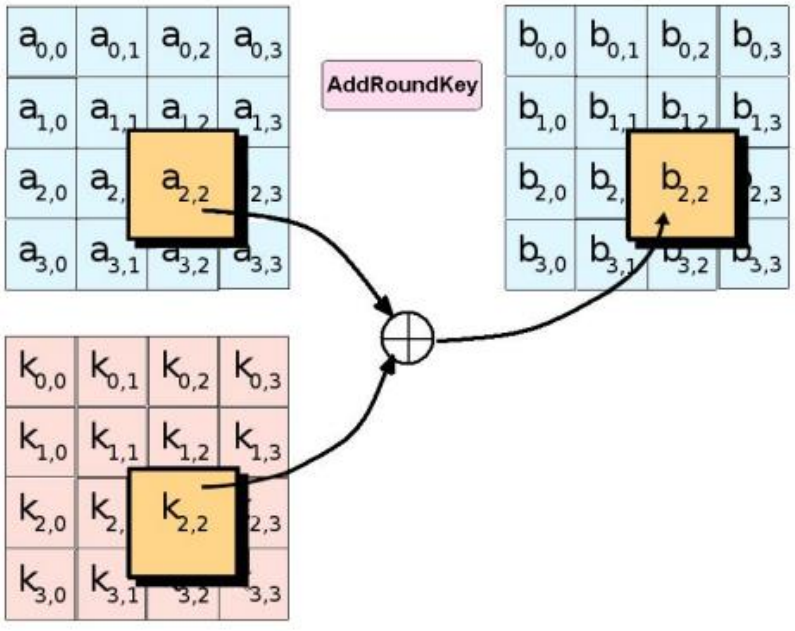

Рисунок 2.7 – Додавання раундового ключа

Рисунок 2.8 демонструє властивості розсіювання та перемішування інформації в ході шифрування алгоритмом AES. Видно, що два раунди забезпечують повне розсіювання і перемішування інформації. Досягається це за рахунок рахунок використання функцій ShiftRows і MixColumns. Операція SubBytes надає шифруванню стійкість проти диференціального криптоаналізу, а операція AddRoundKey забезпечує необхідну секретну випадковість.

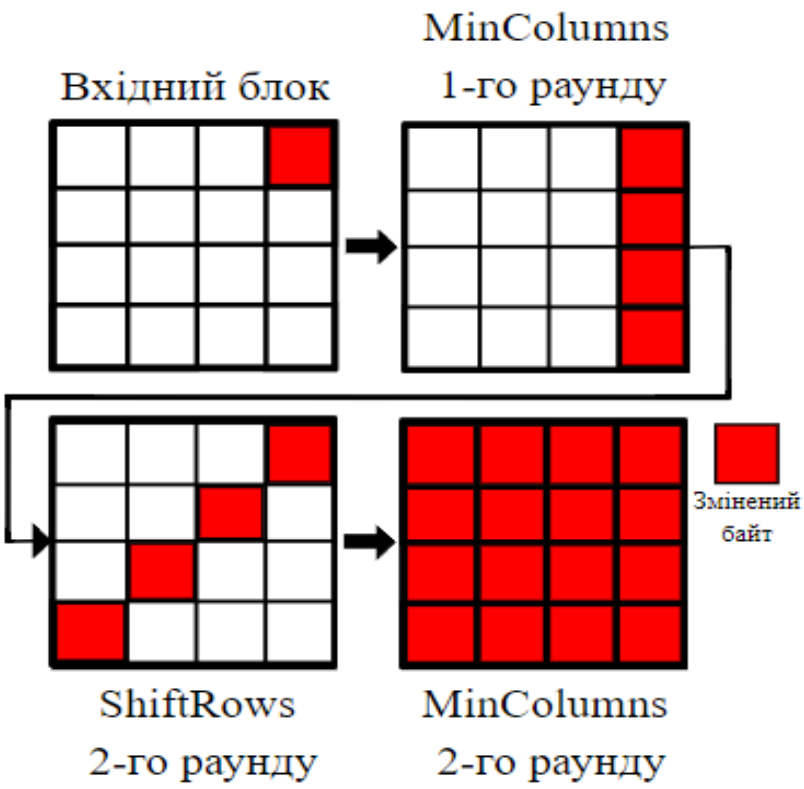

Рисунок 2.8 – Додавання раундового ключа

2.1.5 Ключовий розклад AES

Раундові ключі виробляються із ключа шифру K за допомогою процедури розширення ключа, внаслідок чого формується масив раундових ключів, з якого потім безпосередньо вибирається потрібний раундовий ключ. Кожен раундовий ключ має довжину 128 біт (або 4 чотирибайтові слова w_i,

w_{i+1}, w_{i+2}, w_{i+3}, а довжина в бітах всіх раундових ключів дорівнює 128 біт (10 раундів +1) = 1408 біт (або 44 чотирибайтові слова w_0, w_1, w_2,…,w_{42}, w_{43}). Перші чотири слова w_0, w_1, w_2, w_3 у ключовому масиві заповнені ключем шифру, решти вироблених 40 слів вибираються по 4 слова для ключа раунду. Вибір слів простий: перші чотири слова (вони збігаються з ключом шифру) є ключем з номером 0, наступні чотири слова w_4, w_5, w_6, w_7 – раундовим ключем для першого повного раунду.

Нові слова w_{i+4}, w_{i+5}, w_{i+6}, w_{i+7} наступного раундового ключа визначаються зі слів w_i, w_{i+1}, w_{i+2}, w_{i+3} попереднього ключа на основі рівнянь [17]:

$$w_{i+5} = w_{i+4} \oplus w_{i+1};$$

$$w_{i+6} = w_{i+5} \oplus w_{i+2};$$

$$w_{i+7} = w_{i+6} \oplus w_{i+3}.$$

Перше слово w_i+4 у кожному раундовому ключі змінюється по-іншому:

$$w_{i+7} = w_i \oplus g(w_{i+3}).$$

Тут дія функції g зводиться до послідовного виконання трьох кроків, що відображають слово в слово:

– циклічний зсув чотирибайтового слова вліво на один байт (Операція RotWord);

– заміна кожного байта слова, отриманого на попередньому кроці, в відповідно до таблиці SubBytes, що використовується при шифрування (операція SubWord);

– підсумовування по *mod2* байтів, отриманих на попередньому кроці, з раундової постійної *Rcon[i] = (RC[i],0,0,0)*, несекретної та унікальною для

кожного раундового ключа K_i. Три самі праві байти цієї константи – нульові, а ненульовий лівий байт змінюється за відомим законом рекурсії: *RC[1]=1*, *RC [i]=2, RC [i – 1], i = 1,2,…,10*.

Мета підсумовування з раундовими константами – зруйнувати будь-яку симетрію, що може виникнути на різних етапах розгортання ключа і призвести до появи слабких ключів, як у алгоритмі DES [18]. Робота алгоритму розширення ключа продемонстрована на рисунку 2.9.

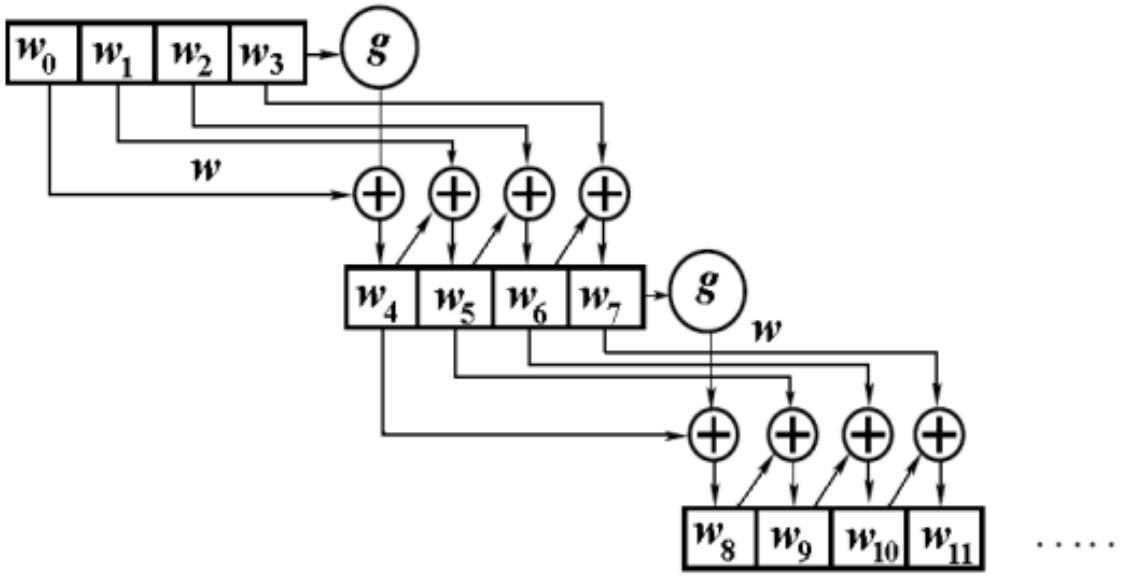

Рисунок 2.9 – Додавання раундового ключа

2.1.6 Розшифрування AES

Для розшифрування шифротексту всі використовувані шифруючі перетворення можуть бути інвертовані та застосовані в зворотному порядку. Перед першим раундом дешифрування виконується операція AddRoundKey, що накладає на шифротекст чотири Останні слова розширеного ключа. Потім виконується 10 раундів дешифрування, кожен з яких здійснює такі операції:

– операція InvShiftRows, зворотна операція ShiftRows. Байти в останніх трьох рядках матриці State циклічно зсуваються вліво на різну кількість байт.

Перший рядок нерухомий, а нижні три рядки зсуваються вліво на 1, 2 і 3 байти відповідно;

– операція InvSubBytes, зворотна операція SubBytes. Байти матриці State замінюються новими значеннями за наведеною нижче таблиці (табл. 2.2) зворотної заміни, що є інвертованим S-боксом [19].

Таблиця 2.2 – Таблиця зворотньої заміни байтів

	Таблиця перетворювань InvSubBytes															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DE	6E
A	47	E1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	D	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7E	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	CB	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

$$c^{-1}(x) = \{0B\}x^3 + \{0D\}x^2 + \{09\}x + \{0E\}; \qquad (2.14)$$

– операція InvMixColumns – процедура, зворотна до процедури MixColumns. Кожен стовпець матриці State розглядається як чотиричленний многочлен над полем $GF(2^8)$ і множиться на фіксований многочлен (2.14) о модулю многочлена x^4+1.

