НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2021 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Інженерія програмного забезпечення комп’ютерних систем”
спеціальності 121 “Інженерія програмного забезпечення”

на тему: Метод гомоморфного шифрування для захищеної обробки даних в хмарах

Виконав (-ла): студент (-ка) 4 курсу, групи ІП-73
(шифр групи)

Тимошенко Іван Андрійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доцент, к.т.н. Марковський О.П. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) д.т.н., проф. Сімоненко В.П. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент декан ФПМ, д.т.н., проф. Дичка І.А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2021 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2021 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

_____Тимошенка Івана Андрійовича_____

1. Тема проєкту *Метод гомоморфного шифрування для захищеної обробки даних в хмарах*
керівник проєкту *Марковський Олександр Петрович, к.т.н., доцент*,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від ___11.05___ 2021 року №_1139-с_

2. Термін здачі студентом закінченого проєкту _____*2 червня 2021 р.*

3. Вихідні дані до проєкту *див. технічне завдання*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються) *Аналіз проблеми обробки даних з використанням віддалених обчислювальних ресурсів. Розробка методів обробки конфіденційної інформації з використанням*

віддалених обчислювальних ресурсів. Реалізація системи віддаленої захищеної медіанної та середньоарифметичної фільтрації.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) *структурна схема системи, схема взаємодії класів, блок-схема алгоритму.*

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Норм.контроль	*Сімоненко В.П., проф., д.т.н.*		

7. Дата видачі завдання_____

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*10.12.2021-15.12.2021*	
2.	*Вивчення та аналіз завдання*	*15.12.2021-15.03.2021*	
3.	*Розробка архітектури та загальної структури системи медіанної та середньоарифметичної фільтрації.*	*15.03.2021-25.03.2021*	
4.	*Розробка структур окремих підсистем*	*25.03.2021-5.04.2021*	
5.	*Програмна реалізація системи*	*5.04.2021-15.04.2021*	
6.	*Оформлення пояснювальної записки*	*15.04.2021-20.05.2021*	
7.	*Захист програмного продукту*	*25.04.2021*	
8.	*Передзахист*	*23.05.2021*	
9.	*Захист*	*19.06.2021*	

Студент-дипломник

(підпис)

Керівник проєкту

(підпис)

Анотація

Проект присвячено розробці системи захищеної обробки конфіденційних даних за допомогою віддалених обчислювальних ресурсів.

Були розроблені методи для виконання медіанної та середньоарифметичної фільтрації зображення, а також, метод захищеної середньоарифметичної фільтрації групи зображень. Розроблені методи дозволяють виконувати обробку зображень у зашифрованому вигляді, що дозволяє використовувати хмарні технології для покращення якості зображень.

Завдяки використанню потужних систем вдається підвищити швидкість обробки конфіденційних даних на поряд.

Annotation

The goal of presented by diploma project is development of a system of secure processing of confidential data using remote computing resources.

Methods for performing median and arithmetic mean image filtering have been developed, as well as a method for secure arithmetic mean filtering of a group of images. The developed methods allow performing image processing in encrypted form, which allows the use of cloud technologies to improve image quality.

The processing speed of sensitive data can be significantly increased through the use of powerful systems.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпляр	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467400.002 ТЗ***	Метод гомоморфного	5		
			шифрування для захищеної			
			обробки даних в хмарах			
			Технічне завдання			
	А4	***ІАЛЦ.467400.003 ПЗ***	Метод гомоморфного	70		
			шифрування для захищеної			
			обробки даних в хмарах			
			Пояснювальна записка			
	А4	***ІАЛЦ.467400.004 Д1***	Метод гомоморфного	1		
			шифрування для захищеної			
			обробки даних в хмарах			
			Блок-схема алгоритму			
	А4	***ІАЛЦ.467400.005 Д2***	Метод гомоморфного	1		
			шифрування для захищеної			
			обробки даних в хмарах			
			Діаграма класів			
	А4	***ІАЛЦ.467400.006 Д3***	Метод гомоморфного	1		
			шифрування для захищеної			
			обробки даних в хмарах			
			Структурна схема			
	А4	***ІАЛЦ.467400.007 Д4***	Метод гомоморфного	15		
			шифрування для захищеної			
			обробки даних в хмарах			
			Текст програми			

					ІАЛЦ. 467400.001 ОА		
Зм	***Лист***	***№ докум.***	***Підп.***	***Дата***			
Розроб		Тимошенко І.А.			*Метод гомоморфного шифрування для захищеної обробки даних в хмарах* Опис альбому	Літ.	Аркуш 1 / Аркушів 1
Перев		Сімоненко В. П.					
					КПІ ім. Ігоря Сікорського», ФІОТ, ІП-73		

Технічне завдання

до дипломного проекту

на тему: «Метод гомоморфного шифрування для захищеної обробки даних в хмарах»

Київ – 2021

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Технічне завдання поширюється на розробку нових методів віддаленої захищеної медіанної та середньоарифметичної фільтрації та дослідження програмних засобів його реалізації.

Область застосування – системи сворення обробки та збереження зображень з можливістю підключення до віддалених обчислювальних ресурсів.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання дипломного проєкту кваліфікаційно-освітнього рівня «бакалавр програмної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Мета проекту полягає в збільшення ефективності обробки зображень з метою видалення цифрових шумів, за рахунок використання віддалених потужних обчислювальних ресурсів для захищеної медіанної та середньоарифметичної фільтрації.

4. ДЖЕРЕЛА РОЗРОБКИ

4.1. Прэтт У. Цифровая обработка изображений / У. Прэтт. – М. : Мир, 1982. – 480 с.

4.2 Sathish V. Cloud-based Image Processing With Data Priority Distribution Mechanism./ Sathish V., Sangeetha T.A. // Journal of Computer Applications.- Vol.6, №1.- 2013.- P. 6-8.

4.3 Марковський О.П. Захищена реалізація фільтрації зображень в GRID-системах / О.П. Марковський, М.В. Невдащенко, А.М. Білашевська // Вісник Національного технічного університету України ”КПІ” Інформатика, управління та обчислювальна техніка. – Київ: ВЕК+. – 2014. – № 61. – С.105-109.

4.4 Марковський О.П., Гуменюк І.О., Міратаї Аліреза, Торошанко Я.І., Волощук М.О. Метод прискореної захищеної фільтрації зображень на віддалених комп’ютерних системах / О.П. Марковський, // Телекомунікаційні та інформаційні технології**. -** № 4 (65).- 2019.- С.99-110.

4.5 Марковський О.П., Гуменюк І.О., Міратаї Аліреза, Торошанко Я.І., Волощук М.О. Метод прискореної захищеної фільтрації зображень на віддалених комп’ютерних системах / О.П. Марковський, // Телекомунікаційні та інформаційні технології. - № 4 (65).- 2019.- С.99-110.

4.6 Форсайт Д. Компьютерное зрение. Современный подход / Д. Форсайт, Ж. Понс. – М. : Вильямс, 2004. – 928.

4.7 Boroujerdi N. Cloud Computing: Changing Cogitation about Computing/ N. Boroujerdi, S. Nazem // IJCSI International Journal of Computer Science Issues. – Vol. 9. – Issue 4. – 2012. – №3. – PP. 169-180.

4.8 Костенко Ю. В. Метод защищенного модулярного экспоненцирования на удаленных компьютерных системах / Ю. В. Костенко, А.П. Марковский, О.В. Русанова // Вісник Національного технічного університету України “КПІ” Інформатика, управління та обчислювальна техніка. – К.: ТОО „ВЕК+”. – № 64. – 2016. – С. 51-54.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до продукту, що розробляється

5.1.1 Розроблені методи шифрування зображень повинні мати високий рівень захисту від зовнішніх атак при передачі зображень по відкритому каналу.

5.1.2 Метод фільтрації зображень на віддаленому сервері не має розкривати конфіденційної інформації про оригінальне зображення.

5.1.2 Сумарна кількість обчислювальних ресурсів необхідних для виконання шифрування та дешифрування зображення повинна бути меншою за кількість ресурсів необхідних для виконання фільтрації оригінальним алгоритмом.

5.1.3 Результат обробки зображення за допомогою віддалених методів обробки має повністю співпадати з результатом обробки зображення оригінальним методом фільтрації.

5.1.4 Розроблені методи віддаленої обробки зображень повинні підтримувати обробку зображень представлених в популярних кольорових моделях RGB та HSV.

5.2 Вимоги до програмного забезпечення

- Операційна система Windows 7, Windows 10
- Visual Studio 2015 та вище
- C# .Net 5.0 і вище

5.3. Вимоги до апаратної частини

- Процесор рівня Intel i5 і вище.
- Оперативна пам’ять не менше 500 МБ.
- Вільне місце на жорсткому диску не менше 100 МБ.

6. ЕТАПИ РОЗРОБКИ

	Дата
Вивчення літератури	20.12.2020
Створення та узгодження технічного завдання	15.01.2021
Вивчення літературних джерел	27.01.2021
Розробка методів віддаленої фільтрації	14.02.2021
Розробка програмної моделі	01.05.2021
Відлагодження програми та виправлення помилок	15.05.2021
Оформлення документації дипломного проекту	02.06.2021

Пояснювальна записка
до дипломного проекту

на тему: «Метод гомоморфного шифрування для захищеної обробки даних в хмарах»

Київ – 2021

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

RGB	(Red Green Blue) Кольорова модель в основі якої лежить три кольори: червоний, зелений, синій
HSV	(Hue Saturation Value) Кольорова модель в основі якої лежить три компоненти: відтінок, насиченість, яскравість
OTP	(One-time pad) криптографічна техніка абсолютно надійного шифрування даних

ВСТУП

У сучасному світі стрімкий розвиток інформаційних технологій призвів до потреби у рості потужностей обчислювальних ресурсів. Для вирішення багатьох ресурсномістких задач витрати на придбання необхідного обладнання є недоцільними. Саме тому з кінця минулого сторіччя почалась розроблятись ідея спільного використання обчислювальних ресурсів [1]. На сьогоднішній день, як результат даних досліджень існує широкий вибір технологій для вирішення ресурсномістких задач. Одним з найпопулярніших рішень на сьогодні є хмарні обчислення.

Основною ідеєю хмарних обчислень є надання обчислювальних ресурсів в якості послуги. Ці ресурси можуть включати що завгодно: програмне забезпечення, стороннє сховище даних для фотографій та інших цифрових носіїв інформації або сторонні сервери, що використовуються для вирішення широкого спектру наукових та комерційних задач. Організаціям не потрібно вкладати гроші в дороге обладнання, пристрої зберігання даних, програмне забезпечення тощо, а потрібно платити лише за ресурси, які вони використовують. Підприємства можуть оптимізувати витрати та збільшити свої пропозиції, не купуючи та не керуючи всім апаратним та програмним забезпеченням. Дослідники можуть обмінюватися та аналізувати дані в масштабах, які колись були доступні лише проектам з великим розміром фінансування, а користувачі Інтернету можуть швидко отримувати доступ до програмного забезпечення та сховища для створення, обміну та зберігання цифрових даних у кількості, яка далеко виходить за межі обчислювальних можливостей їхніх персональних пристроїв.

Віддалене використання ресурсів дозволяє зменшувати та збільшувати їх кількість та потужність відповідно до поточних потреб, а також працювати звідки завгодно, і коли завгодно, все, що потрібно - це підключення до Інтернету.

Одним з недоліків хмарних обчислень є неможливість їх використання при обробці конфіденційних даних [2], адже для цього дані повинні бути переданні на віддалені обчислювальні ресурси, що збільшує ризики їх розповсюдження. Таким чином, на сьогоднішній день існує широкий клас задач, які не можуть бути вирішенні за допомогою хмарних технологій.

РОЗДІЛ 1

АНАЛІЗ ПРОБЛЕМИ ОБРОБКИ ДАНИХ З ВИКОРИСТАННЯМ ВІДДАЛЕНИХ ОБЧИСЛЮВАЛЬНИХ РЕСУРСІВ

1.1. Методи віддаленого покращення якості зображень за рахунок видалення цифрових шумів

Цифровий шум - дефект зображення, внесений фотосенсорами та електронікою пристроїв, які їх використовують внаслідок недосконалості технологій, а також фотонної природи світла. Шум є невід'ємною частиною при створені, передачі чи обробці цифрових зображень [3]. Ми можемо представити спотворене зображення у вигляді функції: $A(x, y) = B(x, y) + H(x, y)$, де, $A(x, y)$ – функція зіпсованого зображення, $B(x, y)$ – функція оригінального зображення, $H(x, y)$ – функція шуму зображення.

Існує багато різновидів шумів зображення [4]. Серед них можна виділити три основних типи.

Гауссовий шум - це статистичний шум, що має функцію щільності ймовірності, рівну нормальному розподілу, також відому як Гаусовий розподіл. Для генерування цього шуму до функції оригінального зображення додається випадкова гауссова функція. Його також називають електронним шумом, оскільки він виникає в підсилювачах або детекторах.

Шум солі та перцю виникає при додаванні до зображення випадкових яскравих і випадкових темних пікселів у всьому зображенні. Він також відомий як імпульсний шум. Цей шум може бути викликаний різкими та раптовими порушеннями в сигналі при передачі зображення.

Основною проблемою оптичної та цифрової голографії є наявність спекл-шуму в процесі реконструкції зображення. Спекл - випадкова інтерференційна картина, яка утворюється при взаємного впливу когерентних хвиль, що мають випадкові зрушення фаз або випадковий набір

інтенсивностей. На такому зображені можна чітко спостерігати світлі плями, цяточки (їх і називають спекла), які розділені темними ділянками зображення.

Одним з популярних та дієвих методів боротьби з цифровими шумами – є процес фільтрації. Фільтрування даних зображень - це стандартний процес, який використовується майже в кожній системі обробки зображень. Для цього використовуються фільтри. Вони видаляють шум, зберігаючи деталі самого зображення. Вибір фільтра залежить від виду шуму та типу оригінальних даних. Одними з розповсюджених способів покращення якості зображення є медіанна та середньоарифметична фільтрація.

Основний принцип роботи фільтрів базується на ідеї знаходження у вхідному сигналі значень, які значно відрізняються від сусідніх, та подальшою заміною їх на більш підходящі [5].

Будь-яке цифрове зображення може бути представлене у вигляді матриці, де кожен елемент матриці відображає інтенсивність пікселів. Стан 2D-матриць, що відображає розподіл інтенсивності зображення, називається просторовим доменом.

$$A = \begin{bmatrix} a_{1,1}\, a_{1,2} \ldots a_{1,h} \\ a_{2,1}\, a_{2,2} \ldots a_{2,h} \\ \ldots\ldots \\ a_{k,1}\, a_{k,2} \ldots a_{k,h} \end{bmatrix}$$

$a_{x,y}$ – значення інтенсивності кольору пікселя з координатами (x, y)

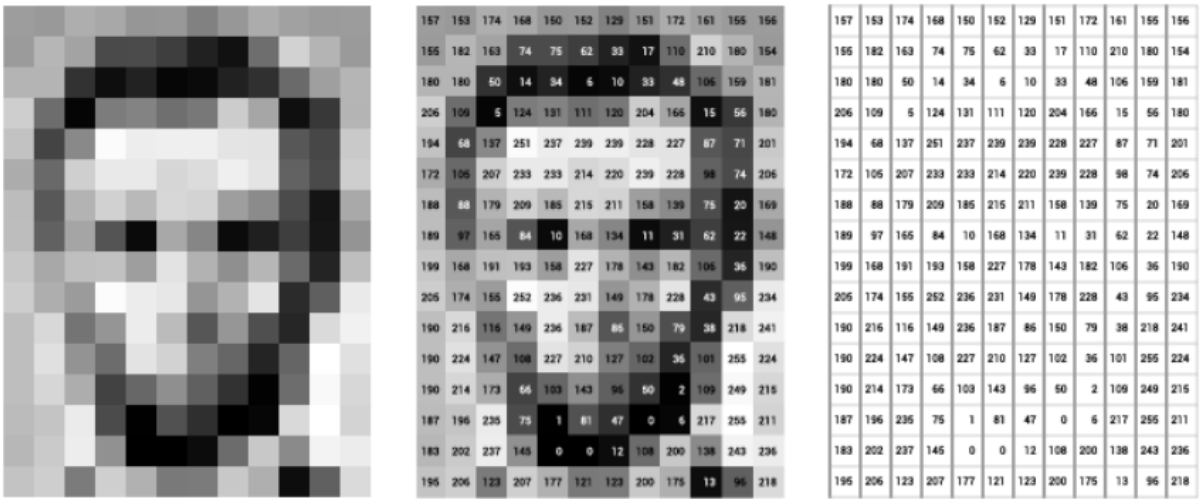


Рис.1.1. Представлення чорно-білого зображення у вигляді матриці

На рис.1.1 зображено представлення чорно-білого зображення у вигляді двомірної матриці. Кожен елемент матриці відповідає інтенсивності кольору відповідного пікселя.