Таку операцію множення матриці на многочлен можна записати у матричному виді (рис. 2.10);

$$\begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \cdot \begin{pmatrix} s_0' \\ s_1' \\ s_2' \\ s_3' \end{pmatrix} = \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

Рисунок 2.10 – Операція InvMixColumns у матричному виді

– операція AddRoundKey є зворотною сама до себе, так як полягає лише у підсумовуванні за *mod 2*. Останній раунд розшифрування не містить операції InvMixColumns. У алгоритмі розшифрування послідовність перетворень відрізняється від порядку операцій шифрування, а алгоритм розширення ключа залишається незмінним. Однак два властивості алгоритму AES дозволяють побудувати іншу еквівалентну процедуру розшифрування, де послідовність операцій перетворення залишається тією самою (природно із заміною операцій на зворотні);

– комутативність операцій SubBytes та ShiftRows;

– лінійність операцій перемішування у стовпці MixColumns та InvMixColumns по відношенню до даних стовпця, що означає InvMixColumns(State ⊕ RoundKey) = InvMixColumns(State) ⊕ InvMixColumns(RoundKey).

Це дає можливість інвертувати порядок виконання процедур InvSubBytes та InvShiftRows. Якщо ж додатково в послідовності раундових ключів змінити слова за допомогою процедури InvMixColumns (не чіпаючи перші та останні чотири слова), то порядок виконання процедур AddRoundKey та InvMixColumns також можна змінити на зворотний.

2.2 Run Length Encoding

2.2.1 Загальні характеристики RLE

Кодування довжини циклу (RLE) – це техніка, яка не дуже широко використовується в наш час, але це чудовий спосіб зрозуміти деякі проблеми, пов’язані з використанням стиснення [20].

Наприклад, таке просте чорно-біле зображення на рисунку 2.11.

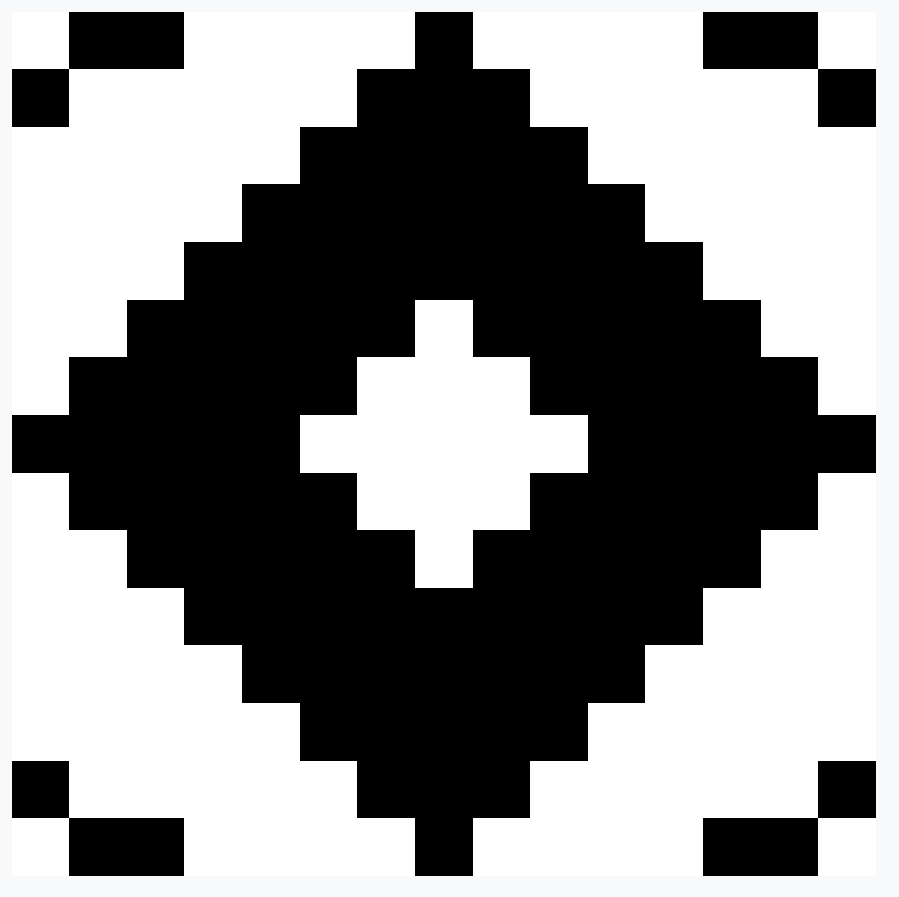

Рисунок 2.11 – Просте зображення ромбу у пікселях

Один з дуже простих способів зберігання цього зображення [19] у двійковому форматі – це використання формату, де «0» означає білий колір, а «1» – чорний (це «растрове зображення», тому пікселі зіставлені зі значеннями бітів). Використовуючи цей метод, наведене вище зображення виглядатиме як на рисунку 2.12.

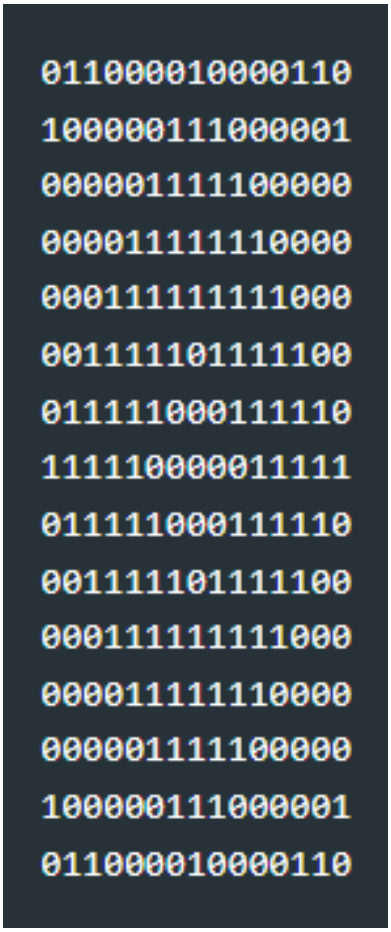

Рисунок 2.12 – Зображення у війковому форматі

2.2.2 Приклад роботи RLE

Існує формат зображень, який використовує просте представлення з одним символом на піксель, яке ми щойно описали. Цей формат називається «портативний формат растрових зображень» (PBM) [21]. Файли PBM зберігаються з розширенням «.pbm» і містять простий заголовок, за яким слідують дані зображення. Дані у файлі можна переглянути, відкривши його в текстовому редакторі, подібно до відкриття файлу .txt, а саме зображення можна переглянути, відкривши його в програмі для малювання або перегляду зображень, яка підтримує файли PBM (формат не дуже добре підтримується, але ряд програм для перегляду і редагування зображень можуть їх відображати). Файл PBM для зображення ромбу, який був використаний раніше, має вигляд як на рисунку 2.13.

```
P1
15 15
0 1 1 0 0 0 0 1 0 0 0 0 1 1 0
1 0 0 0 0 0 1 1 1 0 0 0 0 0 1
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 1 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 0 0 0
0 0 1 1 1 1 1 0 1 1 1 1 1 0 0
0 1 1 1 1 1 0 0 0 1 1 1 1 1 0
1 1 1 1 1 0 0 0 0 0 1 1 1 1 1
0 1 1 1 1 1 0 0 0 1 1 1 1 1 0
0 0 1 1 1 1 1 0 1 1 1 1 1 0 0
0 0 0 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0
1 0 0 0 0 0 1 1 1 0 0 0 0 0 1
0 1 1 0 0 0 0 1 0 0 0 0 1 1 0
```

Рисунок 2.13 – Зображення у форматі PBM

Перші два рядки є заголовком. Перший рядок визначає формат файлу (P1 означає, що файл містить ASCII нулі та одиниці) [22]. Другий рядок визначає ширину, а потім висоту зображення в пікселях. Це дозволяє комп'ютеру знати розмір і розміри зображення, навіть якщо символи нового рядка, що розділяють рядки у файлі, були відсутні. Решта даних – це зображення, як показано вище. Якщо на вашому комп'ютері є програма, яка може відкривати PBM-файли, ви можете переглянути зображення за допомогою неї. Можна навіть написати програму для виведення цих файлів, а потім відобразити їх як зображення.

Оскільки у цьому форматі цифри представлені за допомогою ASCII [23], він не дуже ефективний, але корисний, якщо ви хочете прочитати те, що знаходиться всередині файлу. Існують варіації цього формату, які пакують пікселі в біти замість символів, а також варіації, які можна використовувати для відтінків сірого і кольорових зображень.

Ключове питання стиснення полягає в тому, чи можна представити те саме зображення, використовуючи меншу кількість бітів, але при цьому мати змогу відновити оригінальне зображення [24].

Припустимо, у нас є великий масив з однаковими значеннями (рис. 2.14). Тож треба буде записувати, наприклад, багато нулів. Було б набагато краще записати «п’ять нулів» замість «нуль-нуль-нуль-нуль-нуль-нуль-нуль-нуль». Це основна ідея кодування RLE, яке використовується для економії місця при зберіганні цифрових зображень [25]. У кодуванні довжини серії змінюється кожен рядок числами, які вказують, скільки пікселів підряд мають однаковий колір, завжди починаючи з кількості білих пікселів. Наприклад, перший рядок на зображенні вище містить один білий, два чорних, чотири білих, один чорний, чотири білих, два чорних і один білий пікселі. Це можна представити як на рисунку 2.14 та рисунку 2.15.

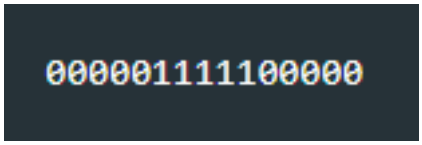

Рисунок 2.14 – Перший рядок зображення у бінарному виді

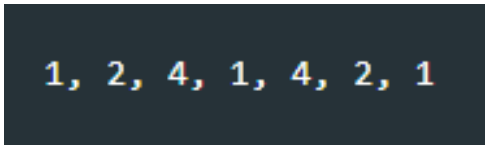

Рисунок 2.15 – Кодування первого рядка

Для другого рядка, оскільки потрібно вказати кількість білих пікселів до того, як зада кількість чорних, потрібно явно вказати, що на початку рядка є нуль (рис. 2.16).

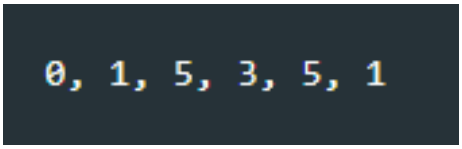

Рисунок 2.16 –Кодування другого рядка

Причина того, що на початку другого рядка нуль в тому, що якби у нас не було чіткого правила, з чого починати, комп'ютер не зміг би зрозуміти, який колір є яким, коли він відображатиме зображення, представлене в такому вигляді [26].

Основне місце використання чорно-білих відсканованих зображень зараз – це факсимільні апарати, які використовують цей підхід до стиснення. Однією з причин, чому він так добре працює зі сканованими сторінками, є те, що кількість білих пікселів, що йдуть підряд, величезна. Фактично, будуть цілі відскановані рядки, які є нічим іншим, як білими пікселями [27]. Типова факсимільна сторінка має 200 пікселів у поперечнику або більше, тому заміна 200 біт одним числом – це велика економія. Саме число може займати кілька бітів, і в деяких місцях на відсканованій сторінці лише кілька послідовних пікселів замінюються на число, але в цілому економія є значною. Фактично, факсимільні апарати надсилали б сторінки в 7 разів довше, якби не використовували стиснення.

3 КОМП'ЮТЕРНА МОДЕЛЬ ФІЛЬТРАЦІЇ ЗОБРАЖЕНЬ

3.1 Обґрунтування вибору середовища програмної реалізації

У рамках кваліфікаційної роботи був розроблений десктоп застосунок контролю та архівації результатів іспитів з допомогою сучасних методів шифрування. Для реалізації було обране середовище Visual Studio 2019. Це обумовлено тим, що Visual Studio працює з великою кількістю мов програмування, у цьому випадку мовою C#, має підтримку Microsoft та має багато вбудований бібліотек, які полегшують роботу.

Мова C# була обрана через те що вона має ряд переваг:

– C# є високорівневою мовою програмування, що дозволяє розробникам виражати ідеї більш чітко та зрозуміло, зменшуючи кількість коду, який потрібно написати;

– є велика кількість готових бібліотек та фреймворків для десктоп розробки, що полегшує роботу;

– C# використовується в середовищі .NET, яке забезпечує доступ до різних функціональних можливостей, таких як робота з базами даних, мережами, графічними інтерфейсами, та іншими;

– для розробки графічного інтерфейсу в C# використовується технологія Windows Presentation Foundation (WPF), яка надає багатий функціонал та дозволяє створювати сучасні та привабливі інтерфейси;

– для десктоп розробки C# часто використовується під платформу Windows, але існують також можливості розробки мультиплатформенних застосунків, таких як використання Xamarin для мобільних платформ;

– C# працює в середовищі .NET, яке надає управління пам'яттю та інші безпекові переваги, зменшуючи можливість виникнення помилок у програмному коді;

– спільнота C# та .NET дуже активна, що означає наявність багатофункціональних бібліотек, готових рішень та ресурсів для розробників;

– C# може бути використаний в поєднанні з іншими мовами програмування та технологіями, що полегшує інтеграцію з різними системами.

Visual Studio, інтегроване середовище розробки (IDE) від Microsoft, вражає своєю унікальністю та зручністю. Це потужне інструментарію для створення різноманітних програм та застосунків. Перше, що робить Visual Studio унікальним, це його інтегрований підхід до розробки, який об'єднує редактор коду, відлагоджувач, та візуальний дизайнер в одному вікні. Додайте до цього масу інструментів для аналізу коду, контролю версій та автоматизації рутинних завдань.

Visual Studio надає широкі можливості для розробки різноманітних застосунків, включаючи вебзастосункитки, настільні програми, мобільні застосунки та ігри. Воно підтримує різні мови програмування, такі як C#, Visual Basic, C++, F#, Python, та інші.

Visual Studio добре інтегроване з платформою .NET, що дозволяє легко створювати програми для Windows, вебзастосунки, служби та інше. Ви можете використовувати мови програмування, такі як C# або Visual Basic, для розробки застосунків для .NET Framework чи .NET Core.

Visual Studio має широкий набір інструментів для розробки, таких як редактор коду з підсвічуванням синтаксису, відладка, вбудовані засоби тестування, дизайнер інтерфейсів користувача, та багато іншого (рис. 3.1).

Visual Studio надає потужні засоби відлагодження, включаючи можливість крокування по коду, аналіз значень змінних, встановлення точок зупинки, та інше. Це полегшує виявлення та виправлення помилок у програмному коді. Також Visual Studio підтримує різні системи контролю версій, такі як Git, що дозволяє командам розробників спільно працювати над проектами.

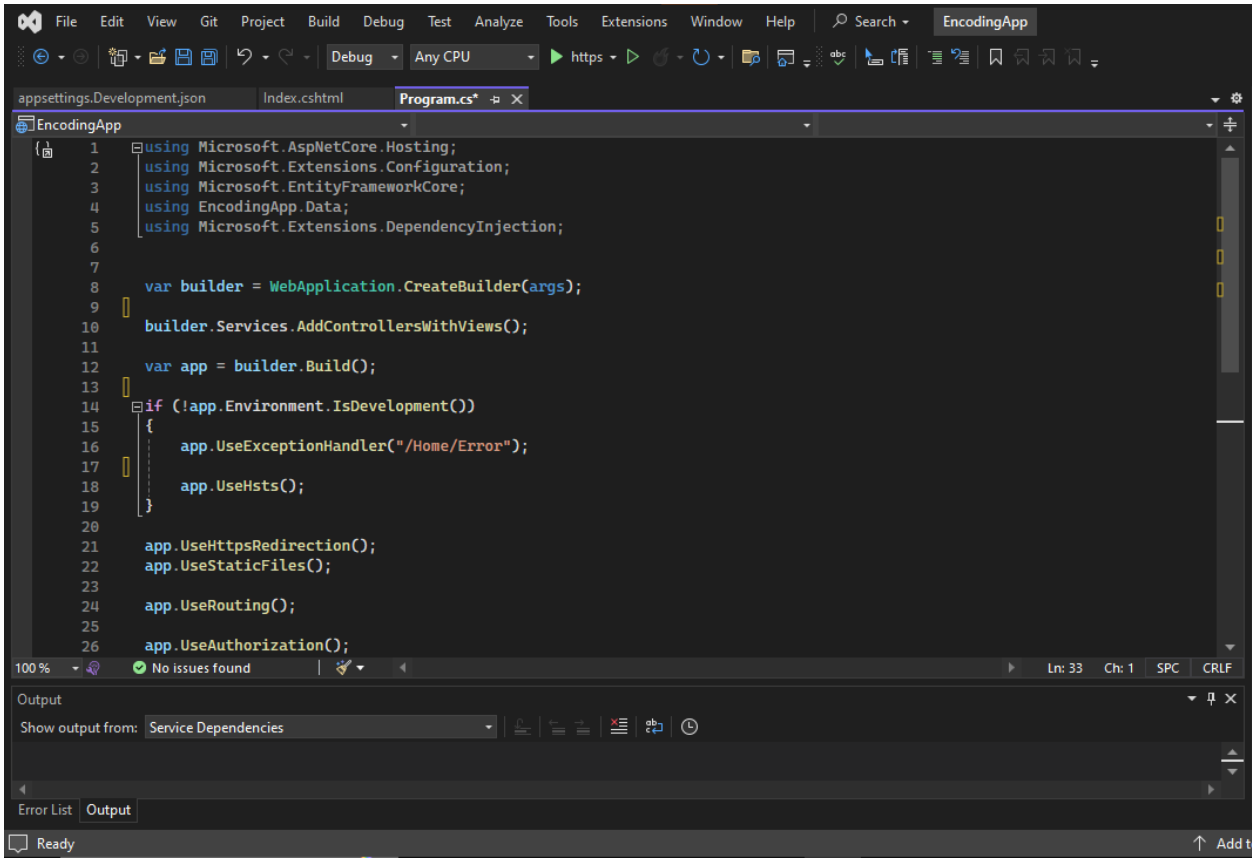

Рисунок 3.1 – Приклад інтерфейсу Visual Studio

3.2 Програмна реалізація

Програмна реалізація базується на основі існуючої бази даних (рис. 3.2) системи автоматичного контролю та архівації іспитів (САКАРІ).

Наповнення бази даних включає у собі результати еталонних вимірювань та фактичні значення які отримали шляхом іспитів безпосередньо.

Так як це система автоматичного контролю повинно бути реалізовано зчитування текстових файлів та додавання їх до бази даних з метою подальшої обробки та навпаки. Фінальна таблиця буде архівована та записана у відповідну директорію.

Дані, що зберігаються, повинні бути захищенні від крадіжок, тож на етапі занесення вони будуть шифруватись з використанням надійного методу шифрування Advanced Encryption Standard (AES), застосування якого

дозволяє забезпечити конфіденційність та безпеку інформації, зберігаючи її в зашифрованому виді у базі даних [28].

	Id	Tact	Time	FactTact	FactTime	Admission
	6	0	0	0	0	True
	7	224	8,96	224	8,96	True
	8	2921	116,84	2925	119,84	False
	9	2951	118,04	2956	118,05	False
	10	3061	122,44	3062	122,43	True
	11	6063	242,52	6065	245,52	True
	12	7041	281,64	7041	281,65	True
	13	7216	288,64	7216	288,64	True
	14	7255	290,2	7255	290,5	True
	15	7260	290,4	7260	290,6	True
	16	18508	740,32	18505	740,42	True
	17	19111	764,44	19111	764,44	True
	18	19136	765,44	19138	766,44	True

Рисунок 3.2 – Вхідна база даних

3.2.1 Робота з базою даних

Контроль бази даних уявляє собою порівняння фактичних результатів, які були отримані безпосередньо іспитами, з еталонними, очікуваними результатами. Для цих значень існує норма відхилень, якої повинно підтримуватись. Результати які мають відхилення поза нормою будуть записуватись у іншу таблицю з метою повторного відпрацювання та виявлення причин відхилення.

Таким чином база даних (рис. 3.3) буде мати 2 таблиці:

– таблиця значень;

– таблиця значень з відхиленнями та підтвердженням.

У першій таблиці (рис. 3.4) будуть дані, які потрапляють з текстового файлу еталонних значен. Ці дані записуються в іншу таблицю (рис. 3.5) з додаванням фактичних значень, порахованого відхилення та допуску до зберігання.