Для RGB-зображення просторова область представлена у вигляді тривимірного вектора 2D-матриць.

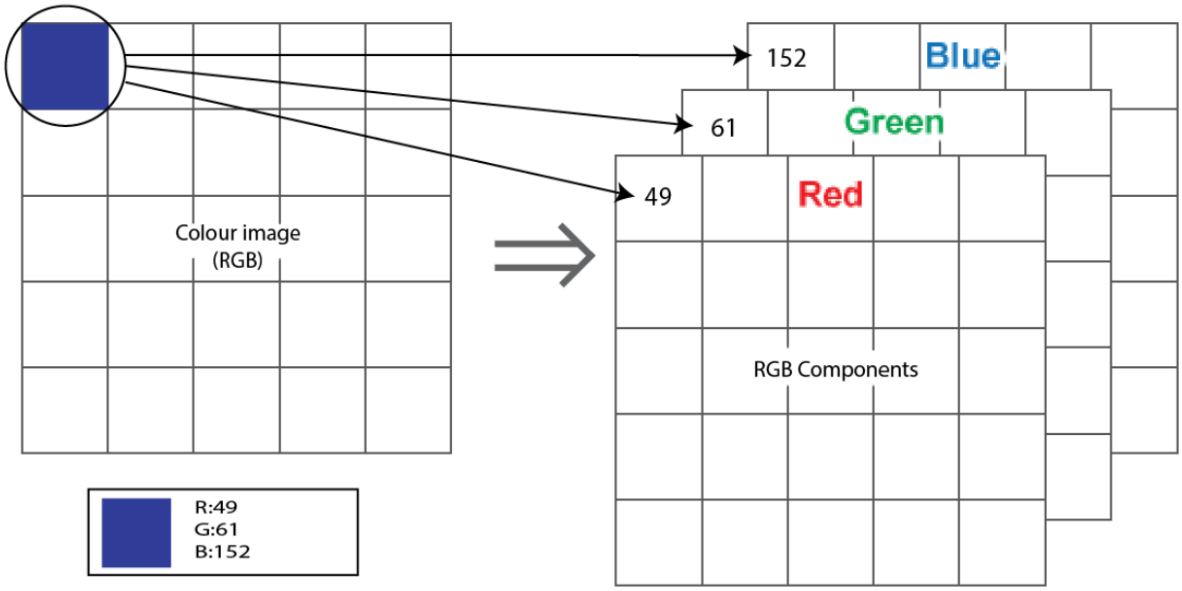


Рис.1.2. Представлення кольорового зображення у вигляді матриці [2]

Результатом обробки зображення за допомогою фільтра є відфільтроване зображення – матриця відфільтрованих значень.

$$S = \begin{bmatrix} s_{1,1}\, s_{1,2} \ldots s_{1,h} \\ s_{2,1}\ s_{2,2} \ldots s_{2,h} \\ \ldots. \\ s_{k,1}\, s_{k,2} \ldots s_{k,h} \end{bmatrix}$$

$s_{x,y}$ – значення інтенсивності кольору пікселя з координатами (x, y) після фільтрації.

Медіанний фільтр - це нелінійна цифрова техніка фільтрації, яка використовується для видалення шуму із зображення або сигналу. Фільтр являє собою вікно (апертуру), яке поступово "ковзає" по зображенню. Форма вікна може бути різноманітною, але для успішної фільтрації воно має містити непарну кількість елементів. Процес обробки здійснюється шляхом пересування вікна над зображенням. Відфільтроване зображення отримується шляхом розміщення відфільтрованого значення в місці розташування центру цього вікна на вихідному зображенні. При використанні медіанного фільтра кожне значення отримується за формулою:

$$s_{x,y} = a_{t,p} : N(a_{i,j} \le a_{t,p}) = N(a_{i,j} \ge a_{t,p}),$$

де $x - \frac{r-1}{2} \le i \le x + \frac{r-1}{2}$, $y - \frac{r-1}{2} \le i \le x + \frac{r-1}{2}$, $t \subseteq i$, $p \subseteq j$,

r – розмір квадратної апертури фільтра.

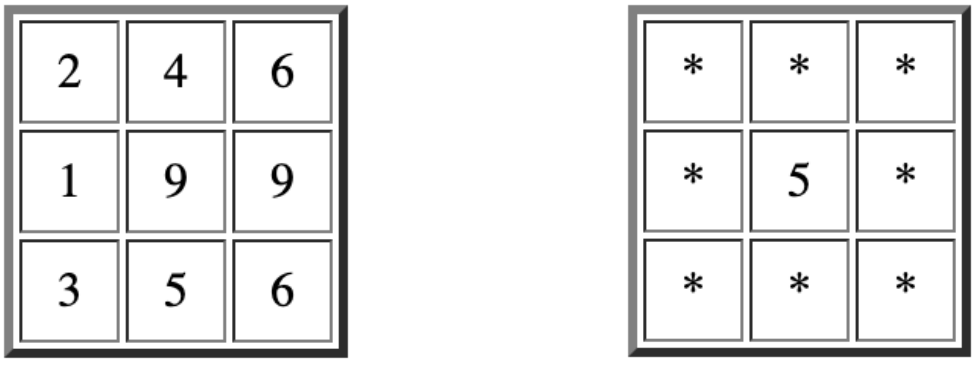


Рис.1.3. Обробка квадратної апертури за допомогою медіанного фільтра

На рис.1.3 зліва зображені вхідні нефільтровані значення квадратної апертури розміру r = 3.

Значення які потрапили в апертуру сортуються {1, 2, 3, 4, 5, 6, 6, 9, 9} після чого обирається медіанний елемент. В даній вибірці він дорівнює 5. Наступним кроком отримане значення записуються на місце центрального елемента апертури у вихідному відфільтрованому зображені.

Рис.1.4. Обробка зображення за допомогою медіанного фільтра

На рис.1.4 проілюстрований результат застосування медіанного фільтра з метою видалення шумів з зображення.

Середньоарифметична фільтрація дуже схожа на медіанну. Основною відмінністю є те, що середній елемент апертури замінюється на середньоарифметичне, а не медіанне значення його сусідів. При використанні медіанного фільтра кожне значення отримується за формулою:

$$s_{x,y} = \frac{1}{r^2} \sum_{i=x-\frac{r-1}{2}}^{x+\frac{r-1}{2}} \sum_{j=y-\frac{r-1}{2}}^{y+\frac{r-1}{2}} a_{i,j}$$

2	1	6
8	10	9
5	4	9

*	*	*
*	6	*
*	*	*

Рис.1.5. Обробка квадратної апертури за допомогою середньоарифметичного фільтра

На рис.1.5 зліва зображені вхідні нефільтровані значення квадратної апертури розміру r = 3. Зі значень, які потрапили в апертуру вираховується середньоарифметичне, після чого отримане значення записуються на місце центрального елемента апертури у вихідному відфільтрованому зображені.

Рис.1.6. Обробка зображення за допомогою середньоарифметичного фільтра

На рис.1.6 проілюстрований результат застосування середньоарифметичного фільтра з метою видалення шумів з зображення

Медіанна та середньоарифметична фільтрація є але ресурсномісткими операціями [6], адже для кожного елемента зображення потрібно знайти

медіанне чи середньоарифметичне значення його сусідів. Використання хмарних технологій дозволило б зробити обробку зображень більш ефективним, але найчастіше, зображення є конфіденційною інформацією, яка не може бути передана на сторонні ресурси в незашифрованому вигляді.

1.2. Аналіз дискретного перетворення Фур'є з метою використання для покращення якості зображення

Дискретне перетворення Фур'є є одним з найпотужніших інструментів цифрової обробки сигналів [7], який дозволяє нам знаходити спектр сигналу кінцевої тривалості. Перетворення Фур'є є важливим інструментом обробки зображень, який використовується для розкладання зображення на його синусоїдальні та косинусоїдальні компоненти. Результатом перетворення є зображення у частотній області, тоді як вхідне зображення є еквівалентом просторової області. В частотній області на зображенні кожна точка представляє певну частоту, що міститься в просторової області зображення.

Для квадратного зображення розміром N × N двовимірний DFT визначається як:

$$F(k,l) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j)\, e^{-\imath 2\pi (\frac{ki}{N} + \frac{lj}{N})}$$

де *f(a, b)* - зображення в просторовій області, а експоненціальний доданок є базовою функцією, що відповідає кожній точці *F(k, l)* у просторі Фур'є. Значення кожної точки *F(k, l)* отримується множенням просторового зображення з відповідною базовою функцією та підсумовуванням результату.

$$f(a,b) = \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} F(k,l)\, e^{\imath 2\pi (\frac{ka}{N} + \frac{lb}{N})}$$

Подібним чином зображення Фур'є може бути перетворено в просторову область. Обернене перетворення Фур'є задається формулою:

Перетворення Фур'є використовується, якщо ми хочемо отримати доступ до геометричних характеристик зображення просторового домену. Оскільки зображення в області Фур’є розкладається на синусоїдальні компоненти, легко дослідити або обробити певні частоти зображення, впливаючи тим самим на геометричну структуру в просторовій області.

Перетворення Фур'є має широкий діапазон застосувань, таких як аналіз, фільтрація, реконструкція та стиснення зображень.

1.3. Аналіз методів обробки конфіденційної інформації на віддалених обчислювальних ресурсів

На сьогоднішній день, існує велика кількість методів для захисту інформації. В більшій мірі вони вирішують проблему безпечної передачі та збереження інформації, але не дають можливості ефективно взаємодіяти з зашифрованими даними. Для обробки конфіденційної інформації на віддалених обчислювальних ресурсах, необхідно мати можливість виконувати необхідних набір операцій над переданими даними й при цьому не розголошувати їх вміст. Для ефективного використання даних алгоритмів, кількість обчислювальних ресурсів, яку витрачає користувач на шифрування й дешифрування даних, повинна бути меншою за кількість ресурсів необхідних на виконання оригінального алгоритму.

Існуючи методи віддаленої роботи з зашифрованими даними мають спільний принцип роботи [8]. Припустимо, що оригінальний алгоритм обробки даних представлений у вигляді функціонального перетворення f, тоді оброблені дані мають обчислюються за наступною формулою:

$$s_{plain} = f(a_{plain}),$$

де, a_{plain} – оригінальні дані, s_{plain} – результат обробки оригінальних даних функціональним перетворенням f.

Для віддаленої обробки конфіденційних даних, необхідно попередньо їх зашифрувати. Для цього к оригінальним даним застосовується інше функціональне перетворення *C*, тоді зашифровані дані мають наступний вигляд:

$$a_c = C\ (a_{plain}, k_e),$$

де k_e – ключ шифрування даних, a_c – зашифровані оригінальні дані.

Після шифрування дані в захищеному вигляді відправляють до віддалених обчислювальних ресурсів, де відбувається їх обробка за допомогою функціонального перетворення *g*. Оригінальне перетворення *f* може співпадати з перетворенням *g*. Після обробки данні мають наступний вигляд: $S_e = g(C\ (a_{plain}, k_e))$.

Після виконання функціонального перетворення g, дані повертаються користувачеві й дешифруються за допомогою зворотного перетворення *D*. Після дешифрування данні повинні співпадати з результатом обробки оригінальних даних перетворенням *f* [9].

$$S_{plain} = D(g(C\ (a_{plain}, k_e)), k_d),$$

де k_d – ключ дешифрування даних.

Основною складністю при вирішенні задачі віддаленої обробки інформації є знаходження таких функціональних перетворень *C*, *D* та *g*, щоб, по-перше, виконувалась рівність для довільних оригінальних даних [10], а, по-друге, щоб сумарні обчислювальні ресурси витраченні на перетворення *C* , *D* не перевищували ресурсів необхідних для виконання перетворення *f.*

Висновки до розділу 1

В результаті аналізу існуючих труднощів пов’язаних з обробкою цифрових зображень, методів вирішення даних проблем, а також можливостей застосування віддалених обчислювальних ресурсів, можна зробити наступні висновки:

1. Обробка цифрових зображень за допомогою медіанної та середньоарифметичної фільтрації є популярними та ефективними методами покращення якості зображення за рахунок видалення шумів. Недоліком даних підходів є висока ресурсозатратність, оскільки для успішної обробки необхідно обробити кожний піксель окремо. Обробка зображень у незашифрованому вигляді з використанням віддалених обчислювальних ресурсів є неможливою, оскільки конфіденційні зображення не можуть бути передані до віддалених ресурсів у незахищеному вигляді.

2. Обрані критерії ефективності обробки конфіденційної інформації на віддалених обчислювальних ресурсах. По-перше, рівень захищеності зашифрованої оригінальної інформації, що передається для обробки на віддалені ресурси. По-друге, алгоритмічна складність шифрування та дешифрування вхідних даних. Для ефективної реалізації, кількість операцій необхідних користувачеві для захисту конфіденційної інформації не повинна перевищувати кількість операцій необхідну для виконання оригінального алгоритму.

3. Аналіз існуючих методів для виконання захищеної обробки інформації з використанням вище перелічених критеріїв показав, що існуючи методи не є досконалими. Наявність можливостей для покращення якості віддаленої обробки конфіденційної інформації підтверджує доцільність проведення подальших досліджень у цьому напрямку.

4. Проведений аналіз існуючих технологій віддалених обчислювальних систем показав існуючий потенціал використання хмарних обчислень в операціях обробки зображень з метою покращення їх якості. Використання

даних технологій дозволить суттєво підвищити швидкість фільтрації зображень. Однак, оскільки, в більшій мірі, зображення є конфіденційною інформацією, передача й подальша обробка їх в незахищеному вигляді є неможливою.

РОЗДІЛ 2

РОЗРОБКА МЕТОДІВ ОБРОБКИ КОНІДЕНЦІЙНОЇ ІНФОРМАЦІЇ З ВИКОРИСТАННЯМ ВІДДАЛЕНИХ ОБЧИСЛЮВАЛЬНИХ РЕСУРСІВ

Однією з найпопулярніших задач в області комп'ютерних обчислень є обробка медіа-контенту. З розвитком інформаційних технологій, ростуть потреби в ресурсах, необхідних для виконання даних операцій [11]. На сьогоднішній день, для ефективної обробки зображень чи відео-матеріалів, необхідно мати потужні обчислювальні пристрої, оскільки процес видозмінення зображення чи відео потребує обробки кожного фрагменту окремо. Придбання дороговартісного обладнання не завжди є доцільним та можливим рішенням. Гарним рішенням у даному випадку може стати перенесення процесу обробки на віддалені обчислювальні ресурси. Основною перешкодою в даному підході є конфіденційних характер даних, що не дозволяє передачу та обробку даних в незахищеному вигляді. Отже, на сьогоднішній день існує потреба у методах обробки зашифрованих медіа-файлів, для використання їх на віддалених ресурсах.

2.1. Розробка методу медіанної фільтрації зашифрованих зображень

Одною з основних проблем, які виникають при роботі з цифровими зображеннями є поява шумів. Одним з найефективніших та найпопулярніших методів для їх видалення є обробка зображення за допомогою медіанного фільтра [12]. Алгоритм фільтрації представляє собою покроковий аналіз кожного пікселя зображення та подальше корегування його інтенсивності за рахунок заміни його значення на медіанне значення його найближчих сусідів. Обробка зображення за допомогою медіанного фільтра не дозволяє відновити

оригінальне, неспотворене зображення, але, безумовно, підвищує його якість за рахунок видалення аномальних фрагментів.

Основною перешкодою при спробі перенести фільтрацію зображень на віддалені обчислювальні ресурси є конфіденційних характер оригінальних даних, і як наслідок, необхідність шифрування зображення перед його передачею на будь-які сторонні ресурси [13]. Одним з варіантів вирішення даної проблеми є винаходження методу фільтрації даних у зашифрованому вигляді. Це дозволить перенести обробку на потужні хмарні ресурси, чим збільшить швидкість фільтрації, за умови, якщо загальна складність алгоритмів шифрування та дешифрування буде меншою за складність алгоритму фільтрації незашифрованої інформації.

Оскільки кожний вид фільтрації представляє собою набір різних операцій, то не може бути знайденого універсального методу для шифрування та дешифрування даних для подальшої їх обробки в захищеному вигляді.