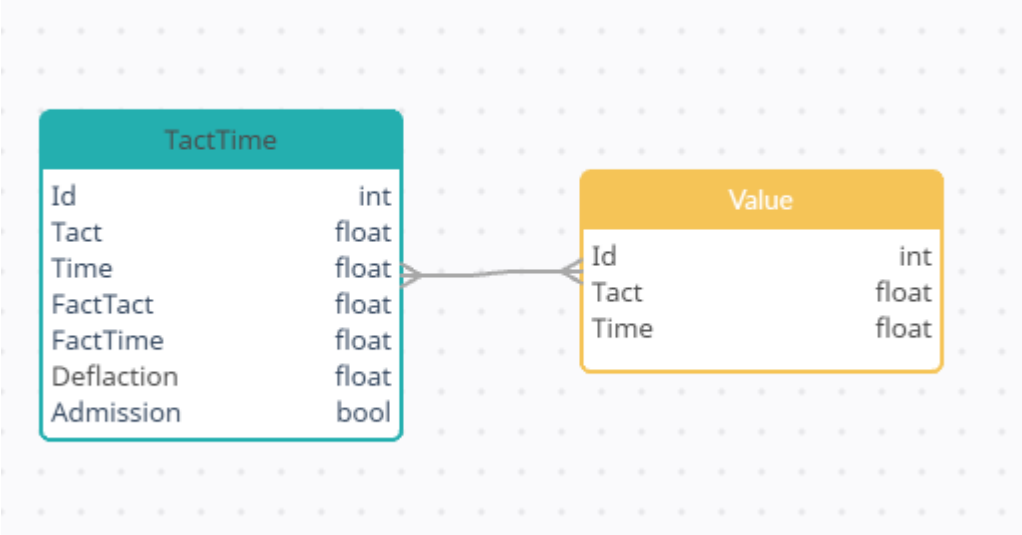

Рисунок 3.3 – Модель бази даних

	Name	Data Type	Allow Nulls	Default
🔑	Id	int	☐	
	Tact	float	☑	
	Time	float	☑	

Рисунок 3.4 – Таблиця Value

	Name	Data Type	Allow Nulls
🔑	Id	int	☐
	Tact	float	☑
	Time	float	☑
	FactTact	float	☑
	FactTime	float	☑
	Admission	bit	☑

Рисунок 3.5 – Таблиця TactEtalon

Для створення таблиць були використані такі запити мовою Structured Query Language (SQL) [29] (рис. 3.6 та рис. 3.7).

```
CREATE TABLE [dbo].[Value] (
    [Id]   INT        IDENTITY (1, 1) NOT NULL,
    [Tact] FLOAT (53) NULL,
    [Time] FLOAT (53) NULL,
    PRIMARY KEY CLUSTERED ([Id] ASC)
);
```

Рисунок 3.6 – Скріпт таблиці Value

```
CREATE TABLE [dbo].[FactEtalon] (
    [Id]        INT        IDENTITY (1, 1) NOT NULL,
    [Tact]      FLOAT (53) NULL,
    [Time]      FLOAT (53) NULL,
    [FactTact]  FLOAT (53) NULL,
    [FactTime]  FLOAT (53) NULL,
    [Admission] BIT        NULL,
    PRIMARY KEY CLUSTERED ([Id] ASC)
);
```

Рисунок 3.7 – Скріпт таблиці FactEtalon

3.2.2 Імпортування даних з файлу

Як вже згадувалось вище інформація з іспитів(еталонні та фактичні значення) надходять у текстових файлах. Для роботи з ними їх треба занести до проекту.

Щоб мати змогу взаємодіяти з базою даних треба встановити підключення, додаванням до файлу конфігурацій коду (рис. 3.8).

```
<add connectionString="Data Source=(LocalDB)\
     MSSQLLocalDB;AttachDbFilename=
     C:\Users\user\source\repos\wap\wap\DatabaseV.mdf;
     Integrated Security=True" name="DBV"/>
```

Рисунок 3.8 – Підключення до бази даних

Також щоб підключення відбувалось при відкритті застосунку додається об’єкт класу SqlConnection у метод загрузки вікна Form_Load (рис. 3.9).

```
sqlConnection = new SqlConnection
    (ConfigurationManager.ConnectionStrings["DBV"].ConnectionString);
sqlConnection.Open();
```

Рисунок 3.9 – Об’єкт класу SqlConnection

В методі button_Click додається об`єкт класу OpenFileDialog для зручного пошуку та вибору файлів на обробку з додаванням фільтру допустимих форматів (рис. 3.10).

```
OpenFileDialog openFileDialog = new OpenFileDialog();
openFileDialog.Filter = "Text Files|*.txt|All Files|*.*";
```

Рисунок 3.10 – Створення браузера файлів

Для того щоб працювати з отриманими даними буде використаний фреймворк Entity Framework.

Entity Framework (EF) – це технологія об’єктно-реляційного відображення для платформи .NET. Вона надає зручний спосіб взаємодії з базою даних за допомогою об’єктно-орієнтованого підходу, де таблиці бази даних представляються класами, а записи – об’єктами цих класів.

Модель даних (Data Model): у EF модель даних визначає структуру бази даних та взаємозв’язки між таблицями у вигляді класів C# або VB.NET.

DbContext: це клас, який представляє контекст бази даних у застосунку. Він взаємодіє з базою даних, дозволяючи зчитувати та записувати дані.

Отримана інформація перетворюється в тип даних, яким можна оперувати за допомогою функціоналу Entity Framework записується у об'єкт класу DataModel та передається до бази даних (рис. 3.11).

```
using (var dbContext = new DbContext())
{
    foreach (string line in lines)
    {
        string[] parts = line.Split(',');
        DataModel dataModel = new DataModel
        {
            Column1 = parts[0],
            Column2 = int.Parse(parts[1]),
        };
        dbContext.DataTable.Add(dataModel);
    }
    dbContext.SaveChanges();
}
```

Рисунок 3.11 – Створення браузера файлів

3.2.3 Шифрування даних

Реалізація AES шифрування в C# може виконуватися за допомогою бібліотеки .NET Framework або .NET Core. Найбільш поширеною бібліотекою для цього є AesManaged з простору імен System.Security.Cryptography.

В даному випадку AES шифрування виконано у режимі CBC (Cipher Block Chaining) з використанням ключа та ініціалізаційного вектора (IV). У реальних застосунках ключі та вектори ініціалізації повинні бути обережно зберігатися та оброблятися, а сам алгоритм шифрування може змінюватися залежно від конкретних потреб та безпекових вимог.

Cipher Block Chaining – це один із режимів робот и для блочних шифрів, таких як AES (Advanced Encryption Standard). У режимі CBC, кожен блок тексту відкритого тексту обробляється шифром, а потім

результат XOR-ується з попереднім блоком шифртексту перед подальшим шифруванням.

У випадку AES, який працює з блоками фіксованого розміру (128 біт, або 16 байт), процес CBC можна описати набором опецарій.

Ініціалізація вектора (IV): створюється випадковий вектор (IV), який використовується для першого блоку.

Шифрування першого блоку:

– відкритий текст першого блоку XOR-ується з IV;

– результат XOR-у шифрується за допомогою AES.

Шифрування і XOR-ування для наступних блоків:

– кожен наступний блок відкритого тексту XOR-ується з шифртекстом попереднього блоку;

– результат XOR-у шифрується за допомогою AES.

Одже треба сформувати ключ (16 байт) та ініціалізійний вектор (16 байт) для роботи з шифруванням. Вони матимуть значення «0123456789ABCDEF» та «FEDCBA9876543210» відповідно.

Ключі, записані у змінні key та IV відповідно (рис. 3.12).

```
string key = "0123456789ABCDEF";
string iv = "FEDCBA9876543210";
```

Рисунок 3.12 – Ключ та ініціалізійний вектор

Створення об’єкта класу AesManaged: створюється об’єкт AesManaged, який буде використовуватися для шифрування.

Створення об’єкта класу Encryptor: з об’єкта AesManaged створюється об’єкт ICryptoTransform, який відповідає за шифрування.

Використання CryptoStream для шифрування: створюється CryptoStream, який об’єднує потік даних для шифрування. Цей потік працює з вхідним потоком даних, які потрібно зашифрувати.

Використання StreamWriter для запису результату: за допомогою StreamWriter записуються зашифровані дані в CryptoStream.

Повернення зашифрованих даних: отримані зашифровані дані повертаються як результат шифрування.

Метод шифрування буде мати наступний вигляд (рис. 3.13).

```
static string Encrypt(string Text, string key, string iv)
{
    using (AesManaged aesAlg = new AesManaged())
    {
        aesAlg.Key = Encoding.UTF8.GetBytes(key);
        aesAlg.IV = Encoding.UTF8.GetBytes(iv);

        ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);

        using (MemoryStream msEncrypt = new MemoryStream())
        {
            using (CryptoStream csEncrypt = new CryptoStream
                (msEncrypt, encryptor, CryptoStreamMode.Write))
            {
                using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))
                {
                    swEncrypt.Write(Text);
                }
            }
            return Convert.ToBase64String(msEncrypt.ToArray());
        }
    }
}
```

Рисунок 3.13 – Метод шифрування

Для прикладу можна провести шифрування такого тексту: «243324 41234123 53214321 432143 1234 214 46 7 653 34523 53245 35 34 3245 435 3245 2345 3245 345 35 3»

Зашифрований текст буде мати такий выгляд (рис. 3.14).

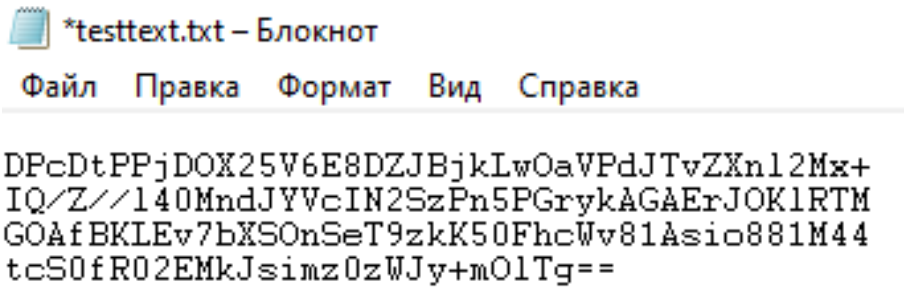

Рисунок 3.14 – Зашифрований текст

3.2.4 Дешифрування даних

Дешифрування даних має виконуватись у порядку, оберненому до шифрування. Щоб це реалізувати має бути доступ до раніше згенерованих ключа та ініціалізійного вектору. Без таких самих ключів, які були використані для шифрування даних не можна буде їх дешифрувати.

Крок дешифрації шифрування AES включає в себе низку операцій, які протилежні крокам шифрування. Один із ключових етапів - це крок «Inverse SubBytes», який використовує обернену табличку заміни (Inverse S-Box) для відновлення оригінальних значень байтів.

Після «Inverse SubBytes» виконується обернене зсування байтів, обернене міксування констант, і обернене додавання ключа. Ці операції відновлюють вихідні дані до їхнього попереднього стану перед шифруванням.

Крок дешифрації також включає в себе обернені версії інших кроків шифрування AES, таких як «Inverse ShiftRows» та «Inverse MixColumns»

Встановлення ключа і вектора ініціалізації (IV): ключ і вектор ініціалізації повинні бути тими самими, які використовувалися під час шифрування. Це дозволить правильно відновити оригінальні дані.

Створення об’єкту AesManaged: використовуючи ключ і вектор ініціалізації, створюється об’єкт AesManaged.

Створення об’єкта Decryptor: з об’єкта AesManaged створюється об’єкт ICryptoTransform, який відповідає за розшифрування.

Використання CryptoStream для дешифрування: створюється CryptoStream, який об’єднує потік даних для дешифрування. Цей потік працює з вхідним потоком зашифрованих даних.

Використання StreamReader для отримання результату: за допомогою StreamReader отримуємо розшифрований текст з CryptoStream.

Метод дешифрування буде мати наступний вигляд (рис. 3.15).

```
static string Decrypt(string cipherText, string key, string iv)
{
    using (AesManaged aesAlg = new AesManaged())
    {
        aesAlg.Key = Encoding.UTF8.GetBytes(key);
        aesAlg.IV = Encoding.UTF8.GetBytes(iv);

        ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV);

        using (MemoryStream msDecrypt = new MemoryStream(Convert.FromBase64String(cipherText)))
        {
            using (CryptoStream csDecrypt = new CryptoStream
                (msDecrypt, decryptor, CryptoStreamMode.Read))
            {
                using (StreamReader srDecrypt = new StreamReader(csDecrypt))
                {
                    return srDecrypt.ReadToEnd();
                }
            }
        }
    }
}
```

Рисунок 3.15 – Метод дешифрування

Для дешифрування був використаний текст, отриманий на етапі шифрування. Додається до текстового файлу дешифрований текст (рис. 3.16).

```
string decryptedText = Decrypt(encryptedText, key, iv);
WriteToFile(filePath, decryptedText);
```

Рисунок 3.16 – Додавання дещифрованого тексту до текстового файлу

Результат дешифрації (рис. 3.17).

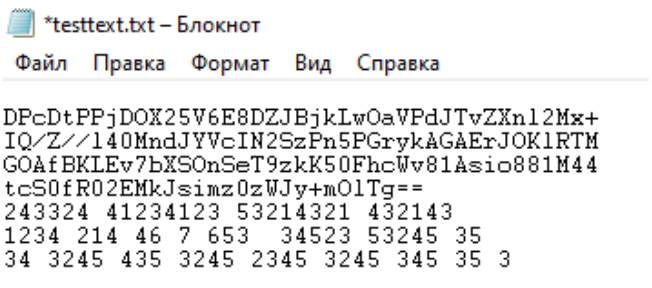

Рисунок 3.17 – Результат дешифрації

3.2.5 Архівація даних

Архівація – це важливий етап обробки і зберігання даних, який забезпечує їх компактність та ефективне використання простору. Два з популярних методів архівації, які використовуються для зменшення обсягу даних, це Deflate і Run-Length Encoding (RLE). Кожен з цих методів має свої унікальні особливості та властивості, які роблять їх застосування актуальним у різних сценаріях.

Метод Deflate є потужним алгоритмом стиснення, який використовує комбінацію LZ77 та Huffman-кодування. Цей метод володіє високою ступенем стискання та широко використовується для стиснення файлів та передачі даних через мережу. Велика ефективність та універсальність роблять Deflate одним із ключових алгоритмів архівації. Також цей алгоритм використовується в усім відомому форматі стиснення .zip.

З іншого боку, метод Run-Length Encoding (RLE) базується на принципі зберігання повторюваних символів у вигляді пар «символ-кількість». Цей простий метод дозволяє ефективно стискати послідовності, де велика кількість символів повторюється. Його простота та низький обчислювальний витрати роблять його популярним для конкретних завдань.

У дослідженні буде розглянута реалізація вищевказаних методів Deflate та RLE архівування.

Для реалізації архівації з використанням алгоритму Deflate в C#, буде використаний клас DeflateStream.

У якості прикладу знадобиться текстовий файл, який використовувався для маніпуляцій з шифруванням (рис. 3. 18).

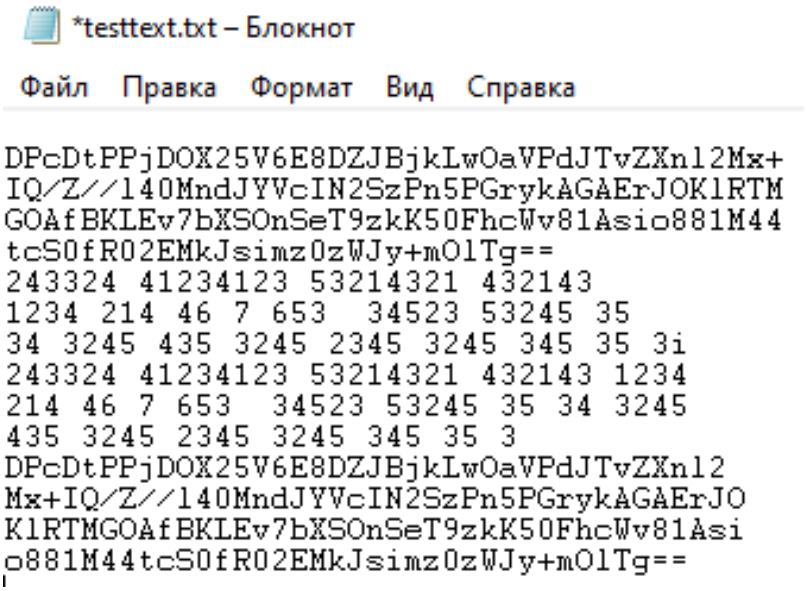

Рисунок 3.18 – Файл для архівації

У цьому прикладі (рис. 3.19) функція CompressFile приймає текст з файлу, перетворює його в байтовий масив та стискає його за допомогою DeflateStream. Функція DecompressText використовує DeflateStream для розпакування стисненого байтового масиву у рядок (рис. 3.20).

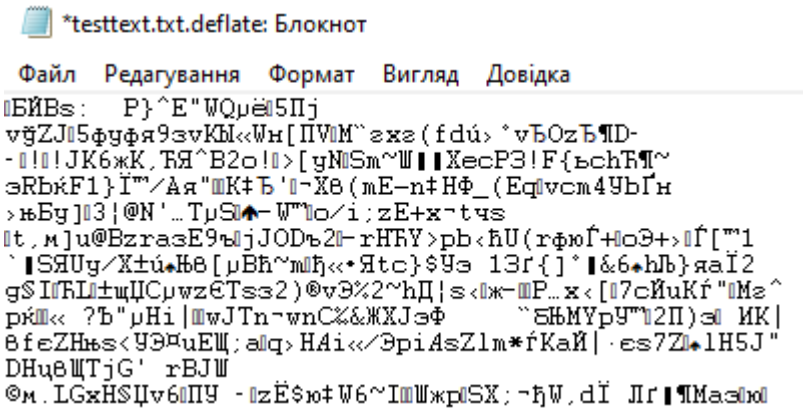

Рисунок 3.19 – Стиснений рядок

```
static void CompressFile(string filePath)
{
    try
    {
        using (FileStream originalFileStream = new FileStream(filePath, FileMode.Open))
        {
            using (FileStream compressedFileStream = File.Create(filePath + ".deflate"))
            {
                using (DeflateStream compressionStream = new DeflateStream
                    (compressedFileStream, CompressionMode.Compress))
                {
                    originalFileStream.CopyTo(compressionStream);
                }
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Помилка при архівації файлу: {ex.Message}");
    }
}
```

Рисунок 3.20 – Метод стиснення

Виконуються такі дії:

– using (FileStream originalFileStream = new FileStream(filePath, FileMode.Open)): Створюється потік для читання оригінального файлу (originalFileStream). Вихідний файл відкривається у режимі читання;

– using (FileStream compressedFileStream = File.Create(filePath + ".deflate")): Створюється потік для запису стиснутого файлу (compressedFileStream). Файл, в який буде записано стиснений вміст, має ту ж назву, що і оригінальний файл, з додаванням розширення .deflate;

– using (DeflateStream compressionStream = new DeflateStream(compressedFileStream, CompressionMode.Compress)): Тут створюється DeflateStream для стиснення. Він отримує compressedFileStream для запису стисненого вмісту;

– originalFileStream.CopyTo(compressionStream): Копіюється вміст оригінального файлу в compressionStream. Це викликає стиснення даних, які потім записуються в стиснутий файл з розширенням .deflate.

Для розпакування стисненого файлу буде використаний метод DecompressFile (рис. 3.21).

```
static string DecompressFile(string compressedFilePath)
{
    try
    {
        using (FileStream compressedFileStream = new FileStream(compressedFilePath, FileMode.Open))
        {
            using (FileStream decompressedFileStream = File.Create
                (compressedFilePath.Replace(".deflate", "_decompressed.txt")))
            {
                using (DeflateStream decompressionStream = new DeflateStream
                    (compressedFileStream, CompressionMode.Decompress))
                {
                    decompressionStream.CopyTo(decompressedFileStream);
                }
            }
        }

        return compressedFilePath.Replace(".deflate", "_decompressed.txt");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Помилка при розпакуванні файлу: {ex.Message}");
        return string.Empty;
    }
}
```

Рисунок 3.21 – Метод розпакування даних

Виконуються такі дії:

– using (FileStream compressedFileStream = new FileStream (compressedFilePath, FileMode.Open)): Тут створюється потік для читання стиснутого файлу (compressedFileStream);

– using (FileStream decompressedFileStream = File.Create (compressedFilePath.Replace(".deflate", "_decompressed.txt"))): Створюється потік для запису розпакованого файлу (decompressedFileStream). Шлях до розпакованого файлу формується шляхом заміни розширення .deflate на _decompressed.txt;

– using (DeflateStream decompressionStream = new DeflateStream (compressedFileStream, CompressionMode.Decompress)): Тут створюється DeflateStream для розпаковування. Йому передається потік compressedFileStream для читання стиснутого вмісту;

– decompressionStream.CopyTo(decompressedFileStream): Копіюється вміст стиснутого файлу в розпакований файл. Це відбувається за допомогою методу CopyTo, який читає дані з decompressionStream і записує їх у decompressedFileStream.