Базовою операцією при виконанні медіанної фільтрації є порівняння елементів множини між собою, оскільки це є єдиним можливим методом для знаходження медіанного значення у довільній вибірці [14]. Відповідно, для виконання фільтрації на віддалених обчислювальних ресурсах, зашифроване зображення повинно зберігати інформацію про порядок оригінальних даних. Оскільки можливі значення пікселів зображення представляють собою кінцеву множину цілих чисел, пропонується у якості рішення перевести множину оригінальних значень у іншу довільну множину чисел зі збереженням порядку елементів.

Для цього кожен елемент множини оригінальних значень помножується на довільне число g, обране з діапазону чисел, порядок якого значно перевищує, діапазон можливих значень пікселів зображення. Операція множення збільшує різницю між елементами множини та зберігає початковий порядок. Після розширення вхідного діапазону, відбувається

його зміщення за рахунок додавання до кожного елемента множини однакового випадкового числа p, обраного з діапазону чисел, порядок якого значно перевищує, діапазон можливих значень пікселів зображення.

Останнім етапом обробки вхідних даних є зміщення кожного елемента множини на довільне значення, яке є унікальним для кожного пікселя, та обирається з діапазону [0; g). Оскільки значення пікселів зображення є елементами множини цілих чисел, то після помноження їх на ціле число g, мінімальна можлива різницею між двома довільними значеннями не може бути меншою за значення g. Це означає, що збільшення кожного елемента множини на довільне число з діапазону [0; g) не приведе до порушення оригінального порядку.

Алгоритм шифрування даних:

1) Обирається довільне число g з діапазону $(0, 2^n - 1)$, де n – розрядність множини вхідних значень.
2) Кожне значення пікселя вхідного зображення множиться на p.
3) Обирається довільне число p з діапазону $(0, 2^n - 1)$, де n – розрядність множини вхідних значень.
4) До кожного значення множини додається число p.
5) До кожного значення множини додається випадкове число діапазону [0; g).
6) Значення секретних ключів шифрування g та p зберігаються для подальшого дешифрування.

Після шифрування данні передаються на віддалені обчислювальні ресурси, де відбувається медіанна фільтрація зображення. Координата кожного зашифрованого пікселя відповідає координаті його оригіналу. Після обробки відфільтровані дані повертаються та розшифровуються за допомогою зворотних перетворень.

Оскільки можливі значення пікселів є кінцевою множиною, то у разі необхідності, може бути згенеровано таблиця відповідності зашифрованих

даних оригінальним. завчасно. Такий метод шифрування має більшу загальну складність, але має значну перевагу у швидкості при можливості проведення підготовчих обрахунків. У такому разі алгоритм шифрування виглядає наступним чином:

1) Обирається довільне число g з діапазону $(0, 2^n)$, де n – розрядність множини вхідних значень.
2) Всі значення з множини можливих значень пікселів вхідного зображення множиться на p.
3) Обирається довільне число p з діапазону $(0, 2^n)$, де n – розрядність множини вхідних значень.
4) До кожного значення множини додається число p.
5) До кожного значення множини додається випадкове число діапазону [0; g).
6) Значення секретних ключів шифрування g та p зберігаються для подальшого дешифрування.

У разі якщо у вхідному зображенні є два чи більше пікселів з однаковим значенням, для кожного с них п.5 повинен бути обрахований окремо.

Розроблений метод шифрування даних відповідає необхідним умовам зазначеним вище.

На Рис.2.1 зображено представлення цифрового зображення у вигляді двомірної матриці, де кожний елемент відповідає значенню яскравості пікселя з відповідними координатами.

32	54	43	2	53
17	6	84	45	3
31	37	74	26	78
16	22	58	53	29
84	24	63	43	43

Рис.2.1. Представлення зображення у вигляді матриці пікселів

При обробці зазначеного вище зображення квадратною апертурою розміру 3x3, для обробки пікселя з координатами (1,1) будуть обрані наступні значення – 32, 54, 43, 17, 6, 84, 31, 74.

32	54	43	2	53
17	6	84	45	3
31	37	74	26	78
16	22	58	53	29
84	24	63	43	43

Рис.2.2. Значення апертури при обробці пікселя з координатами (1,1)

Після обрання даних, які потрапили в апертуру, серед них обирається медіанне значення, яке записується у відфільтроване зображення за координатою (1,1). У даній вибірці медіанним значенням є число 37. На Рис.2.3 зображена апертура відфільтрованого зображення.

32	54	43	2	53
17	**37**	84	45	3
31	37	74	26	78
16	22	58	53	29
84	24	63	43	43

Рис.2.3. Значення апертури після обробки пікселя з координатами (1,1)

Після обробки пікселя з координатами (1,1) апертура зміщується для обробки наступного значення з координатами (1,2). Апертура буде мати вигляд, показаний на рис.2.4.

32	54	43	2	53
17	6	84	45	3
31	37	74	26	78
16	22	58	53	29
84	24	63	43	43

Рис.2.4. Значення апертури при обробці пікселя з координатами (1,2)

Після фільтрації обраних значень, центральне значення апертури буде змінне на 43.

32	54	43	2	53
17	6	**43**	45	3
31	37	74	26	78
16	22	58	53	29
84	24	63	43	43

Рис.2.5. Значення апертури після обробки пікселя з координатами (1,2)

Після обробки усіх вхідних пікселів, зображення буде мати наступний вигляд:

32	54	43	2	53
17	**37**	**43**	**45**	3
31	**31**	**45**	**53**	78
16	**37**	**43**	**53**	29
84	24	63	43	43

Рис.2.6. Представлення зображення у вигляді матриці інтенсивності пікселів після обробки медіанним фільтром

Алгоритм захищеної медіанної фільтрації починається з вибору випадкового числа g. Припустимо, в даному прикладі g рівне 14. Наступним кроком, кожний елемент вхідної матриці множиться на g. Матриця інтенсивності пікселів буде мати наступний вигляд:

448	756	602	28	742
238	84	1176	630	42
434	518	1036	364	1092
224	308	812	742	406
1176	336	882	602	602

Рис.2.7. Представлення зображення у вигляді матриці інтенсивності пікселів після множення на число g = 14

Після розширення числового діапазону, відбувається його лінійне зміщення, за рахунок додавання довільного числа p. Припустимо, в даному прикладі g рівне 325. Після зсуву матриця інтенсивності буде мати наступний вигляд:

773	1081	927	353	1067
563	409	1501	955	367
759	843	1361	689	1417
549	633	1137	1067	731
1501	661	1207	927	927

Рис.2.8. Представлення зображення у вигляді матриці інтенсивності пікселів після додавання числа p = 325

Останнім етапом шифрування зображення відбувається додавання випадкового числа з діапазону [0;g) до кожного значення матриці.

775	1086	931	362	1074
573	413	1512	959	375
767	843	1368	693	1422
555	641	1142	1071	738
1511	671	1217	936	932

Рис.2.9. Представлення зображення у вигляді матриці інтенсивності пікселів після додавання випадкових значень з діапазону [0;g)

Після закінчення шифрування, зображення передається на віддалені обчислювальні ресурси, де відбувається процес фільтрації. Перед відправкою значення секретних ключів шифрування g та p зберігаються для подальшого дешифрування.

Алгоритм фільтрації зображення з використанням хмарних технологій ідентичний оригінальному алгоритму. Після обробки зображення буде мати наступний вигляд:

775	1086	931	362	1074
573	843	931	959	375
767	767	959	1071	1422
555	843	936	1071	738
1511	671	1217	936	932

Рис.2.10. Представлення зашифрованого зображення у вигляді матриці інтенсивності пікселів після віддаленої обробки медіанним фільтром

Оброблене зображення повертається користувачеві, після чого відбувається процес дешифрування за рахунок обернених перетворень. Для відтворення оригінальних значень інтенсивності пікселів необхідно змістити діапазон зашифрованих значень назад, за рахунок віднімання від кожного значення матриці числа p.

450	761	606	37	749
248	518	606	634	50
442	442	634	746	1097
230	518	611	746	413
1186	346	892	611	607

Рис.2.11. Представлення відфільтрованого зображення у вигляді матриці інтенсивності пікселів після віднімання числа p = 325

Після лінійного зсуву відбувається процес зменшення відстані між елементами множини, за допомогою ділення кожного елемента на g та виділення цілої частини числа.

32	54	43	2	53
17	37	43	45	3
31	31	45	53	78
16	37	43	53	29
84	24	63	43	43

Рис.2.12. Представлення відфільтрованого зображення у вигляді матриці інтенсивності пікселів після ділення на число g = 14 та відділення цілої частини

Значення інтенсивності пікселів дешифрованого відфільтрованого зображення, представленого на рис.2.12, повністю співпадають зі значеннями зображення, відфільтрованого оригінальним алгоритмом, рис.2.6.

Представлений вище приклад повністю ілюструє поетапне виконання алгоритму медіанної фільтрації конфіденційних даних за допомогою віддалених обчислювальних ресурсів.

Захищеність зашифрованих даних забезпечується за рахунок значної різниці між розміром діапазону можливих значень інтенсивності пікселів та

діапазоном зашифрованих значень. При виборі значень ключів g та p з проміжку $(0, 2^n)$, де n – розрядність множини вхідних значень, та k – кількість пікселів зображення, складність перебору для відтворення оригінального зображення буде дорівнювати $C_{2^n}^k$, що робить перебір можливих значень зображення неефективною операцією.

Для ефективної обробки зображень за допомогою віддалених обчислювальних ресурсів, необхідно, щоб сумарна складність обчислення шифрування та дешифрування не більшою за складність обчислення медіанної фільтрації зображення. При виконанні попередніх обчислень шифрування та дешифрування зображення може бути зведено до пошуку та підстановці по таблиці відповідності зашифрованих даних оригінальним. При використанні хеш-таблиць для збереження та пошуку, складність представленого алгоритму буде лінійною. В свою чергу, обчислення медіанної фільтрації має квадратичну складність [15]. Таким чином, за допомогою наведеного вище алгоритму може бути проведена ефективна обробка зображень за допомогою віддалених обчислювальних ресурсів.

2.2. Розробка методу середньоарифметичної фільтрації зашифрованих зображень

Одним з ефективних та популярних методів для видалення цифрових шумів є обробка зображення за допомогою середньоарифметичного фільтра. Алгоритм фільтрації представляє собою покроковий аналіз кожного пікселя зображення та подальше корування його інтенсивності за рахунок заміни його значення на середньоарифметичне значення його найближчих сусідів [16]. Обробка зображення за допомогою середньоарифметичного фільтра не дозволяє відновити оригінальне, неспотворене зображення, але, безумовно, підвищує його якість за рахунок видалення аномальних фрагментів.

Базовою операцією при виконанні фільтрації є додавання елементів множини між собою та ділення елементів на константу, яка дорівнює розміру

апертури. Відповідно, для виконання фільтрації на віддалених обчислювальних ресурсах, зашифроване зображення повинно зберігати можливість виконання перелічених операцій.

Операція знаходження середньоарифметичного значення є дистрибутивною операцією. Для будь-яких довільних множин чисел сума середньоарифметичних значень цих множин буде дорівнювати середньоарифметичному значенню об'єднанню цих множин [17].

$$f(A) + f(B) = f(A \cup B),$$

де f – операція знаходження середньоарифметичного значення; A, B – довільні множини чисел;

Виходячи з вище представленої властивості дистрибутивності операції знаходження середньоарифметичного значення, шифрування вхідного зображення може відбуватися за рахунок накладання допоміжної послідовності чисел на оригінальні значення інтенсивності пікселів. Загальний алгоритм шифрування при використанні допоміжної послідовності буде наступний:

1) Генерується допоміжна послідовність чисел, яка за розміром дорівнює кількості пікселів вхідного зображення.
2) Обчислюється відфільтрована допоміжна послідовність, за рахунок виконання середньоарифметичної фільтрації допоміжної послідовності.
3) Кожний елемент допоміжної послідовності додається до відповідного значення оригінального зображення.

Після додавання зашифроване зображення передається до віддалених обчислювальних ресурсів, де відбувається процес середньоарифметичної фільтрації. Після отримання обробленого зашифрованого зображення, оригінальне оброблене зображення може бути відновлено за рахунок віднімання від елементів зашифрованої послідовності відповідних елементів відфільтрованої допоміжної послідовності.

Вибір методу генерації допоміжної послідовності є ключовим етапом розробки алгоритму захищеної середньоарифметичної фільтрації. Стійкість шифрування зображення повністю залежить від кількості операцій необхідних для відтворення секретної послідовності.

Одним з варіантів знаходження секретної послідовності є, безпосередньо, генерація користувачем послідовності випадкових чисел за допомогою власних обчислювальних ресурсів. У такому разі шифрування зображення може бути виконано за допомогою шифру “one-time pad” чи одноразовий блокнот.

OTP - це алгоритм шифрування, який з достовірністю не може бути зламаним [18]. У цьому алгоритмі відкритий текст поєднується з випадковим секретним ключем. Кожний біт або символ відкритого тексту шифрується, за допомогою комбінації його з відповідним бітом або символом секретного ключа. Отримані зашифровані дані буде неможливо розшифрувати, при виконанні наступних чотирьох умов:

1) Ключ шифрування повинен бути згенерований випадково.
2) Розмір ключа повинен дорівнювати або бути більшим розміру відкритих даних.
3) Ключ не може бути використаним повторно повністю або частково.
4) Ключ повинен зберігатися в повній таємниці.

У разі виконання вище перелічених умов шифрування зображення буде мати абсолютну стійкість. Алгоритм при використанні техніки “one-time pad” буде наступним:

1) Генерується випадкова (допоміжна) послідовність чисел, яка за розміром дорівнює кількості пікселів вхідного зображення.
2) Обчислюється відфільтрована допоміжна послідовність, за рахунок виконання середньоарифметичної фільтрації допоміжної послідовності.

3) Кожний елемент допоміжної послідовності додається до відповідного значення оригінального зображення.
4) Після шифрування допоміжна послідовність повинна бути знищена та ніколи не використовуватись повторно.

Після додавання зашифроване зображення передається до віддалених обчислювальних ресурсів, де відбувається процес середньоарифметичної фільтрації. Після отримання обробленого зашифрованого зображення, оригінальне оброблене зображення може бути відновлено за рахунок віднімання від елементів зашифрованої послідовності відповідних елементів відфільтрована допоміжної послідовності.

Приклад виконання оригінального алгоритму середньоарифметичної фільтрації:

На Рис.2.13 зображено представлення цифрового зображення у вигляді двомірної матриці, де кожний елемент відповідає значенню яскравості пікселя з відповідними координатами.

32	54	43	2	53
17	6	84	45	3
31	37	74	26	78
16	22	58	53	29
84	24	63	43	43

Рис.2.13. Представлення зображення у вигляді матриці інтенсивності пікселів

При обробці зазначеного вище зображення квадратною апертурою розміру 3x3, для обробки пікселя з координатами (1,1) будуть обрані наступні значення – 32, 54, 43, 17, 6, 84, 31, 74.

32	54	43	2	53
17	6	84	45	3
31	37	74	26	78
16	22	58	53	29
84	24	63	43	43

Рис.2.14. Значення апертури при обробці пікселя з координатами (1,1)

Після обрання даних, які потрапили в апертуру, серед них обирається середньоарифметичне значення, яке записується у відфільтроване зображення за координатою (1,1). У даній вибірці середнім значенням є число 42. На Рис.2.15 зображена апертура відфільтрованого зображення.

32	54	43	2	53
17	**42**	84	45	3
31	37	74	26	78
16	22	58	53	29
84	24	63	43	43

Рис.2.15. Значення апертури після обробки пікселя з координатами (1,1)

Після обробки пікселя з координатами (1,1) апертура зміщується для обробки наступного значення з координатами (1,2).

32	54	43	2	53
17	6	84	45	3
31	37	74	26	78
16	22	58	53	29
84	24	63	43	43

Рис.2.16. Значення апертури при обробці пікселя з координатами (1,2)

Після фільтрації обраних значень, центральне значення апертури буде змінне на 40. Апертура буде мати вигляд, показаний на рис.2.17.

32	54	43	2	53
17	6	**41**	45	3
31	37	74	26	78
16	22	58	53	29
84	24	63	43	43

Рис.2.17. Значення апертури після обробки пікселя з координатами (1,2)

Після обробки усіх вхідних пікселів, зображення буде мати наступний вигляд:

32	54	43	2	53
17	**42**	**41**	**45**	3
31	**38**	**45**	**50**	78
16	**45**	**44**	**52**	29
84	24	63	43	43

Рис.2.18. Представлення зображення у вигляді матриці інтенсивності пікселів після обробки середньоарифметичним фільтром

Приклад віддаленої середньоарифметичної фільтрації з використанням алгоритму “one-time pad”:

Етап підготовчих обчислень:

Першим кроком відбувається генерація випадкової секретної послідовності.