Основна ідея RLE (Run-Length Encoding) (рис. 3.22) полягає в тому, щоб замінити повторювані символи вхідного тексту кількістю повторень і самим символом.

Реалізація RLE матиме такі етапи:

– створюється об’єкт StringBuilder, який буде використовуватися для побудови стисненого рядка;

– за допомогою циклу for відбувається проходження по вхідному тексту та обробляється кожен символ вхідного тексту;

– використовуючи цикл while, визначається кількість повторень поточного символу;

– додавання у стиснений рядок: Кількість повторень і символ додаються до рядка compressed;

– записується стиснутий рядок у файл за допомогою функції WriteToFile.

```
static void CompressRle(string compressedFilePath, string input)
{
    try
    {
        StringBuilder compressed = new StringBuilder();

        for (int i = 0; i < input.Length; i++)
        {
            int count = 1;
            while (i + 1 < input.Length && input[i] == input[i + 1])
            {
                count++;
                i++;
            }

            compressed.Append(count);
            compressed.Append(input[i]);
        }

        WriteToFile(compressedFilePath, compressed.ToString());
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Помилка при стисненні файлу: {ex.Message}");
    }
}
```

Рисунок 3.22 – Метод Run-Length Encoding

3.3 Інструкція користувача

Таким чином, застосунок матиме 3 вікна з різним функціоналом:

– перше вікно LoadFromFile призначене для завантаження еталонних файлів до застосунку;

– друге вікно EnterFactValues для запису фактичних значень з іспитів до еталонних;

– третє вікно Code&Packing для обчислювання відхилень та архівування.

Для користування застосунком користувач потрібен мати опит роботи з директивами та знання предметної області для пошуку потрібних файлів та їх редагування.

Перш за все потрібно запустити застосунок завантажити файл до застосунку (рис. 3.23):

– перейти на вкладку LoadFromFile;

– натиснути кнопку «Load» та обрати у браузері (рис. 3.24) потрібний файл для обробки.

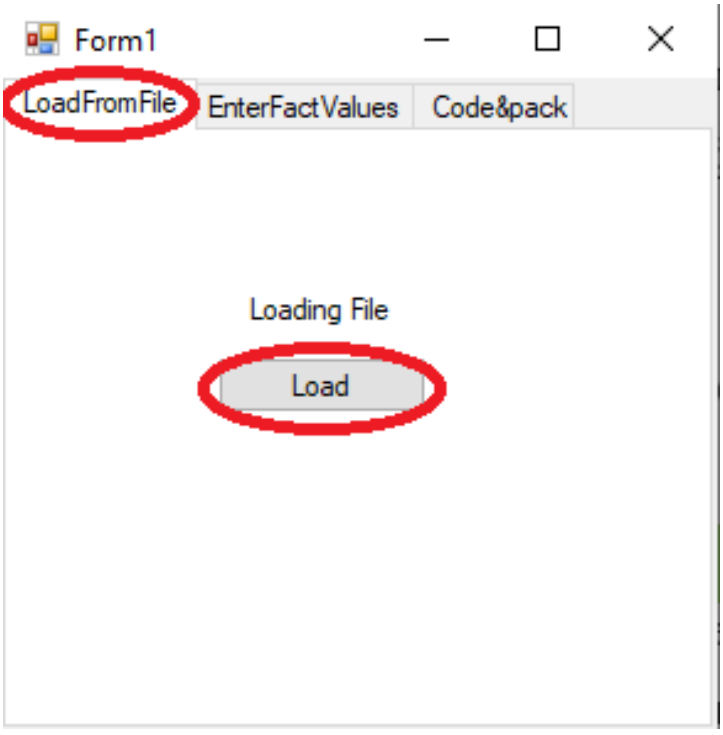

Рисунок 3.23 – Завантаження еталонного файлу

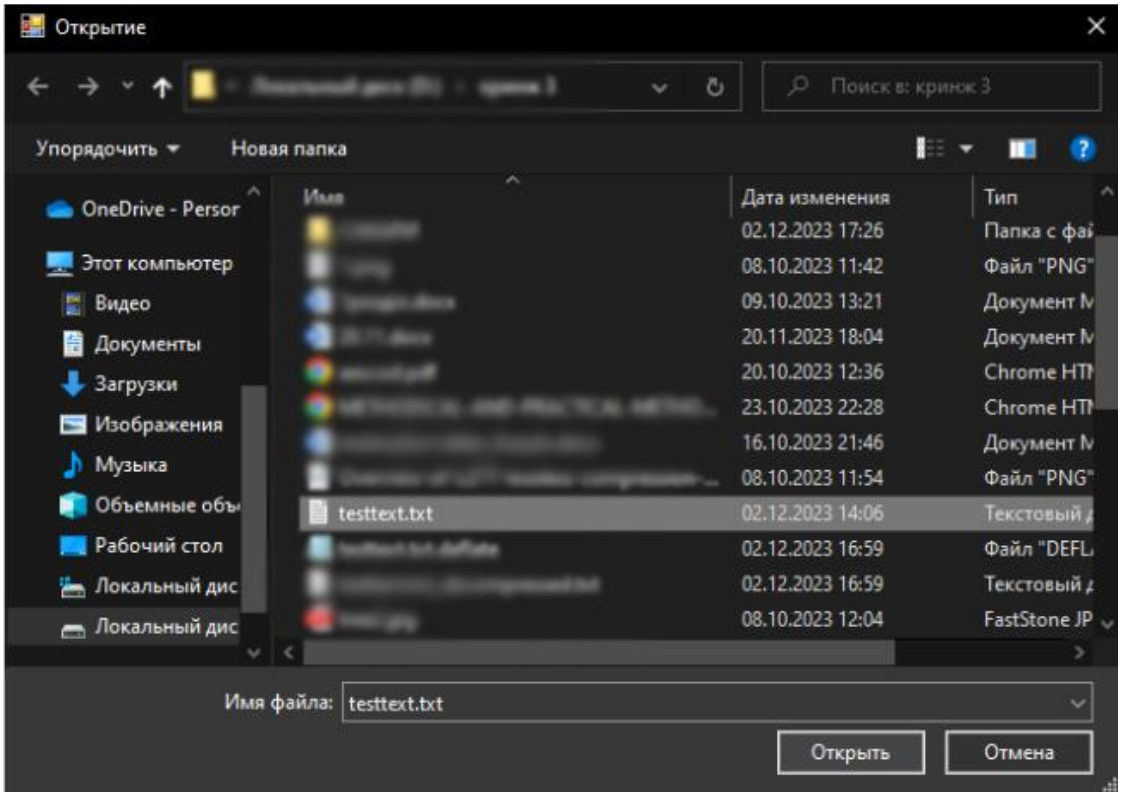

Рисунок 3.24 – Вибір потрібного файлу

Додання фактичних значень:

– обрати вкладку EnterFactValues (рис. 3.25);

– ввести фактичні дані (рис. 3.26);

– натиснути кнопку «AddCalculate»(рис. 3.27).

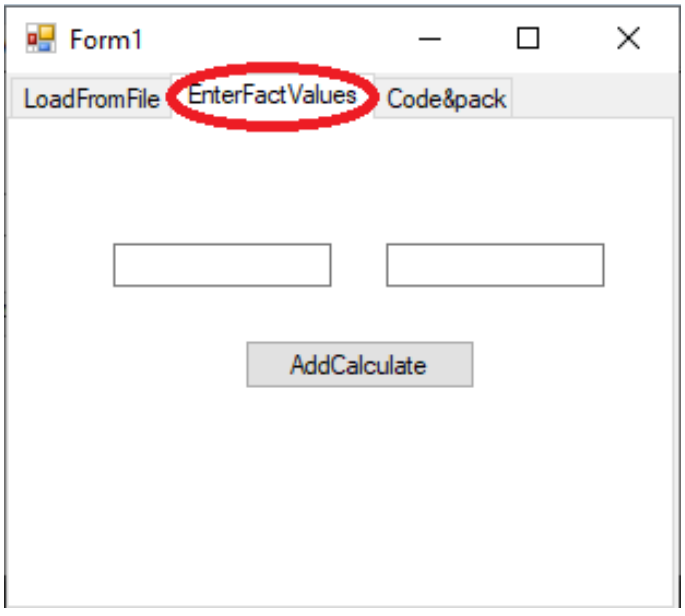

Рисунок 3.25 – Вкладка EnterFactValue

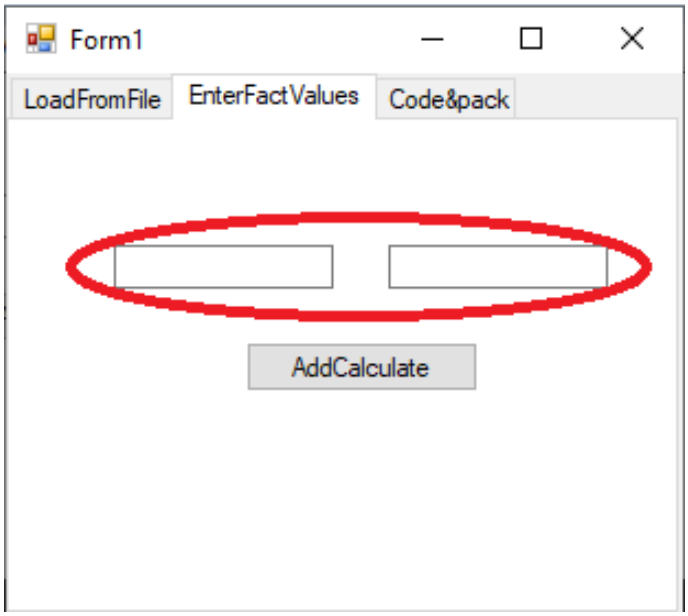

Рисунок 3.26 – Поля набору значень

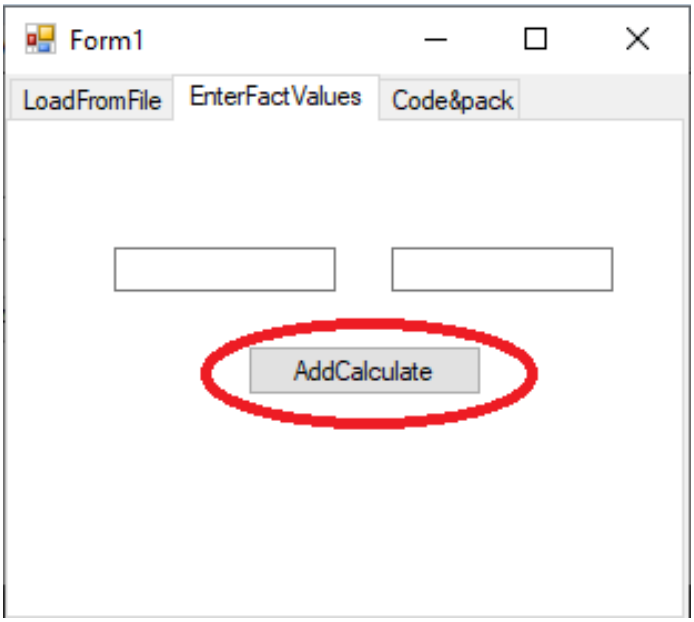

Рисунок 3.27 – Запуск розрахунків відхилень

Вибір методу стиснення та архівація:

– обрати вкладку Code&Pack (рис. 3.28);

– вибрати потрібний формат стиснення у випадаючому віконці вибору (рис. 3.29);

– натиснути кнопку «Pack» для пакування за обраним форматом стиснення (рис. 3.30).