62	16	89	1	88
72	43	42	50	68
30	12	83	11	56
41	38	88	87	72
13	88	47	92	26

Рис.2.19. Випадкова допоміжна послідовність

Після генерації послідовність оброблюється за допомогою середньоарифметичного фільтра. На цьому закінчується етап попередніх обчислень.

62	16	89	1	88
72	50	39	54	68
30	50	50	62	56
41	49	61	62	72
13	88	47	92	26

Рис.2.20. Відфільтрована допоміжна послідовність

Після надходження зображення, відбувається процес шифрування, який представляє собою додавання елементів зображення до відповідних значень допоміжної послідовності.

32	54	43	2	53
17	6	84	45	3
31	37	74	26	78
16	22	58	53	29
84	24	63	43	43

Рис.2.21. Представлення зображення у вигляді матриці інтенсивності пікселів

94	70	132	3	141
89	49	126	95	71
61	49	157	37	134
57	60	146	140	101
97	112	110	135	69

Рис.2.22. Матриця зашифрованих значень інтенсивності пікселів вхідного зображення

Після шифрування, зображення у захищеному вигляді відправляється на віддалені обчислювальні ресурси, де відбувається подальший процес фільтрації.

94	70	132	3	141
89	92	80	100	71
61	88	95	112	134
57	94	105	114	101
97	112	110	135	69

Рис.2.23. Представлення зашифрованого зображення у вигляді матриці інтенсивності пікселів після віддаленої обробки середньоарифметичним фільтром

Останнім кроком відбувається процес дешифрування відфільтрованого зображення за рахунок віднімання від елементів обробленої послідовності елементів відфільтрованої допоміжної послідовності.

32	54	43	2	53
17	**42**	**41**	**45**	3
31	**38**	**45**	**50**	78
16	**45**	**44**	**52**	29
84	24	63	43	43

Рис.2.24. Представлення відфільтрованого зображення у вигляді матриці інтенсивності пікселів після дешифрування

Представлений вище приклад повністю ілюструє поетапне виконання алгоритму середньоарифметичної фільтрації конфіденційних даних за допомогою віддалених обчислювальних ресурсів.

Алгоритм шифрування зображення з використанням техніки “one-time pad”, має один недолік: обчислювальна складність шифрування та дешифрування перевищує складність оригінального алгоритму середньоарифметичної фільтрації. Однак, оскільки пункт 1, 2 можуть бути обраховані завчасно, наведений алгоритм може використовуватись в системах з нерівномірним навантаженням. У вільних проміжках часу система може займатися генерацією та фільтрацією допоміжних послідовностей, а при надходженні зображень лише шифруванням та дешифруванням зображень, за рахунок додавання та віднімання допоміжних послідовностей. При такому методі використання, алгоритм буде мати лінійну складність *O(2n).*

2.3. Оптимізація методу середньоарифметичної фільтрації зображень

Шифрування зображення за допомогою додавання випадкової послідовності та подальше дешифрування за рахунок віднімання її відфільтрованої версії, дозволяє захищати зображення з великим рівнем стійкості, але потреба в попередній фільтрації допоміжного зображення,

робить даний алгоритм ефективним в дуже вузькому спектрі задач. Рішенням цієї проблеми може стати часткове перенесення підготовчих обчислень на віддалені ресурси.

Етап підготовчих обчислень починається з генерації групи випадкових матриць, розмір яких співпадає з розміром зображень, які будуть фільтруватися в подальшому. Після чого, дана група відправляється на віддалені обчислювальні ресурси, де відбувається середньоарифметична фільтрація кожної матриці. Оброблені значення повертаються користувачеві. Для подальшого шифрування зображення генерується матриця шифрування за допомогою лінійного перетворення:

$$C = g_1 \cdot R_1 + g_2 \cdot R_2 + \cdots + g_k \cdot R_k ,$$

де, g_i – секретний коефіцієнт лінійного перетворення, R_i – матриця з випадковими значеннями елементів.

До кожного значення інтенсивності пікселів оригінального зображення додається відповідний елемент матриці *C*.

Для дешифрування зображення генерується матриця обернених перетворень:

$$D = g_1 \cdot F_1 + g_2 \cdot F_2 + \cdots + g_k \cdot F_k ,$$

де, g_i – секретний коефіцієнт лінійного перетворення, F_i – матриця з відфільтрованими випадковими значеннями елементів.

Процес дешифрування відбувається за рахунок віднімання від значень відфільтрованого зашифрованого зображення відповідних значень матриці *D*.

Таким чином в етапі підготовчих замість фільтрації допоміжної матриці випадкових чисел, користувачеві потрібно згенерувати її за рахунок лінійних перетворень. Даний алгоритм потребує суттєво меншу кількість необхідних обчислювальних витрат для шифрування та дешифрування оригінальних зображень, що дозволяє використовувати його в якості універсального методу для віддаленої середньоарифметичної фільтрації зображень.

Алгоритм шифрування виглядає наступним чином:

Етап підготовчих обчислень:

1) Генерується група випадкових матриць R_1, R_2, …, R_n.
2) За допомогою віддалених обчислювальних ресурсів кожна матриця з групи R_1, R_2, …, R_n оброблюється за допомогою середньоарифметичного фільтра. Отримані значення F_1, F_2, …, F_n повертаються користувачеві.

Етап шифрування зображення:

1) З групи R_1, R_2, …, R_n обирається k матриць довільним чином для подальшої генерації псевдовипадкової матриці шифрування.
2) Генеруються довільні коефіцієнти лінійного перетворення.
3) Генерується псевдовипадкова матриця шифрування C за допомогою згенерованого лінійного перетворення.
4) До кожного значення інтенсивності пікселів оригінального зображення додається відповідний елемент матриці C.

Після шифрування данні передаються на віддалені обчислювальні ресурси, де відбувається середньоарифметична фільтрація зображення. Після обробки відфільтровані дані повертаються та розшифровуються за допомогою зворотного перетворення.

Етап дешифрування зображення:

1) З групи F_1, F_2, …, F_n обирається k відфільтрованих матриць, які відповідають матрицям з групи R_1, R_2, …, R_k .
2) Генерується матриця дешифрування D за допомогою згенерованого лінійного перетворення.
3) Від кожного значення інтенсивності пікселів відфільтрованого зашифрованого зображення віднімається відповідний елемент матриці D.

Приклад виконання алгоритму віддаленої середньоарифметичної фільтрації:

Етап підготовчих обчислень:

Алгоритм підготовчих обчислень починається з генерації групи випадкових матриць R_1, R_2, …, R_n. Для зручності демонстрації візьмемо n рівним k = 4.

62	16	89	1	88
72	43	42	50	68
30	12	83	11	56
41	38	88	87	72
13	88	47	92	26

86	20	35	26	76
84	52	51	85	86
10	47	30	47	88
48	1	23	35	0
56	6	47	72	92

55	76	36	68	63
44	100	56	43	80
58	40	7	14	25
46	11	9	93	75
84	87	28	100	15

55	32	84	50	90
70	7	49	85	43
31	93	68	43	24
71	74	32	87	86
79	95	27	54	73

Рис.2.25. Група довільних зображень R_1, R_2, R_3, R_4

Наступним кроком відбувається віддалена середньоарифметична фільтрація згенерованих зображень R_1, R_2, R_3, R_4

62	16	89	1	88
72	**50**	**39**	**54**	68
30	**50**	**50**	**62**	56
41	**49**	**61**	**62**	72
13	88	47	92	26

86	20	35	26	76
84	**46**	**44**	**58**	86
10	**38**	**41**	**49**	88
48	**30**	**34**	**48**	0
56	6	47	72	92

55	76	36	68	63
44	**52**	**49**	**44**	80
58	**41**	**41**	**45**	25
46	**41**	**43**	**41**	75
84	87	28	100	15

55	32	84	50	90
70	**54**	**57**	**60**	43
31	**55**	**60**	**57**	24
71	**63**	**64**	**55**	86
79	95	27	54	73

Рис.2.26. Група відфільтрованих довільних зображень F_1, F_2, F_3, F_4.

Відфільтровані значення повертаються користувачеві. На цьому закінчується етап підготовчих обчислень.

Етап шифрування зображення:

Першим кроком в етапі шифрування зображення відбувається генерація коефіцієнтів лінійного перетворення. В даному прикладі $g_1 = 1$, $g_2 = 2$, $g_3 = 1$, $g_4 = 4$. Після цього відбувається генерація матриці шифрування за допомогою лінійного перетворення:

$$C = 1 \cdot R_1 + 2 \cdot R_2 + 3 \cdot R_3 + 4 \cdot R_4.$$

Результуюча матриця буде мати наступний вигляд:

619	412	603	457	789
652	475	508	689	652
348	598	436	319	403
559	369	289	784	641
693	741	333	752	547

Рис.2.27. Матриця шифрування C

Після генерації матриці шифрування відбувається її додавання до оригінальних значень зображення. Після шифрування матриця буде мати наступний вигляд:

651	466	646	459	842
669	481	592	734	655
379	635	510	345	481
575	391	347	837	670
777	765	396	795	590

Рис.2.28. Матриця зашифрованих значень інтенсивності пікселів вхідного зображення

Після закінчення шифрування, зображення передається на віддалені обчислювальні ресурси, де відбувається процес фільтрації. Алгоритм фільтрації зображення з використанням хмарних технологій ідентичний оригінальному алгоритму. Після обробки зображення буде мати наступний вигляд:

651	466	646	459	842
669	**559**	**541**	**585**	655
379	**509**	**541**	**575**	481
575	**531**	**558**	**552**	670
777	765	396	795	590

Рис.2.29. Представлення зашифрованого зображення у вигляді матриці інтенсивності пікселів після віддаленої обробки середньоарифметичним фільтром

Етап шифрування зображення:

Першим кроком дешифрування зображення відбувається генерація матриці дешифрування: $D = 1 \cdot F_1 + 2 \cdot F_2 + 3 \cdot F_3 + 4 \cdot F_4$.

Матриця дешифрування буде мати наступний вигляд:

619	412	603	457	789
652	517	500	540	652
348	470	496	525	403
559	485	513	500	641
693	741	333	752	547

Рис.2.30. Матриця шифрування *D*

Процес дешифрування відбувається за рахунок віднімання від зашифрованого відфільтрованого зображення матриці обернених перетворень. Дешифрована матриця буде мати наступний вигляд:

32	54	43	2	53
17	**42**	**41**	**45**	3
31	**38**	**45**	**50**	78
16	**45**	**44**	**52**	29
84	24	63	43	43

Рис.2.31 Представлення відфільтрованого зображення у вигляді матриці інтенсивності пікселів після дешифрування

Значення інтенсивності пікселів дешифрованого відфільтрованого зображення, представленого на рис.2.18, повністю співпадають зі значеннями зображення, відфільтрованого оригінальним алгоритмом, рис.2.31.

Представлений вище приклад повністю ілюструє поетапне виконання алгоритму середньоарифметичної фільтрації конфіденційних даних за допомогою віддалених обчислювальних ресурсів.

Розроблений метод шифрування даних відповідає необхідним умовам швидкодії та стійкості. Обчислювальна складність виконання лінійного перетворення з матрицями дорівнює O(*N*2k*). Оскільки для роботи алгоритму

потрібно згенерувати дві допоміжні матриці, загальна складність шифрування та дешифрування дорівнює O($N*4k$). Складність оригінального алгоритму середньоарифметичної фільтрації дорівнює O($N*r^2$), де r – розмір апертури фільтрування. Для ефективного виконання віддаленої середньоарифметичної фільтрації $4k$ повинно бути меншою за r^2. У разі, якщо $k = r$, алгоритм віддаленої обробки зображень буде ефективний для $r > 3$.

Для злому представленого алгоритму шифрування зловмиснику необхідно підібрати усі значення відфільтрованих допоміжних зображень, а також коефіцієнти лінійного перетворення. Для цього потрібно перебрати $C_{n\cdot 2^u}^k$, де u розрядність коефіцієнтів лінійного перетворення. При використанні розрядності рівної u = 32, підбір необхідних значень стає неефективним. *O(2n).*

2.4. Розробка методу одночасної середньоарифметичної фільтрації декількох зображень

При обробці зображень з метою видалення цифрових шумів часто виникає потреба в одночасній обробці великої кількості пошкоджених зображень [19]. Виходячи з цієї потреби, існує доцільність розробки методу середньоарифметичної фільтрації, який буде оптимізований для одночасної обробки групи зображень. Для ефективної роботи, кількість обчислювальних ресурсів необхідних для виконання групової фільтрації повинна бути меншою за сумарну кількість ресурсів необхідну для виконання віддаленої обробки кожного зображення окремо.

Метод одночасної середньоарифметичної фільтрації декількох зображень полягає в наступному:

Перед початком обробки зображень відбувається процес попередніх обчислень. Першим кроком генерується група випадкових секретних послідовностей $R_1, R_2, \ldots, R_n$, які в подальшому будуть використовуватись для шифрування оригінальних зображень, шляхом додавання елемента

послідовності до відповідного елемента зображення. Наступним кроком відбувається обробка згенерованих послідовностей за допомогою середньоарифметичного фільтра. Після фільтрації група відфільтрованих послідовностей F_1, F_2, …, F_n зберігається для подальшого використання в дешифруванні оригінальних зображень. На цьому етап попередніх обчислень закінчується.

При надходженні групи пошкоджених зображень I_1, I_2, …, I_k, відбувається перший етам шифрування, який полягає в виборі k довільних випадкових послідовностей R_1, R_2, …, R_k, та подальшого додавання елементів послідовностей до відповідних значень інтенсивності пікселів зображення. Після першого етапу маємо групу частково зашифрованих зображень X_1, X_2, …, X_k.

Другим кроком шифрування є процес перемішування зображень між собою за рахунок лінійних оборотних перетворень [20]. Для виконання перемішування генеруються дві матриці коефіцієнтів систем лінійних перетворень M та P таким чином, щоб для будь-якого довільного вектору V було вірним наступні твердження:

$$c_1 = m_{11} \cdot v_1 + m_{11} \cdot v_2 + \cdots + m_{1k} \cdot v_k$$
$$c_2 = m_{12} \cdot v_1 + m_{22} \cdot v_2 + \cdots + m_{2k} \cdot v_k$$
$$\cdots$$
$$c_k = m_{k1} \cdot v_1 + m_{k2} \cdot v_2 + \cdots + m_{kk} \cdot v_k$$

$$v_1 = p_{11} \cdot c_1 + p_{11} \cdot c_2 + \cdots + p_{1k} \cdot c_k$$
$$v_2 = p_{12} \cdot c_1 + p_{22} \cdot c_2 + \cdots + p_{2k} \cdot c_k$$
$$\cdots$$
$$v_k = p_{k1} \cdot c_1 + p_{k2} \cdot c_2 + \cdots + p_{kk} \cdot c_k$$

де v_i – елементи довільного вектору v, p_{ij} – коефіцієнти системи лінійних перетворень p, m_{ij} – коефіцієнти системи обернених лінійних перетворень m.

Приклад виконання зворотних лінійних перетворень:

$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$ – матриця коефіцієнтів системи лінійних перетворень *M*;

$\begin{bmatrix} -1 & 1 & 1 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 0 & -1 & 1 \\ 1 & -1 & 0 & 0 \end{bmatrix}$ – матриця коефіцієнтів оберненої системи лінійних перетворень *P*;

$[1 \quad 2 \quad 3 \quad 4]$ – вектор довільних значень *V*;

Зашифровані значення вектора будуть обраховуватись наступним чином:

$$c_1 = m_{11} \cdot v_1 + m_{12} \cdot v_2 + m_{13} \cdot v_3 + m_{14} \cdot v_4 = 1 + 2 + 3 + 4 = 10$$

$$c_2 = m_{21} \cdot v_1 + m_{22} \cdot v_2 + m_{23} \cdot v_3 + m_{24} \cdot v_4 = 1 + 2 + 3 = 6$$

$$c_3 = m_{31} \cdot v_3 + m_{32} \cdot v_3 + m_{33} \cdot v_3 + m_{34} \cdot v_4 = 1 + 4 = 5$$

$$c_4 = m_{41} \cdot v_4 + m_{42} \cdot v_4 + m_{43} \cdot v_4 + m_{44} \cdot v_4 = 1 + 3 + 4 = 8$$

В результаті виконання лінійних перетворень над вектором *V*, маємо зашифрований вектор *C*. Процес дешифрування відбувається за рахунок зворотних лінійних перетворень вектора *C*.

$$v_1 = p_{11} \cdot c_1 + p_{12} \cdot c_2 + p_{13} \cdot c_3 + p_{14} \cdot c_4 = -10 + 6 + 5 = 1$$

$$v_2 = p_{21} \cdot c_1 + p_{22} \cdot c_2 + p_{23} \cdot c_3 + p_{24} \cdot c_4 = 10 - 8 = 2$$

$$v_3 = p_{31} \cdot c_3 + p_{32} \cdot c_3 + p_{33} \cdot c_3 + p_{34} \cdot c_4 = -5 + 8 = 3$$

$$v_4 = p_{41} \cdot c_4 + p_{42} \cdot c_4 + p_{43} \cdot c_4 + p_{44} \cdot c_4 = 10 - 6 = 4$$

$[1 \quad 2 \quad 3 \quad 4]$ – вектор *C* після зворотних лінійних перетворень. Значення отримані після перетворень повністю співпадають з оригінальними значеннями вектора *V*.