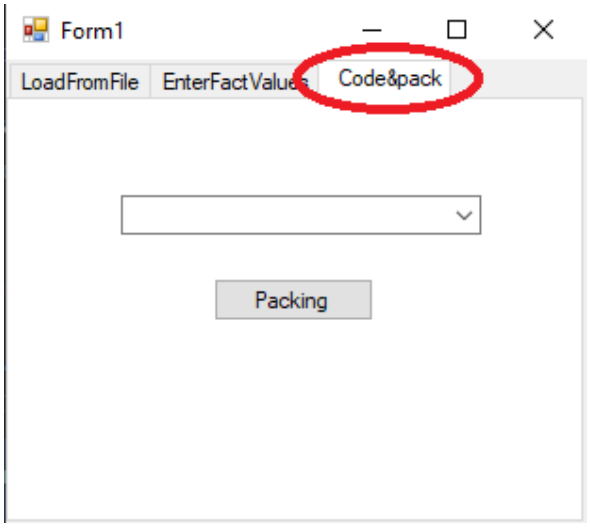

Рисунок 3.28 – Запуск розрахунків відхилень

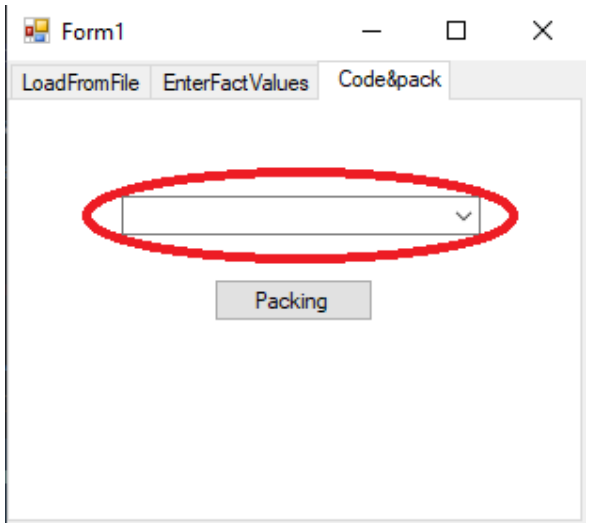

Рисунок 3.29 – Запуск розрахунків відхилень

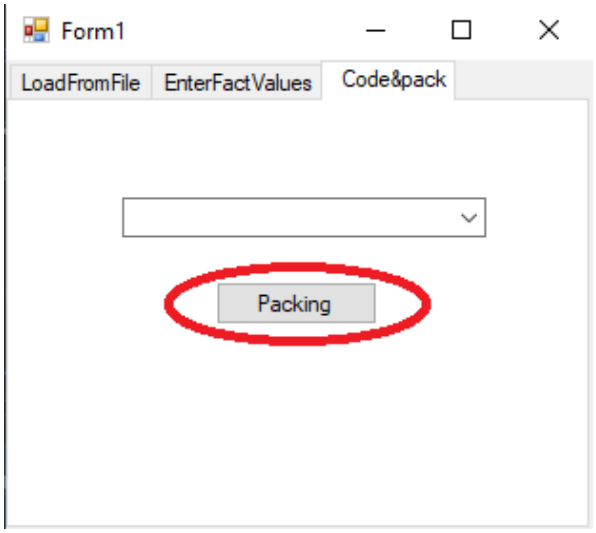

Рисунок 3.30 – Запуск розрахунків відхилень

3.4 Тестування розробленої моделі

Тестування моделі буде здійснено на файлі-прикладі «testtext.txt» (рис. 3.31) у виді готового до архівації.

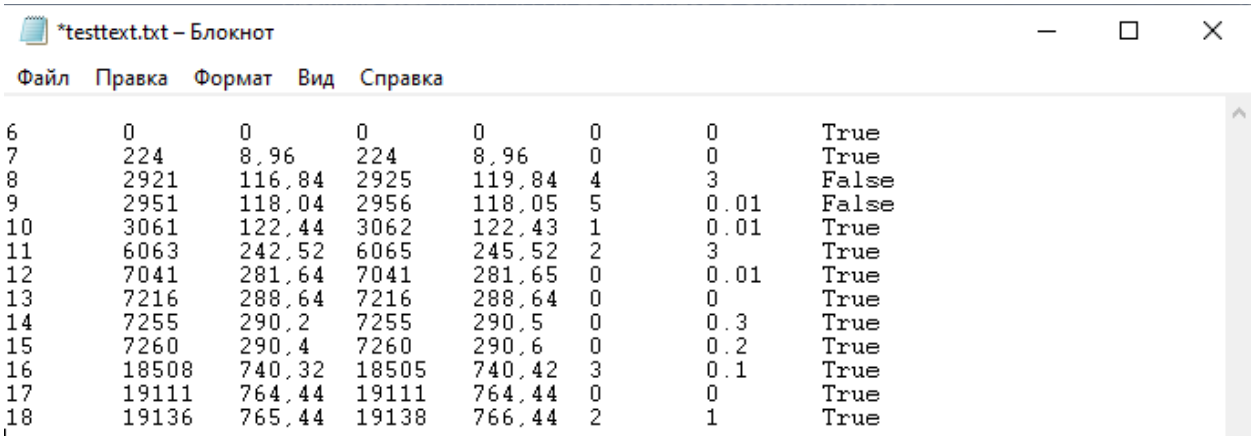

Рисунок 3.31 – Тестовий файл

Перший крок обробки це шифрування. Воно відбувається за методом AES шифрування. Після обробки текст матиме вигляд як на рисунку 3.32.

Рисунок 3.32 – Зашифровані дані

Слід зауважити, що при шифруванні розмір файлу також змінюється (рис. 3.33). AES, як правило, застосовується в блоковому режимі шифрування. Це означає, що весь вхідний текст розбивається на блоки, які шифруються незалежно один від одного. Це може призводити до додаткового вироблення, наприклад, у вигляді доповнень (padding), щоб забезпечити, що останній блок буде повний.

Деякі режими шифрування, такі як CBC (Cipher Block Chaining), вимагають використання IV для початкової ініціалізації процесу шифрування. IV може бути доданий до виводу, збільшуючи загальний розмір.

Для контролю та забезпечення цілісності даних та перевірки валідності шифрованого тексту, можуть використовуватися коди аутентифікації повідомлень (Message Authentication Codes, MAC). Ці дані для контролю цілісності також можуть збільшувати розмір виводу.

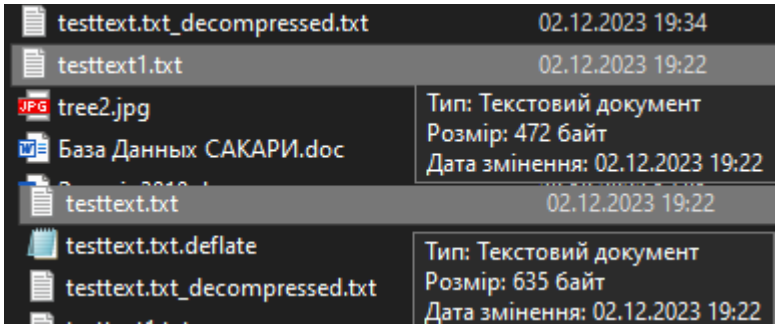

Рисунок 3.33 – Розмір файлу після шифрування

Наступний крок це вибір методу стиснення. На вибір даються Deflate та Run Length Encoding методи.

Відпрацювавши за методом Deflate створюється файл формату .deflate з меншою ємністю (рис. 3.34).

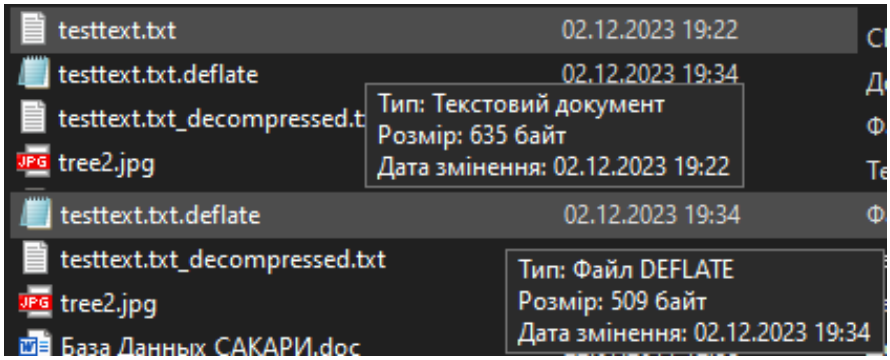

Рисунок 3.34 – Стиснений файл Deflate

Відпрацювавши за методом Run Length Encoding створюється файл формату .rle (рис. 3.35) та має вигляд як на рисунку 3.36.