Після додавання до оригінальних значень інтенсивності пікселів зображень секретних випадкових послідовностей, відбувається перемішування отриманих зашифрованих зображень між собою за допомогою лінійних перетворень системи *m*.

$$c_{11} = m_{11} \cdot x_{11} + m_{12} \cdot x_{21} + m_{13} \cdot x_{31} + \cdots + m_{1k} \cdot x_{k1}$$
$$c_{12} = m_{11} \cdot x_{12} + m_{12} \cdot x_{22} + m_{13} \cdot x_{32} + \cdots + m_{1k} \cdot x_{k2}$$
$$c_{13} = m_{11} \cdot x_{13} + m_{12} \cdot x_{23} + m_{13} \cdot x_{33} + \cdots + m_{1k} \cdot x_{k3}$$
$$\cdots$$
$$c_{1n} = m_{11} \cdot x_{1n} + m_{12} \cdot x_{2n} + m_{13} \cdot x_{3n} + \cdots + m_{1k} \cdot x_{kn}$$

$$c_{21} = m_{21} \cdot x_{11} + m_{22} \cdot x_{21} + m_{23} \cdot x_{31} + \cdots + m_{2k} \cdot x_{k1}$$
$$c_{22} = m_{21} \cdot x_{12} + m_{22} \cdot x_{22} + m_{23} \cdot x_{32} + \cdots + m_{2k} \cdot x_{k2}$$
$$c_{23} = m_{21} \cdot x_{13} + m_{22} \cdot x_{23} + m_{23} \cdot x_{33} + \cdots + m_{2k} \cdot x_{k3}$$
$$\cdots$$
$$c_{2n} = m_{21} \cdot x_{1n} + m_{22} \cdot x_{2n} + m_{23} \cdot x_{3n} + \cdots + m_{2k} \cdot x_{kn}$$

$$c_{k1} = m_{k1} \cdot x_{12} + m_{k2} \cdot x_{21} + m_{k3} \cdot x_{31} + \cdots + m_{kk} \cdot x_{k1}$$
$$c_{k2} = m_{k1} \cdot x_{12} + m_{k2} \cdot x_{22} + m_{k3} \cdot x_{32} + \cdots + m_{kk} \cdot x_{k2}$$
$$c_{k3} = m_{k1} \cdot x_{13} + m_{k2} \cdot x_{23} + m_{k3} \cdot x_{33} + \cdots + m_{kk} \cdot x_{k3}$$
$$\cdots$$
$$c_{kn} = m_{k1} \cdot x_{1n} + m_{k2} \cdot x_{2n} + m_{k3} \cdot x_{3n} + \cdots + m_{kk} \cdot x_{kn}$$

де, k – порядковий номер зображення; n – кількість пікселів в зображені; m_{ij} – елемент матриці перетворень M з координатами $[i, j]$; x_{ki} – піксель зображення X_k за індексом i; c_{ki} – піксель зображення C_k за індексом i.

Процес дешифрування зображень за рахунок обернених перетворень.

$$d_{11} = p_{11} \cdot c_{11} + p_{12} \cdot c_{21} + p_{13} \cdot c_{31} + \cdots + p_{1k} \cdot c_{k1}$$
$$d_{12} = p_{11} \cdot c_{12} + p_{12} \cdot c_{22} + p_{13} \cdot c_{32} + \cdots + p_{1k} \cdot c_{k2}$$
$$d_{13} = p_{11} \cdot c_{13} + p_{12} \cdot c_{23} + p_{13} \cdot c_{33} + \cdots + p_{1k} \cdot c_{k3}$$
$$\cdots$$
$$d_{1n} = p_{11} \cdot c_{1n} + p_{12} \cdot c_{2n} + p_{13} \cdot c_{3n} + \cdots + p_{1k} \cdot c_{kn}$$

$$d_{21} = p_{21} \cdot c_{11} + p_{22} \cdot c_{21} + p_{23} \cdot c_{31} + \cdots + p_{2k} \cdot c_{k1}$$
$$d_{22} = p_{21} \cdot c_{12} + p_{22} \cdot c_{22} + p_{23} \cdot c_{32} + \cdots + p_{2k} \cdot c_{k2}$$
$$d_{23} = p_{21} \cdot c_{13} + p_{22} \cdot c_{23} + p_{23} \cdot c_{33} + \cdots + p_{2k} \cdot c_{k3}$$
$$\cdots$$
$$d_{2n} = p_{21} \cdot c_{1n} + p_{22} \cdot c_{2n} + p_{23} \cdot c_{3n} + \cdots + p_{2k} \cdot c_{kn}$$

$$d_{k1} = p_{k1} \cdot c_{12} + p_{k2} \cdot c_{21} + p_{k3} \cdot c_{31} + \cdots + p_{kk} \cdot c_{k1}$$
$$d_{k2} = p_{k1} \cdot c_{12} + p_{k2} \cdot c_{22} + p_{k3} \cdot c_{32} + \cdots + p_{kk} \cdot c_{k2}$$
$$d_{k3} = p_{k1} \cdot c_{13} + p_{k2} \cdot c_{23} + p_{k3} \cdot c_{33} + \cdots + p_{kk} \cdot c_{k3}$$
$$\cdots$$
$$d_{kn} = p_{k1} \cdot c_{1n} + p_{k2} \cdot c_{2n} + p_{k3} \cdot c_{3n} + \cdots + p_{kk} \cdot c_{kn}$$

де, k – порядковий номер зображення; n – кількість пікселів в зображені; c_{ki} – піксель зображення C_k за індексом i; d_{ki} – піксель дешифрованого зображення D_k за індексом i.

Етап попередніх обчислень алгоритму одночасної середньоарифметичної фільтрації декількох зображень наступний:

1) Генерується група зображень з випадковими значеннями інтенсивності пікселів R_1, R_2, …, R_n.
2) Обчислюється група відфільтрованих випадкових зображень F_1, F_2, …, F_n, шляхом обробки кожного зображення з групи R_1, R_2, …, R_n середньоарифметичним фільтром.
3) Генеруються матриці коефіцієнтів систем обернених лінійних перетворень M та P.

Етап шифрування групи зображень для подальшої віддаленої обробки середньоарифметичним фільтром:

1) Обирається група пошкоджених зображень I_1, I_2, …, I_k для подальшої фільтрації.

2) З групи згенерованих довільних зображень R_1, R_2, …, R_n довільним чином обирається підгрупа зображень R_1, R_2, …, R_k.
3) Кожне зображення з групи I_1, I_2, …, I_k попіксельно додається до відповідного зображення з групи R_1, R_2, …, R_k. В результаті даної операції формується група зображень X_1, X_2, …, X_k.
4) Зображення з групи X_1, X_2, …, X_k перемішуються за допомогою лінійних перетворень системи M. В результаті формується група зашифрованих зображень C_1, C_2, …, C_k.

Після шифрування група зображень C_1, C_2, …, C_k відправляється до віддалених обчислювальних ресурсів, де відбувається процес середньоарифметичної фільтрації. Після обробки відфільтровані дані повертаються та розшифровуються за допомогою зворотних перетворень.

Етап дешифрування групи відфільтрованих зашифрованих зображень

1) Група зображень S_1, S_2, …, S_k оброблюється за допомогою системи зворотних лінійних перетворень P.
2) Від кожного зображення попіксельно віднімається відповідне зображення з групи F_1, F_2, …, F_n.

В результаті дешифрування формується група відфільтрованих оригінальних зображень.

Приклад одночасної віддаленої середньоарифметичної фільтрації декількох зображень.

Етап підготовчих обчислень:

Генерується група зображень з випадковими значеннями інтенсивності пікселів R_1, R_2, …, R_n. Для зручності демонстрації візьмемо n рівним k = 4.

62	16	89	1	88
72	43	42	50	68
30	12	83	11	56
41	38	88	87	72
13	88	47	92	26

86	20	35	26	76
84	52	51	85	86
10	47	30	47	88
48	1	23	35	0
56	6	47	72	92

55	76	36	68	63
44	100	56	43	80
58	40	7	14	25
46	11	9	93	75
84	87	28	100	15

55	32	84	50	90
70	7	49	85	43
31	93	68	43	24
71	74	32	87	86
79	95	27	54	73

Рис.2.32. Група довільних зображень R_1, R_2, R_3, R_4

Наступним кроком відбувається середньоарифметична фільтрація згенерованих зображень R_1, R_2, R_3, R_4

62	16	89	1	88
72	**50**	**39**	**54**	68
30	**50**	**50**	**62**	56
41	**49**	**61**	**62**	72
13	88	47	92	26

86	20	35	26	76
84	**46**	**44**	**58**	86
10	**38**	**41**	**49**	88
48	**30**	**34**	**48**	0
56	6	47	72	92

55	76	36	68	63
44	**52**	**49**	**44**	80
58	**41**	**41**	**45**	25
46	**41**	**43**	**41**	75
84	87	28	100	15

55	32	84	50	90
70	**54**	**57**	**60**	43
31	**55**	**60**	**57**	24
71	**63**	**64**	**55**	86
79	95	27	54	73

Рис.2.33. Група відфільтрованих довільних зображень F_1, F_2, F_3, F_4.

Останнім кроком в етапі передобчислень відбувається генерація матриць коефіцієнтів систем зворотніх лінійних перетворень.

$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$ – матриця коефіцієнтів системи лінійних перетворень M;

$\begin{bmatrix} -1 & 1 & 1 & 0 \\ 1 & 0 & 0 & -1 \\ 0 & 0 & -1 & 1 \\ 1 & -1 & 0 & 0 \end{bmatrix}$ – матриця коефіцієнтів оберненої системи лінійних перетворень P;

Після генерації матриць обернених перетворень закінчується підготовчий етап обчислень.

Етап шифрування групи зображень:

Першим кроком обирається група пошкоджених зображень I_1, I_2, I_3, I_4 для подальшої фільтрації

32	54	43	2	53
17	6	84	45	3
31	37	74	26	78
16	22	58	53	29
84	24	63	43	43

38	72	75	91	55
69	100	83	88	9
70	85	47	35	46
9	96	38	3	30
75	38	51	47	65

58	33	57	59	72
68	100	17	25	5
17	94	3	67	64
37	13	78	97	80
62	12	46	94	94

51	37	32	19	73
89	60	29	100	36
6	33	64	22	87
28	54	11	48	95
1	8	20	78	39

Рис.2.34. Група оригінальних зображень I_1, I_2, I_3, I_4

Наступним кроком до кожного зображення з групи I_1, I_2, I_3, I_4 попіксельно додається відповідне зображення з групи R_1, R_2, R_3, R_4. Після виконання даної операції зображення мають наступний вигляд:

94	70	132	3	141
89	49	126	95	71
61	49	157	37	134
57	60	146	140	101
97	112	110	135	69

124	92	110	117	131
153	152	134	173	95
80	132	77	82	134
57	97	61	38	30
131	44	98	119	157

113	109	93	127	135
112	200	73	68	85
75	134	10	81	89
83	24	87	190	155
146	99	74	194	109

106	69	116	69	163
159	67	78	185	79
37	126	132	65	111
99	128	43	135	181
80	103	47	132	112

Рис.2.35. Група зображень I_1, I_2, I_3, I_4 зашифрована за допомогою накладання випадкових послідовностей

Наступним кроком відбувається процес перемішування зображень за рахунок системи лінійних перетворень *M*. Зашифровані зображення будуть мати наступний вигляд:

437	340	451	316	570
513	468	411	521	330
253	441	376	265	468
296	309	337	503	467
454	385	329	580	447

331	271	335	247	407
354	401	333	336	251
216	315	244	200	357
197	181	294	368	286
374	255	282	448	335

200	139	248	72	304
248	116	204	280	150
98	175	289	102	245
156	188	189	275	282
177	215	157	267	181

313	248	341	199	439
360	316	277	348	235
173	309	299	183	334
239	212	276	465	437
323	314	231	461	290

Рис.2.36. Група зашифрованих зображень C_1, C_2, …, C_k

Після закінчення шифрування, група зображенн передається на віддалені обчислювальні ресурси, де відбувається процес фільтрації кожного зображення.

Етап дешифрування групи зображень:

Після обробки група відфільтрованих зображень буде мати наступні значення:

437	340	451	316	570
513	**410**	**399**	**412**	330
253	**378**	**403**	**409**	468
296	**350**	**389**	**419**	467
454	385	329	580	447

331	271	335	247	407
354	**311**	**298**	**301**	251
216	**282**	**297**	**297**	357
197	**262**	**287**	**313**	286
374	255	282	448	335

200	139	248	72	304
248	**248**	**191**	**181**	150
98	**185**	**202**	**224**	245
156	**183**	**206**	**221**	282
177	215	157	267	181

313	248	341	199	439
360	**293**	**280**	**295**	235
173	**273**	**292**	**317**	334
239	**264**	**306**	**331**	437
323	314	231	461	290

Рис.2.35. Група відфільтрованих зашифрованих зображень S_1, S_2, S_3, S_4

Першим етапом дешифрування відбувається перемішування отриманих зображень за допомогою зворотних лінійних перетворень. Після перемішування зображення матимуть наступний вигляд:

94	70	132	3	141
89	92	80	100	71
61	88	95	112	134
57	94	105	114	101
97	112	110	135	69

124	92	110	117	131
153	117	119	117	95
80	105	105	92	134
57	86	83	88	30
131	44	98	119	157

113	109	93	127	135
112	102	99	85	85
75	89	96	93	89
83	81	99	110	155
146	99	74	194	109

106	69	116	69	163
159	99	101	111	79
37	97	107	112	111
99	88	101	106	181
80	103	47	132	112

Рис.2.36 Група відфільтрованих зашифрованих зображень S_1, S_2, S_3, S_4 після зворотних перемішувань

Останнім етапом дешифрування зображень відбувається процес віднімання від групи отриманих зображень групи відфільтрованих випадкових зображень F_1, F_2, F_3, F_4. Дешифровані зображення мають наступний вигляд:

32	54	43	2	53
17	42	41	45	3
31	38	45	50	78
16	45	44	52	29
84	24	63	43	43

38	72	75	91	55
69	71	75	59	9
70	66	64	42	46
9	57	49	40	30
75	38	51	47	65

58	33	57	59	72
68	50	51	41	5
17	47	55	48	64
37	40	56	69	80
62	12	46	94	94

51	37	32	19	73
89	45	44	51	36
6	42	47	55	87
28	25	38	52	95
1	8	20	78	39

Рис.2.37 Група відфільтрованих дешифрованих зображень

Значення інтенсивності пікселів дешифрованої групи відфільтрованих зображення, повністю співпадають зі значеннями групи зображень, відфільтрованих оригінальним алгоритмом.

Представлений вище приклад повністю ілюструє поетапне виконання алгоритму середньоарифметичної фільтрації групи конфіденційних зображень за допомогою віддалених обчислювальних ресурсів.

Розроблений метод шифрування даних відповідає необхідним умовам швидкодії та стійкості. Обчислювальна складність додавання випадкових допоміжних матриць до оригінальних зображень дорівнює O($N \cdot 2$), оскільки дана операція виконується при шифруванні та дешифруванні. Сумарна складність обчислень накладання послідовностей шифрування на групу зображень дорівнює O($N \cdot 2k$). Складність перемішувань зображень за допомогою лінійних перетворень - O($N \cdot 2k^2$). Загальна складність шифрування та дешифрування групи зображень дорівнює O($N \cdot 2k \cdot (k+1)$).

Для дешифрування значень оригінальних зображень зловмиснику необхідно підібрати значення випадково згенерованих послідовностей, а також систем обернених лінійних перетворень. Кількість комбінацій необхідних для підбору k довільних зображень дорівнює 2^{N*n}. Кількість комбінацій необхідних для підбору матриць обернених перетворень дорівнює $\prod_{i=1}^{k}(2^k - 2^{i-1} - k)$. Загальна кількість варіантів для перебору секретних значень дорівнює $2^{N*n} * \prod_{i=1}^{k}(2^k - 2^{i-1} - k)$, унеможливлює перебір та злом алгоритму при використанні зображень з кількості пікселів більшою за 1024, та кількості зображень в групі $k > 4$.

Висновки до розділу 2

В результаті проведеного аналізу та розробки методів для обробки конфіденційних даних в використанням віддалених обчислювальних ресурсів можна зробити наступні висновки:

1. Розроблений метод медіанної фільтрації зображень на віддалених обчислювальних ресурсах. Метод відповідає переліченим необхідним критеріям ефективності, адже має достатню надійність, яка забезпечуються складністю перебору можливих значень ключів шифрування для відтворення оригінального зображення, та, оскільки, необхідна сумарна кількість обчислювальних ресурсів для шифрування та дешифрування зображення не перевищує кількість ресурсів необхідну для виконання оригінального алгоритму медіанної фільтрації.

2. Розроблений абсолютно стійкий метод середньоарифметичної фільтрації зображень за допомогою віддалених обчислювальних ресурсів. Метод частково відповідає зазначеним критеріям ефективності, адже складність виконання шифрування та дешифрування зображень перевищує складність оригінального алгоритму середньоарифметичної фільтрації. Однак, не дивлячись на зазначений недолік, розроблений метод може використовуватись в системах з нерівномірним навантаженням та покращувати загальну ефективність їх роботи за рахунок виконання попередніх обчислень.

3. Розроблений метод середньоарифметичної фільтрації зображень за допомогою віддалених обчислювальних ресурсів. Прискорення алгоритму досягається за рахунок перенесення частини підготовчих обчислень на віддалені ресурси. Метод відповідає переліченим необхідним критеріям ефективності, адже має достатню надійність, яка забезпечуються складністю перебору можливих значень ключів шифрування для відтворення оригінального зображення, та, оскільки, необхідна сумарна кількість обчислювальних ресурсів для шифрування та дешифрування зображення не

перевищує кількість ресурсів необхідну для виконання оригінального алгоритму фільтрації.

4. Розроблений метод одночасної середньоарифметичної фільтрації декількох зображень за допомогою віддалених обчислювальних ресурсів. Метод відповідає переліченим необхідним критеріям ефективності, адже має достатню надійність, яка забезпечуються складністю перебору можливих значень ключів шифрування для відтворення оригінального зображення, та, оскільки, кількість обчислювальних ресурсів необхідних для виконання групової фільтрації менша за сумарну кількість ресурсів необхідну для виконання віддаленої обробки кожного зображення окремо.

РОЗДІЛ 3

РЕАЛІЗАЦІЯ СИСТЕМИ ВІДДАЛЕНОЇ ЗАХИЩЕНОЇ МЕДІАННОЇ ТА СЕРЕДНЬОАРИФМЕТИЧНОЇ ФІЛЬТРАЦІЇ

3.1. Аналіз технологій для створення програмної системи

Перед початком розробки програмного рішення для реалізації алгоритмів віддаленої медіанної та середньоарифметичної фільтрації, був проведений аналіз існуючих технології для створення програмних систем. Основними критеріями при виборі технології були: можливість роботи із зображеннями в матричному представлені, зручність та швидкодія виконання матричних операцій. В результаті проведеного аналізу було обрано мову програмування C# та фреймворк .Net для подальшої розробки програмної системи.

Мова програмування C# дозволяє опрацьовувати зображення за допомогою бібліотеки System.Drawing. Ключовими компонентами для обробки цифрових зображень є Image, Bitmap та Color.

Клас Image надає можливість завантажувати зображення з файлу, отримувати інформацію про зображення, виконувати прості операції, як поворот та віддзеркалення зображення, малювати зображення за допомогою об'єкта Graphics, зберегти зображення

Клас Bitmap розширює клас Image та надає можливість працювати з растровим представленням зображень. Використання класу Bitmap для представлення зображення у пам’яті дозволяє проводити операції з окремими пікселями даного зображення. Кожен піксель зображення представляється у вигляді екземпляра структури Color.

Колір кожного пікселя представляється у вигляді 32-бітового числа: по 8 бітів для альфа-червоного, зеленого та синього (ARGB). Кожен з чотирьох компонентів - це число від 0 до 255, де 0 не означає відсутність кольору, а 255

– повної його інтенсивності. Альфа-компонент визначає прозорість кольору: 0 повністю прозорий, а 255 повністю непрозорий. Щоб визначити альфа-, червоний, зелений або синій компонент кольору, використовуйте змінні A, R, G або B відповідно.

Мова програмування C# дозволяє описувати матричні операції за рахунок можливості переназначення основних арифметичних операції для власних типів даних. Таким чином, використання даної технології, дозволяє зменшити кількість потенціальних помилок, за рахунок підвищення читабельності вихідного коду.

3.2. Аналіз кольорових моделей представлення цифрового зображення

На сьогоднішній день, існує багато моделей представлення цифрового кольорового зображення у вигляді даних [21]. В ході виконання роботи, був проведений аналіз для вибору представлення зображення з метою його подальшої медіанної та середньоарифметичної фільтрації. Основними кольоровими моделями для роботи з зображеннями є RGB та HSB.

RGB – це кольорова модель представлення зображень [22], колір пікселя в якій формується за рахунок поєднання червоного, зеленого та синього спектрів.

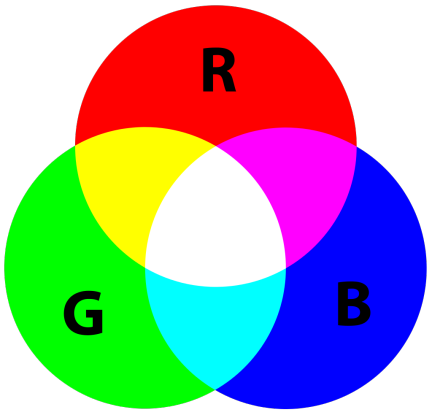


Рис.3.1. Три основні кольори кольорової моделі RGB (червоний, зелений, синій) та їх поєднання

Щоб сформувати колір за допомогою RGB, слід накласти три промені світла: червоний, зелений та синій. Кожен з трьох пучків є компонентом цього кольору, і кожен з них може мати довільне значення інтенсивності. Кольорова модель RGB є адитивною в тому сенсі, що три промені світла складаються разом, а їх світлові спектри додають довжину хвилі, яка дорівнює спектру результуючого кольору.

Нульова інтенсивність для кожного компонента дає найтемніше значення кольору, а максимальне значення – найсвітліше. Результатом поєднання максимальних значень інтенсивності усіх компонент зображення є білий колір.

Цифрове зображення у моделі RGB являє собою поєднання матриць інтенсивності кожного кольору.

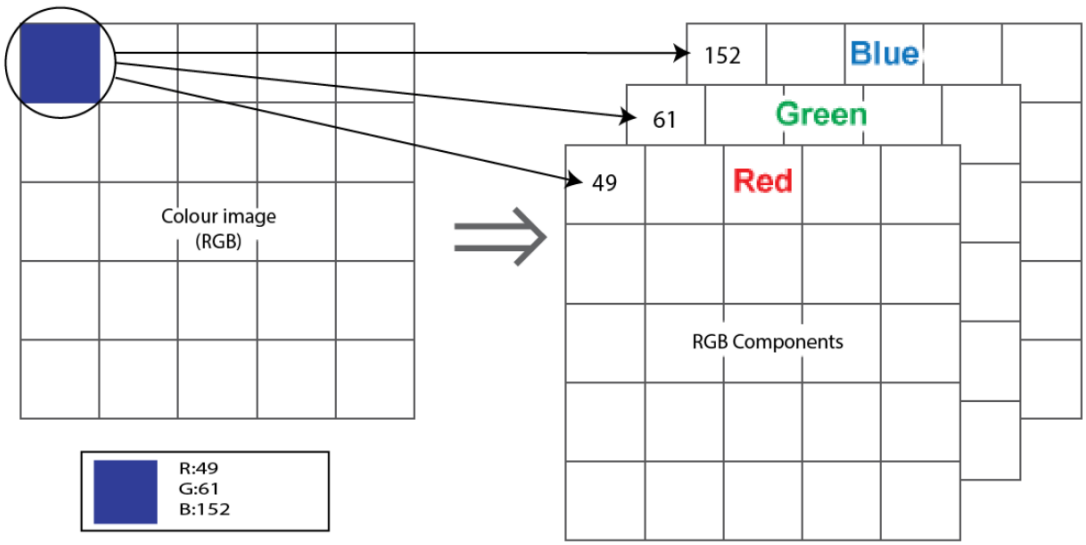


Рис.3.2. Представлення кольорового RGB зображення у вигляді трьох матриць інтенсивності

При використанні такого представлення для виконання медіанної та середньоарифметичної фільтрації зображення розкладається три матриці інтенсивності, після чого відбувається фільтрація кожної матриці окремо. Такий метод обробки зображень призводить до того, що після поєднання відфільтрованих компонент в зображенні можуть з’явитися аномальні значення кольорів. Таким чином, кольорова модель представлення

зображення RGB, не рекомендується для використання для обробки зображення за допомогою медіанного чи середньоарифметичного фільтру.

HSB – кольорова модель, яка складається з трьох компонентів [23]:

- Hue – тон зображення, визначає кут кольору на кольоровому колі RGB. Відтінок 0° дає червоний колір, 120° - зелений, 240° - синій.
- Saturation – насиченість кольору, контролює кількість використовуваного кольору. Колір із 100% насиченістю буде найчистішим із можливих кольорів, тоді як 0% насиченості дає сірі відтінки.
- Brightness – яскравість, колір із яскравістю 0% - це чорний колір, тоді як колір із яскравістю 100% не містить чорного кольору зовсім.

Важливо зазначити, що три виміри кольорової моделі HSV взаємозалежні [24]. Якщо значення яскравості кольору встановлено на 0%, величина відтінку та насиченості не має значення, оскільки колір буде чорним. Аналогічно, якщо насиченість кольору встановлена на 0%, відтінок не має значення, оскільки жоден колір не використовується. Оскільки величина відтінку визначаються в кутах, кольорову модель HSB найкраще зобразити у вигляді циліндра.

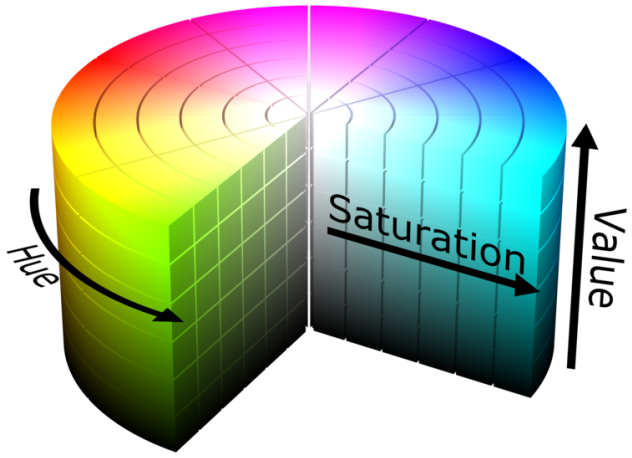


Рис.3.3. Три основні компоненти кольорової моделі HVB (відтінок, насиченість, яскравість) та їх поєднання

При використанні такого представлення для виконання медіанної та середньоарифметичної фільтрації зображення розкладається три матриці значень відповідних компонентів зображення, після чого відбувається фільтрація кожної матриці окремо. Використання даного цифрового представлення дозволяє позбутися проблеми виникнення аномальних значень кольорів, оскільки відсутня процедура генерації відтінку зображення при комбінації відфільтрованих компонентів зображення.

3.3. Розробка системи віддаленої групової середньоарифметичної фільтрації

В результаті виконання даної роботи було створено програмний засіб для здійснення віддаленої групової фільтрації зображень. Розроблена система складається з наступних компонентів:

1) Matrix – клас, в якому представлений необхідний набір операцій для обробки значень представлених у матричній формі: fillRandomValues – метод, який дозволяє ініціалізувати матрицю випадковими значеннями, clone – метод для клонування значень матриці, toString – метод для серіалізації матриці у строковий вигляд, а також визначені оператори додавання, віднімання матриць та множення матриці на число.
2) HSB – клас, в який дозволяє конвертувати значення кольору пікселя з формату RGB в формат HSB та навпаки за допомогою наступних методів: ColorToHSB – метод, конвертує екземпляр класу Color, що представляє значення пікселя в форматі RGB, в формат HSB, для подальшої обробки за допомогою фільтрів, ColorFromHSB – метод, який конвертує представлення значення пікселя у форматі HVB в формат RGB.
3) Image – клас, який є представленням зображення у форматі HSB. Зображення розкладається на основні компоненти кольорової моделі

зберігається у трьох змінних типу Matrix: hue, saturation, brightness. Клас Image містить наступні методи для роботи з зображеннями: Image – конструктор класу, який зчитує зображення з файлової системи користувача та розкладає його на компоненти кольорової моделі; save – метод, який дозволяє конвертувати зображення у формат Bitmap та зберегти його у вигляді файлу; medianFilterImage – метод, який фільтрує зображення за допомогою класу MedianFilter; meanFilterImage – метод, який фільтрує зображення за допомогою класу MeanFilter.

4) MedianFilter – клас, основною задачею якого є виконання медіанної над типом даних Matrix. Фільтрація відбувається за допомогою метода process.
5) MeanFilter – клас, основною задачею якого є виконання середньоарифметичної над типом даних Matrix. Фільтрація відбувається за допомогою метода process.
6) MedianMatrixEncryptor – клас, який займається шифруванням та дешифруванням матриць типу Matrix для виконання подальшої віддаленої медіанної фільтрації, даної матриці за допомогою двох методів: encryptMatrix, decryptMatrix. Метод encryptMatrix, окрім зашифрованих значень матриці, генерує таблицю відповідності зашифрованих значень інтенсивності пікселів оригінальним, яка повинна бути передана в якості параметра в метод decryptMatrix, для подальшого дешифрування.
7) MedianImageEncryptor – клас, який дозволяє шифрувати та дешифрувати зображення типу Image за допомогою методів encryptImage та decryptImage. Метод encryptImage, окрім зашифрованого зображення, генерує три таблиці відповідності зашифрованих значень інтенсивності пікселів оригінальним, які

повинні бути передані в якості параметра в метод decryptImage, для подальшого дешифрування зображення.

8) MeanGroupMatrixEncryproг – клас, який займається шифруванням та дешифруванням групи матриць типу Matrix для виконання подальшої віддаленої середньоарифметичної фільтрації, даної групи за допомогою двох методів: encryptMatrices, decryptMatrices.

   Також, під час ініціалізації класу, відбувається процес генерації випадкових допоміжних матриць, та їх фільтрація за допомогою класу MeanFilter, для подальшого використання в шифруванні та дешифруванні. Метод encryptMatrices, окрім зашифрованих значень матриці, генерує ключ, який містить в собі інформацію про обрані випадкові послідовності, який повинен бути переданий в якості параметра в метод decryptMatrices, для подальшого дешифрування групи матриць.

MeanImageEncryptor – клас, який дозволяє шифрувати та дешифрувати зображення типу Image за допомогою методів encryptImage та decryptImage. Метод encryptImage, окрім зашифрованого зображення, генерує три ключі, який містять інформацію про обрані випадкові послідовності, які повинні бути передані в якості параметра в метод decryptImage, для подальшого дешифрування відфільтрованого зображення. при комбінації відфільтрованих компонентів зображення.

3.3. Інструкція користувачу

Для використання розробленої системи віддаленої медіанної та середньоарифметичної фільтрації користувачеві необхідно запустити дві версії програми: клієнтську та серверну.

Клієнтська версія програми займається ініціалізацію необхідних допоміжних, генерацією ключів шифрування, зчитуванням та представленням зображень у форматі HSB, шифруванням та дешифруванням

зображень. Для реалізації програмного комплексу, користувач має скомпілювати розроблену програмну систему з власним програмним кодом обміну інформації с серверною частиною системи.

Серверна версія програми займається обробкою зашифрованих зображень у форматі HSB за допомогою медіанного та середньоарифметичного фільтрів. Для реалізації програмного комплексу, користувач має скомпілювати розроблену програмну систему з власним сервером обробки клієнтських запитів.

Для обробки зображень з емуляцією процесу обміну зображеннями між клієнтом та сервером, необхідно запустити виконуваний файл програми з вказанням бажаного методу фільтрації в параметрі method. При запуску програми в режимі емуляції файли з директорії photo, яка знаходиться в корні програми будуть відфільтровані за допомогою обраного методу та збережені у директорію mean-filtered-photo та median-filtered-photo, які також знаходяться в корні програми.

Для запуску емуляції віддаленої групової середньоарифметичної фільтрації, користувачеві необхідно запустити виконуваний файл за допомогою наступної команди: “./emulator –method mean”. При успішному виконанні, користувач має побачити наступну інформацію, щодо виконання роботи програми.