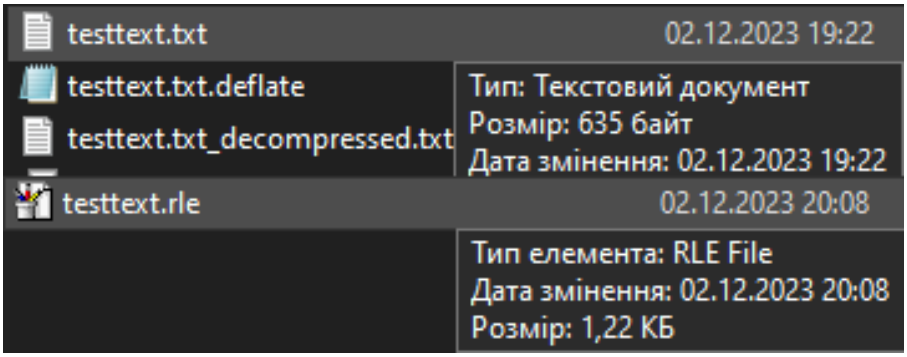

Рисунок 3.35 – Стиснений файл RLE

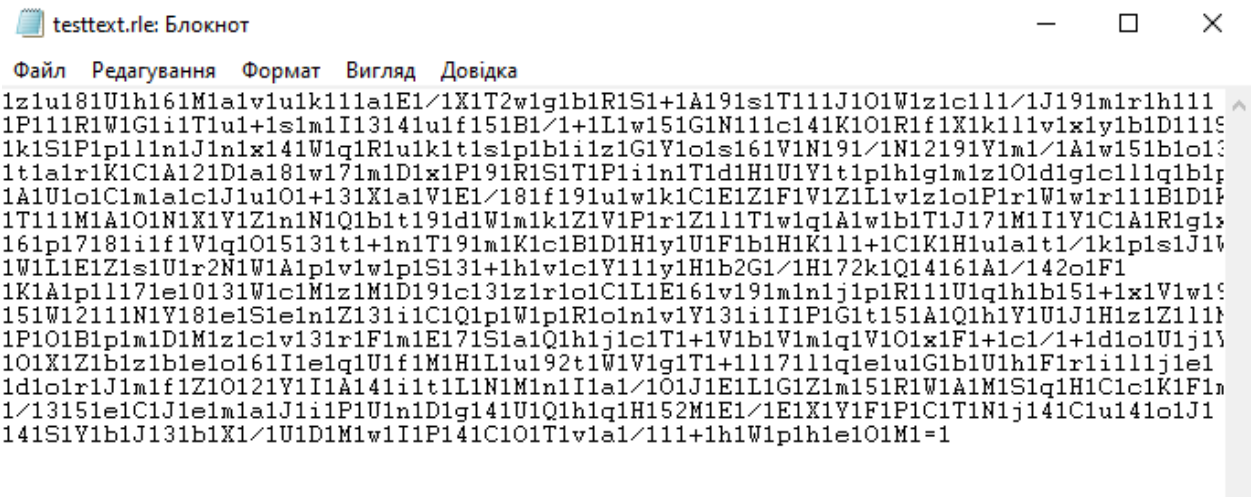

Рисунок 3.36 – Файл RLE

Вихідний файл мав розмір 635 байт після стиснення 509 та 1,22 кілобайти відповідно.

Як видно з прикладу більш сучасний метод Deflate у даній ситуації має виграш в стиснені даних. Це обумовлено тим, що Run Length Encoding використовує просту логіку стиснення повторюваних символів. Однак, при порівнянні з Deflate, він виявляється менш ефективним у багатьох сценаріях.

Deflate, в основному, використовує комбінацію LZ77 та Huffman coding. Це дозволяє йому краще виявляти шаблони та повторення в даних, що дозволяє досягти більшого ступеня стиснення. У порівнянні з RLE, Deflate може ефективно працювати з більш різноманітними типами даних, такими як тексти, зображення, та інші.

Окрім того, Deflate використовує більш складні алгоритми, що дозволяє йому адаптуватися до різних структур даних та забезпечувати гнучкість в різних сценаріях. У практиці, це робить Deflate більш універсальним та потужним інструментом для стиснення різних типів файлів.

ВИСНОВКИ

У рамках кваліфікаційної роботи було досліджено та реалізовано систему шифрування та архівування даних за допомогою таких методів як Advanced Encryption Standard, Deflate та Run-Length Encoding шифрування.

Як виявилось ці методи різні, та підходять для різних ситуацій. Кожен з них має сильні та слабкі сторони. Deflate метод шифрування має набагато складнішу структуру ніж другий метод, окрім складної структури забезпечує гарну якість стискання файлів. З боку RLE маємо простий алгоритм для числових рядків які повторюються. Цей алгоритм підходить для стиснення однорідних масивів даних. Реалізована система автоматичного контролю бере ці особливості та об'єднує, отримуючі гарну якість шифрування та стиснення даних.

AES шифрування має велику якість шифрування даних: для трьох варіантів ключів AES повний перебір вимагає 2^{127} , 2^{191} чи 2^{255} операцій відповідно. Навіть найменше з цих чисел свідчить, що атака з використанням перебору ключів сьогодні немає практичного значення. Шифрування є ключовою складовою інформаційної безпеки та конфіденційності даних. Надійні методи шифрування дозволяють захищати інформацію від несанкціонованого доступу, забезпечуючи конфіденційність та цілісність. правильне використання методів шифрування допомагає захищати конфіденційність та цілісність даних в світі, де безпека даних стає все більше завданням першочергового значення.

Результати дослідження апробовано у вигляді тези доповіді під час 6 міжнародної наукової конференції «METHODICAL AND PRACTICAL METHODS OF CREATING INVENTIONS» [30].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Що таке архівування даних. URL: https://smartik.kiev.ua/shcho-take-arkhivuvannia-danykh/ (дата звернення 07.09.2023).

2. Машталір С.В. (2021) Комплекс навчально-методичного забезпечення навчальної дисципліни "Методологія стиснення даних" підготовки магістра, спеціальність 122 - Комп'ютерні науки, освітньо-професійна програма: навч. посібник. Харків, ХНУРЕ.

3. D. Salomon, G. Motta, D. Bryant (2007). Data Compression: The Complete Reference.

4. Christof Paar and Jan Pelzl (2009). Understanding Cryptography: A Textbook for Students and Practitioners.

5. Lars R. Knudsen (2011). The Block Cipher Companion.

6. Bernard Menezes (2018). Cryptography and Network Security: Principles and Practice.

7. Yakymenko I., Kasyanchuk M., Nykolajchuk Y. (2010) Matrix algorithms of processing of the information flow in computer systems based on theoretical and numerical Krestenson's basis. *Proceedings of the X-th International Conference "Modern Problems of Radio Engineering, Telecommunications and Computer Science" (TCSET–2010).* Lviv-Slavske. p. 241.

8. Yang L. L., Hanzo L. (2001) Minimum-distance decoding of redundant residue number system codes. *Proc. IEEE ICC '2001. Helsinki (Finland)*. pp. 2975-2979.

9. Гороховатський, В. О., & Творошенко, І. С. (2021). Методи інтелектуального аналізу та оброблення даних: навч. посібник.

10. Кобилін, О. А., & Творошенко, І. С. (2021). Методи цифрової обробки зображень: навч. посібник. *Харків: ХНУРЕ*.

11. Kobylin O., Gorokhovatskyi V., Tvoroshenko I., and Peredrii O. (2020) The application of non-parametric statistics methods in image classifiers based on structural description components, *Telecommunications and Radio Engineering*, 79(10), pp. 855-863.

12. The flow of improved BRISK algorithm. URL: https://www.researchgate.net/figure/the-flow-of-improved-BRISK-algorithm_fig1_328946366 (дата звернення 14.10.2023).

13. Dafydd Stuttard, Marcus Pinto (2011). The Web Application Hacker's Handbook.

14. Яковлева О. В. (2017) Комплекс навчально-методичного забезпечення навчальної дисципліни "Бази даних та інформаційні системи" підготовки бакалаврів спеціальності 122 – Комп'ютерні науки, спеціалізація "Інформатика" Харків, ХНУРЕ.

15. A. Silderschatz, Henry F. Korth, S. Sudarchan (2005). Database System Concepts.

16. Jon Duckett (2011). HTML & CSS: Design and Build Web Sites.

17. Основні концепції мови програмування C# URL: https://metanit.com/sharp/ (дата звернення 17.10.2023).

18. Markus Winand (2012). SQL performance explained.

19. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., and Zeghid M. (2022) Tools for fast metric data search in structural methods for image classification, *IEEE Access*, 10, pp. 124738-124746.

20. Gorokhovatskyi V., Tvoroshenko I., Kobylin O., and Vlasenko N. (2023) Search for visual objects by request in the form of a cluster representation for the structural image description, *Advances in Electrical and Electronic Engineering*, 21(1), pp. 19-27.

21. Гороховатський В.О., Творошенко І.С., Чмутов Ю.В. (2022) Застосування систем ортогональних функцій для формування простору ознак у методах класифікації зображень, *Сучасні інформаційні системи*, 6(3), С. 5-12.

22. Гороховатський В., Передрій О., Творошенко І., Марков Т. (2023) Матриця відстаней для множини компонентів структурного опису як інструмент для створення класифікатора зображень, *Сучасні інформаційні системи*, 7(1), С. 5-13.

23. Pomazan V., Tvoroshenko I., and Gorokhovatskyi V. (2023) Development of an application for recognizing emotions using convolutional neural networks, *International Journal of Academic Information Systems Research*, 7(7), pp. 25-36.

24. Pomazan V., Tvoroshenko I., and Gorokhovatskyi V. (2023) Handwritten character recognition models based on convolutional neural networks, *International Journal of Academic Engineering Research*, 7(9), pp. 64-72.

25. Tvoroshenko I., Gorokhovatskyi V., Kobylin O., and Tvoroshenko A. (2023) Application of deep learning methods for recognizing and classifying culinary dishes in images, *International Journal of Academic and Applied Research*, 7(9), pp. 57-70.

26. Gorokhovatskyi V., Tvoroshenko I. (2023) Identification of visual objects by the search request. *International scientific symposium «INTELLIGENT SOLUTIONS-S». Computational intelligence (results, problems and perspectives). Decision making theory: proceedings of the international symposium*, September 28, 2023, Kyiv-Uzhorod, Ukraine, pp. 25-27.

27. Yakovleva O., Kovač M., Ardasov V. & Yeremenko I. (2023). Study on adding functionality to online conference system for monitoring the participant activities, *Public Administration and Regional Development*, 19(1), pp. 158-184.

28. Daradkeh Y.I., Gorokhovatskyi V., Tvoroshenko I., Gadetska S., and Al-Dhaifallah M. (2023) Statistical data analysis models for determining the relevance of structural image descriptions, *IEEE Access*, 11, pp. 126938-126949.

29. Tvoroshenko I., Pomazan V., Gorokhovatskyi V., and Kobylin O. (2023) Application of video data classification models using convolutional neural networks, *International Journal of Academic and Applied Research*, 7(11), pp. 134-145.

30. Бганцов Є. (2023) Шифрування даних. Опис та структура алгоритму симетричного шифрування AES, *Abstracts of VI International Scientific and Practical Conference «Methodical and practical methods of creating inventions», (October 24 – 27, 2023).* Sofia, Bulgaria, pp. 222-225.