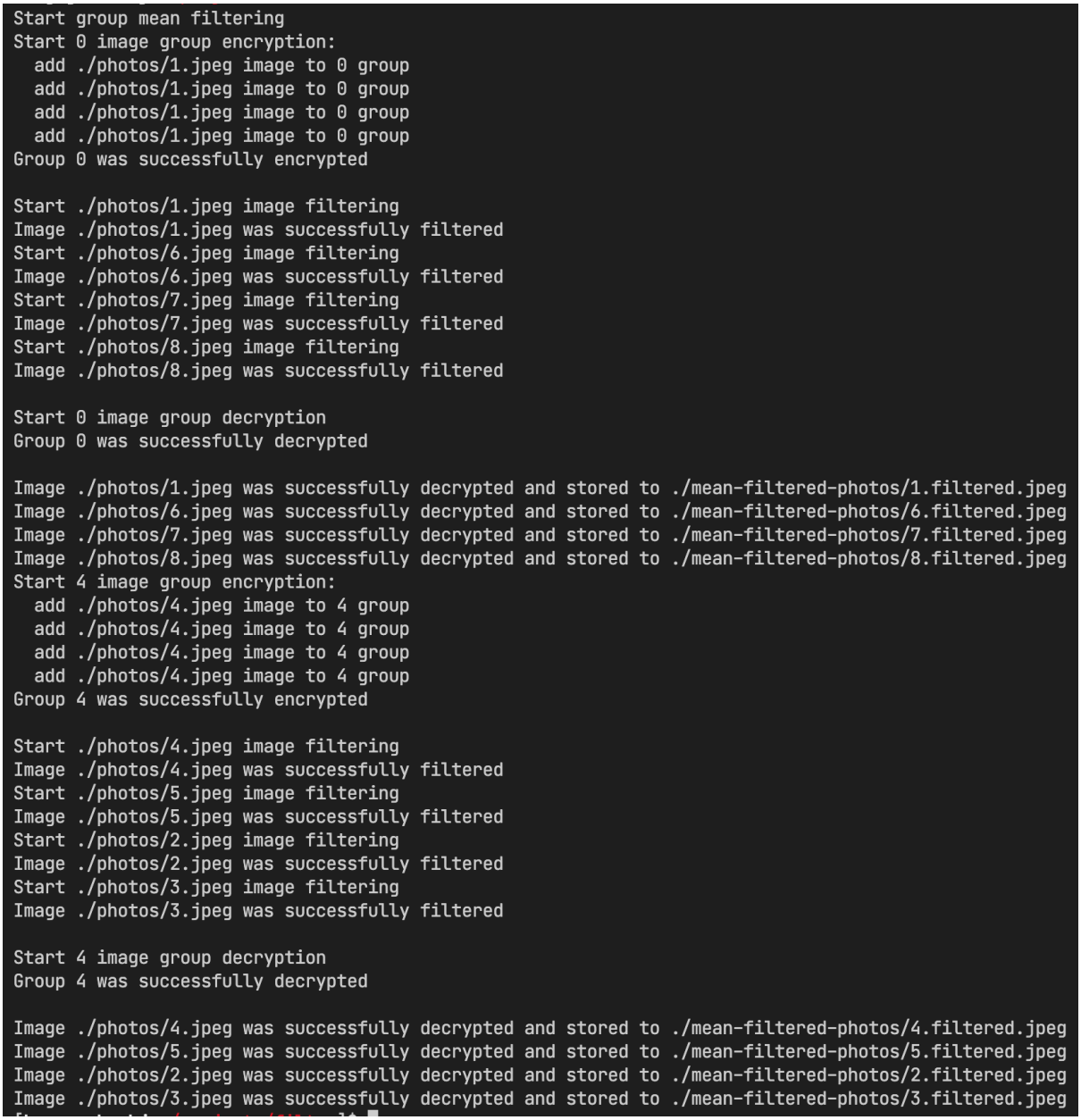


Рис.3.4. Успішний сценарій виконання емуляції віддаленої середньоарифметичної фільтрації двох груп зображень

Для запуску емуляції віддаленої медіанної фільтрації, користувачеві необхідно запустити виконуваний файл за допомогою наступної команди: “./emulator –method median”. При успішному виконанні, користувач має побачити наступну інформацію, щодо виконання роботи програми.

```
Start median filtering
Start ./photos/1.jpeg image encryption
Image ./photos/1.jpeg was successfully encrypted
Start ./photos/1.jpeg image filtering
Image ./photos/1.jpeg was successfully filtered
Start ./photos/1.jpeg image decryption
Image ./photos/1.jpeg was successfully decrypted and stored to ./median-filtered-photos/1.filtered.jpeg

Start ./photos/4.jpeg image encryption
Image ./photos/4.jpeg was successfully encrypted
Start ./photos/4.jpeg image filtering
Image ./photos/4.jpeg was successfully filtered
Start ./photos/4.jpeg image decryption
Image ./photos/4.jpeg was successfully decrypted and stored to ./median-filtered-photos/4.filtered.jpeg

Start ./photos/2.jpeg image encryption
Image ./photos/2.jpeg was successfully encrypted
Start ./photos/2.jpeg image filtering
Image ./photos/2.jpeg was successfully filtered
Start ./photos/2.jpeg image decryption
Image ./photos/2.jpeg was successfully decrypted and stored to ./median-filtered-photos/2.filtered.jpeg

Start ./photos/3.jpeg image encryption
Image ./photos/3.jpeg was successfully encrypted
Start ./photos/3.jpeg image filtering
Image ./photos/3.jpeg was successfully filtered
Start ./photos/3.jpeg image decryption
Image ./photos/3.jpeg was successfully decrypted and stored to ./median-filtered-photos/3.filtered.jpeg
```

Рис.3.5. Успішний сценарій виконання емуляції віддаленої медіанної фільтрації чотирьох зображень

Висновки до розділу 3

В результаті виконання проведеного аналізу та розробок програмно забезпечення у ході написання третього розділу даної дипломної роботи можна зробити наступні висновки:

1. Аналіз існуючих кольорових моделей показав, що найкращім вибором для представлення зображення у пам’яті, з метою виконання його подальшої медіанної та середньоарифметичної фільтрації, є модель представлення HVB. Дане представлення дозволяє розкласти зображення на основні складові, відфільтрувати кожну компоненту окремо та відновити відфільтроване зображення без появи аномальних відтінків кольорів.

2. Виконана розробка програмної системи для виконання медіанної та середньоарифметичної фільтрації за допомогою віддалених обчислювальних ресурсів. Система включає в себе функціональні компоненти для виконання зчитування, шифрування, фільтрації, дешифрування та збереження зображень, а також для виконання усіх підготовчих обрахувань. Розроблене програмне рішення представляє собою точну імплементацію розроблених методів віддаленої групової середньоарифметичної та медіанної фільтрації зображень.

3. Розроблений програмний засіб для виконання емуляції виконання медіанної та середньоарифметичної фільтрації. Для керування даним програмним рішенням був розроблений набір команд, за допомогою яких користувач може обирати метод та набір необхідних параметрів фільтрації.

ВИСНОВКИ

В результаті виконання дипломною проєкту по розробці методів захищеної медіанної та середньоарифметичної фільтрації зображень з використанням віддалених обчислювальних ресурсів, можна зробити наступні висновки:

1. Операції обробки медіа-контенту є однією з найпопулярніших задач в області комп'ютерних обчислень. З розвитком інформаційних технологій, збільшуються потреби в ресурсах, необхідних для виконання даних операцій. На сьогоднішній день, для ефективної обробки зображень чи відео-матеріалів, необхідно мати потужні обчислювальні пристрої, оскільки процес видозмінення зображення чи відео потребує обробки кожного фрагменту окремо.

2. Проведений аналіз існуючих технологій віддалених обчислювальних систем показав існуючий потенціал використання хмарних обчислень в операціях обробки зображень з метою покращення їх якості. Використання даних технологій дозволить суттєво підвищити швидкість фільтрації зображень. Однак, оскільки, в більшій мірі, зображення є конфіденційною інформацією, передача їх в незахищеному вигляді є неможливою. Виходячи с цього, існує доцільність розробки методів обробки даних в зашифрованому вигляді.

3. Розроблений метод медіанної фільтрації зображень на віддалених обчислювальних ресурсах. Метод відповідає переліченим необхідним критеріям ефективності, адже має достатню надійність, яка забезпечуються складністю перебору можливих значень ключів шифрування для відтворення оригінального зображення, та, оскільки, необхідна сумарна кількість обчислювальних ресурсів для шифрування та дешифрування зображення не перевищує кількість ресурсів необхідну для виконання оригінального алгоритму медіанної фільтрації.

4. Розроблений метод середньоарифметичної фільтрації зображень за допомогою віддалених обчислювальних ресурсів. Прискорення алгоритму досягається за рахунок перенесення частини підготовчих обчислень на віддалені ресурси. Метод відповідає переліченим необхідним критеріям ефективності, адже має достатню надійність, яка забезпечуються складністю перебору можливих значень ключів шифрування для відтворення оригінального зображення, та, оскільки, необхідна сумарна кількість обчислювальних ресурсів для шифрування та дешифрування зображення не перевищує кількість ресурсів необхідну для виконання оригінального алгоритму фільтрації.

5. Розроблений метод одночасної середньоарифметичної фільтрації декількох зображень за допомогою віддалених обчислювальних ресурсів. Метод відповідає переліченим необхідним критеріям ефективності, адже має достатню надійність, яка забезпечуються складністю перебору можливих значень ключів шифрування для відтворення оригінального зображення, та, оскільки, кількість обчислювальних ресурсів необхідних для виконання групової фільтрації менша за сумарну кількість ресурсів необхідну для виконання віддаленої обробки кожного зображення окремо.

6. Розроблена програмна система для виконання медіанної та середньоарифметичної фільтрації за допомогою віддалених обчислювальних ресурсів. Система включає в себе функціональні компоненти для виконання зчитування, шифрування, фільтрації, дешифрування та збереження зображень, а також для виконання усіх підготовчих обрахувань. Розроблене програмне рішення представляє собою точну імплементацію розроблених методів віддаленої групової середньоарифметичної та медіанної фільтрації зображень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sathish V. Cloud-based Image Processing With Data Priority Distribution Mechanism./ Sathish V., Sangeetha T.A. // Journal of Computer Applications.- Vol.6, №1.- 2013.- P. 6-8.
2. Monjur Ahmed. Cloud Computing and Security Issues in the Cloud / Monjur Ahmed, Mohammad Ashraf Hossain // International Journal of Network Security & Its Applications (IJNSA).- Vol.6, №1.- 2014.- pp.25-36.
3. Malgouyres F. A noise selection approach of image restoration, Applications in signal and image processing IX / F. Malgouyres // Vol 4478. – 2001. – P. 34-41.
4. Форсайт Д. Компьютерное зрение. Современный подход / Д. Форсайт, Ж. Понс. – М. : Вильямс, 2004. – 928 с. Лисицин В. З. Практикум по фотограмметрии и дистанционному зондированию / В. З. Лисицин. – Харьков : ХНАГХ, 2006. – 200 с.
5. Гуменюк І.О. Метод віддаленої середньоарифметичної фільтрації зображень / І.О. Гуменюк, О.Є. Слюсаренко // Альманах науки.- 2019.- № 11 (32).- С.40-43.
6. Сэвидж Д.Э. Сложность вычислений / Д.Э. Сэвидж. – М.: Факториал. – 1998. – 368 с.
7. Буйбарова М.Ф. Метод захищеної реалізації перетворень Фур'є на віддалених розподілених комп'ютерних системах / М.Ф. Буйбарова, Ю.М. Виноградов, В.Ю. Приймак // Вісник Національного технічного університету України "КПІ" Інформатика, управління та обчислювальна техніка. – К.: ТОО „ВЕК+". – № 64. – 2016. – С. 64-71.
8. Mirataei Alireza, Khalil Hana, Olexander Markovskyi Protected discrete Fourier transform implementation on remote computer systems // Information, Computing and Intelligent systems.-2020.- No 1.- P.26-33.
9. Задірака В.К. Комп'ютерна криптологія: Підручник / В.К. Задірака, О.С. Олексюк. – К.: Вища школа. – 2002. – 504с.

10. Ростовцев А.Г. Алгебраические основы криптографии / А.Г. Ростовцев. – СПб.: Мир и семья. – 2000. – 353 с.
11. Can Xiang Verifiable and Secure Outsourcing Schemes of Modular Exponentiations Using One Untrusted Cloud Server and Their Application / Can Xiang // IACR Cryptology ePrint Archive 2014: P.500. –510.
12. Грузман И.С. Цифровая обработка изображений в информационных системах / И. С. Грузман, В. С. Киричук – Новосибирск : НГТУ, 2002. − 352 с.
13. Марковський О.П. Захищена реалізація фільтрації зображень в GRID-системах / О.П. Марковський, М.В. Невдащенко, А.М. Білашевська // Вісник Національного технічного університету України ”КПІ” Інформатика, управління та обчислювальна техніка. – Київ: ВЕК+. – 2014. – № 61. – С.105-109.
14. PengyaoWang. Rapid processing of remote sensing images based on cloud computing /PengyaoWang,JianqinWang, YingChen, Guangyuan Ni // Future Generation Computer Systems. – Vol.29.- № 8.-2013.- pp.1963-1968.
15. Markovskyi O.P. Secure Modular Exponentiation in Cloud Systems/ O.P. Markovskyi, N. Bardis, S.J. Kirilenko // Proceeding of the Congress on Information Technology. Computationnal and Experimental Physics (CITCEP 2015), 18-20 December 2015, Krakow. Poland. – PP.266-269.
16. Гамаюн В.П. Квазиграфический метод вычисления остатка по модулю / В.П. Гамаюн //Проблеми інформатизації та управління. Зб.наукових праць. – К.: НАУ. – 2005. – Вип.3(14). – С.43-48.
17. Марковський О.П., Гуменюк І.О., Міратаї Аліреза, Торошанко Я.І., Волощук М.О. Метод прискореної захищеної фільтрації зображень на віддалених комп’ютерних системах / О.П. Марковський, // Телекомунікаційні та інформаційні технології**. -** № 4 (65).- 2019.- С.99-110.

18. Харин Ю.С. Математические и компьютерные основы криптологии / Ю.С. Харин, В.И. Берник, Г.В. Матвеев, С.В. Агиевич. – Мн.: Новое знание. – 2003. – 382 с.
19. Ростовцев А.Г. Алгебраические основы криптографии / А.Г. Ростовцев. – СПб.: Мир и семья. – 2000. – 353 с.
20. Сэвидж Д.Э. Сложность вычислений / Д.Э. Сэвидж. – М.: Факториал. – 1998. – 368 с.
21. Загальні положення щодо захисту інформації в комп'ютерних системах від несанкціонованого доступу. НД ТЗІ 2.5-004-99.
22. Задірака В.К. Комп'ютерна криптологія: Підручник / В.К. Задірака, О.С. Олексюк. – К.: Вища школа. – 2002. – 504с.
23. Sutton M. A. Image Correlation for Shape, Motion and Deformation Measurements (Basic Concepts, Theory and Applications)./ Sutton M. A., Orteu J. J., Schreier H. // – New York: Springer, 2009. – 364 p.
24. Malgouyres F. A noise selection approach of image restoration, Applications in signal and image processing IX / F. Malgouyres // Vol 4478. – 2001. – P. 34-41

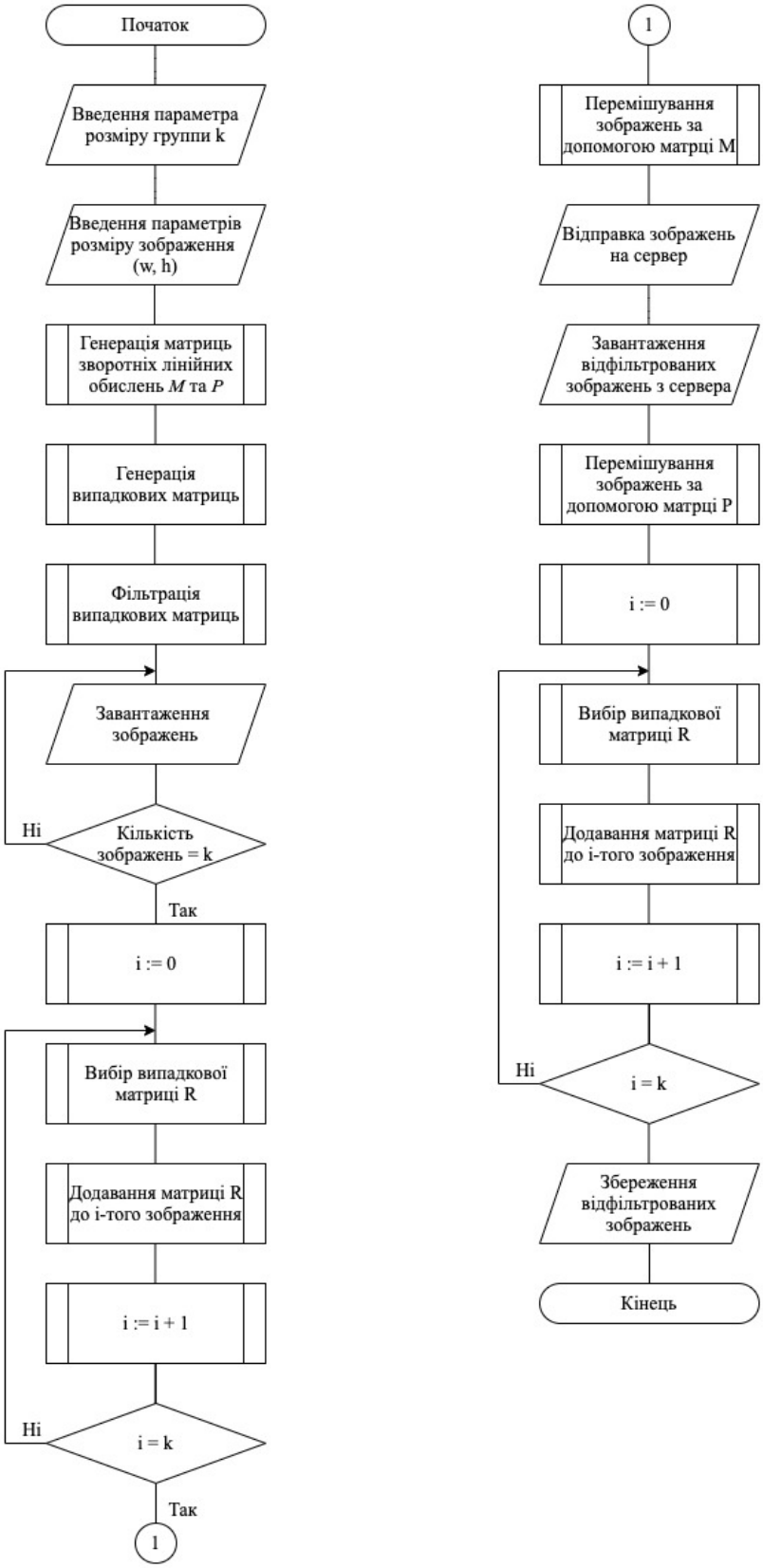
ДОДАТОК А

Метод гомоморфного шифрування для захищеної обробки даних в хмарах

Блок-схема алгоритму

Аркушів 1

Київ – 2021 р.



					ІАЛЦ 467400.004 Д1	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розробив		Тимошенко І.А.			*Метод гомоморфного шифрування для захищеної обробки даних в хмарах* Діаграма класів	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Марковський О.П.				
Н. Контр.		Сімоненко В.П.				«КПІ ім. Ігоря Сікорського», ФІОТ, ІП-73
Затвердив						

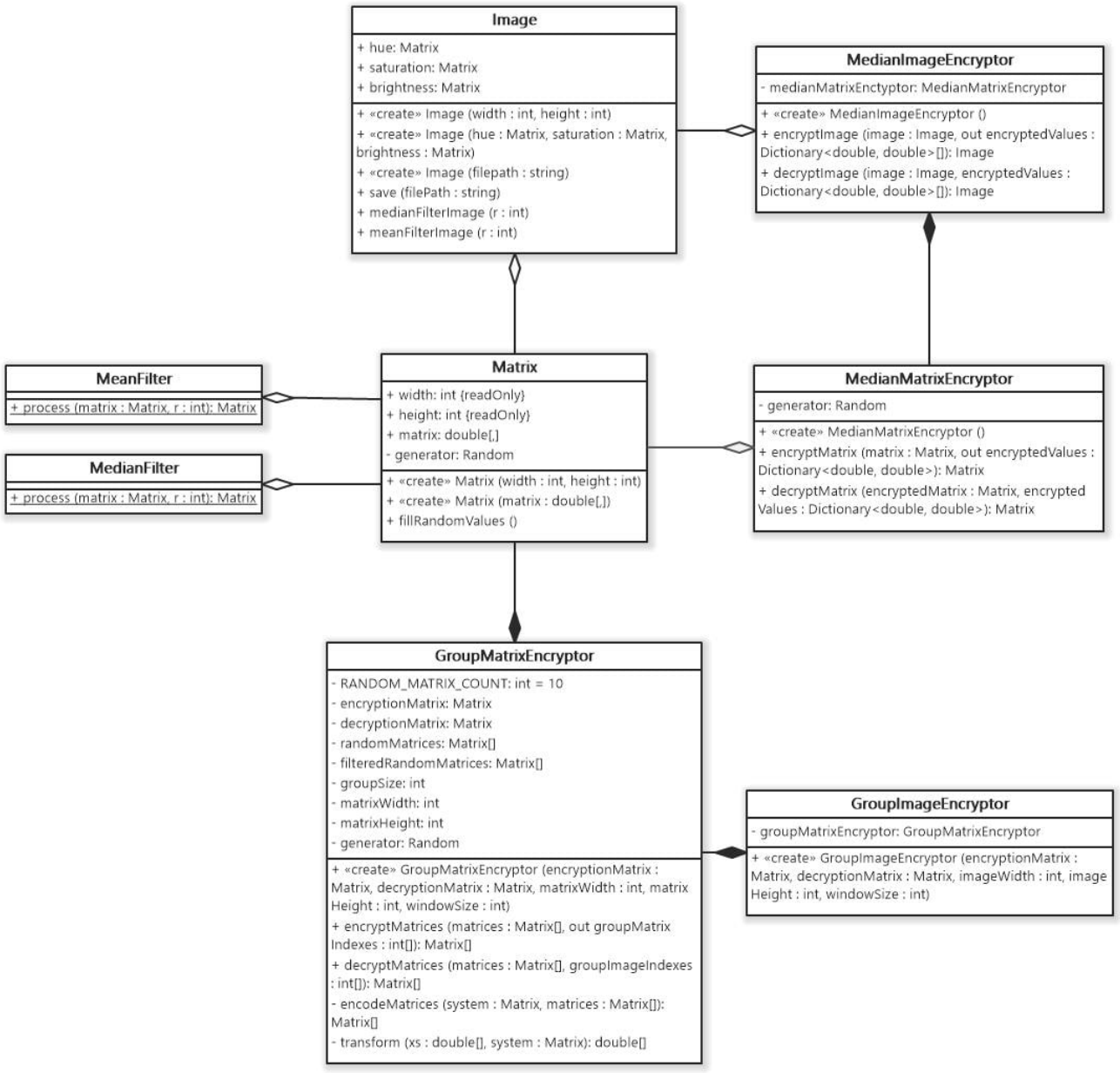
ДОДАТОК Б

Метод гомоморфного шифрування для захищеної обробки даних в хмарах

Діаграма класів

Аркушів 1

Київ – 2021 р.



					ІАЛЦ 467400.005 Д2	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розробив		Тимошенко І.А.			*Метод гомоморфного шифрування для захищеної обробки даних в хмарах* Діаграма класів	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Марковський О.П.				
Н. Контр.		Сімоненко В.П.				«КПІ ім. Ігоря Сікорського», ФІОТ, ІП-73
Затвердив						

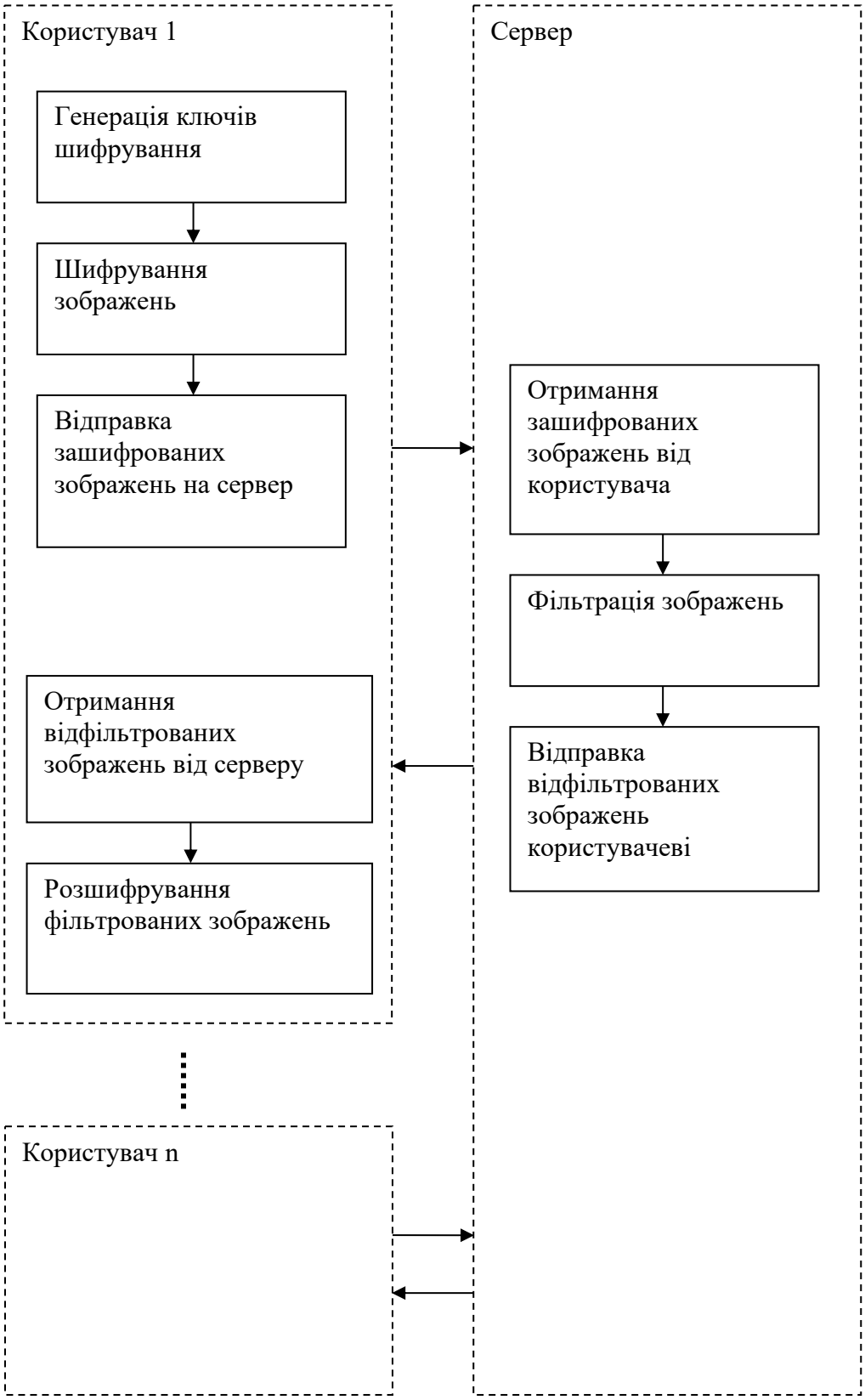
ДОДАТОК В

Метод гомоморфного шифрування для захищеної обробки даних в хмарах

Структурна схема

Аркушів 1

Київ – 2021 р.



					ІАЛЦ 467400.006 Д2	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розробив		Тимошенко І.А.			*Метод гомоморфного шифрування для захищеної обробки даних в хмарах* Діаграма класів	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Марковський О.П.				
Н. Контр.		Сімоненко В.П.				«КПІ ім. Ігоря Сікорського», ФІОТ, ІП-73
Затвердив						

ДОДАТОК Г

Метод гомоморфного шифрування для захищеної обробки даних в хмарах

Текст програми

Аркушів 15

Київ – 2021 р.

Matrix.cs

```
using System;

namespace filter
{
  public class Matrix
  {
    public readonly int width;
    public readonly int height;
    public double[,] matrix;

    private Random generator;

    public Matrix(int width, int height) {
      this.width = width;
      this.height = height;
      this.matrix = new double[width, height];

      this.generator = new Random();
    }

    public Matrix(double[,] matrix) {
      this.width = matrix.GetLength(1);
      this.height = matrix.GetLength(0);
      this.matrix = matrix;

      this.generator = new Random();
    }


    public void fillRandomValues() {
      for (int i = 0; i < this.width; i++)
      {
        for (int j = 0; j < this.height; j++)
        {
          this.matrix[i, j] = generator.Next(20);
        }
      }
    }

    public static Matrix operator +(Matrix matrix1, Matrix
matrix2) {
      if (matrix1.width != matrix2.width || matrix1.height !=
matrix2.height) {
        throw new Exception("Matrices have different size");
      }
```

```
      Matrix resultMatrix = new Matrix(matrix1.width,
matrix1.height);
      for (int i = 0; i < matrix1.width; i++)
      {
        for (int j = 0; j < matrix1.height; j++)
        {
          resultMatrix.matrix[i, j] = matrix1.matrix[i, j] +
matrix2.matrix[i, j];
        }
      }

      return resultMatrix;
    }

    public static Matrix operator -(Matrix matrix1, Matrix
matrix2) {
      if (matrix1.width != matrix2.width || matrix1.height !=
matrix2.height) {
        throw new Exception("Matrices have different size");
      }

      Matrix resultMatrix = new Matrix(matrix1.width,
matrix1.height);
      for (int i = 0; i < matrix1.width; i++)
      {
        for (int j = 0; j < matrix1.height; j++)
        {
          resultMatrix.matrix[i, j] = matrix1.matrix[i, j] -
matrix2.matrix[i, j];
        }
      }

      return resultMatrix;
    }

    public static Matrix operator *(Matrix matrix, float number)
{
      Matrix resultMatrix = new Matrix(matrix.width,
matrix.height);
      for (int i = 0; i < matrix.width; i++)
      {
        for (int j = 0; j < matrix.height; j++)
        {
          resultMatrix.matrix[i, j] = matrix.matrix[i, j] *
number;
        }
      }
```

```
      return resultMatrix;
    }

    public Matrix clone() {
      Matrix resultMatrix = new Matrix(this.width, this.height);
      for (int i = 0; i < this.width; i++)
      {
        for (int j = 0; j < this.height; j++)
        {
          resultMatrix.matrix[i, j] = this.matrix[i, j];
        }
      }
      return resultMatrix;
    }

    public string toString() {
      string result = "";
      for (int i = 0; i < this.width; i++)
      {
        for (int j = 0; j < this.height; j++) {
          result += Math.Round(this.matrix[i, j]) + " ";
        }
        result += "\n";
      }
      return result;
    }
  }
}
```

Image.cs

```
using System.Drawing;

namespace filter
{
  public class Image
  {
    public Matrix hue;
    public Matrix saturation;
    public Matrix brightness;

    public Image(int width, int height)
    {
      this.hue = new Matrix(width, height);
      this.saturation = new Matrix(width, height);
      this.brightness = new Matrix(width, height);
    }
```

```
    public   Image(Matrix   hue,   Matrix   saturation,   Matrix
brightness)
    {
      this.hue = hue;
      this.saturation = saturation;
      this.brightness = brightness;
    }

    public Image(string filepath)
    {
      var image = new Bitmap(filepath);

      this.hue = new Matrix(image.Width, image.Height);
      this.saturation = new Matrix(image.Width, image.Height);
      this.brightness = new Matrix(image.Width, image.Height);

      for (int i = 0; i < image.Width; i++)
      {
        for (int j = 0; j < image.Height; j++)
        {
          var pixelColor = image.GetPixel(i, j);
          HSV.ColorToHSV(
            pixelColor,
            out this.hue.matrix[i, j],
            out this.saturation.matrix[i, j],
            out this.brightness.matrix[i, j]
          );
        }
      }
    }

    public void save(string filePath)
    {
      var image = new Bitmap(this.hue.width, this.hue.height);

      for (int i = 0; i < image.Width; i++)
      {
        for (int j = 0; j < image.Height; j++)
        {
          Color pixelColor = HSV.ColorFromHSV(
            this.hue.matrix[i, j],
            this.saturation.matrix[i, j],
            this.brightness.matrix[i, j]
          );

          image.SetPixel(i, j, pixelColor);
        }
      }
```

```
      image.Save(filePath);
    }

    public void medianFilterImage(int r)
    {
      this.hue = MedianFilter.process(this.hue, r);
      this.saturation   =   MedianFilter.process(this.saturation,
r);
      this.brightness   =   MedianFilter.process(this.brightness,
r);
    }

    public void meanFilterImage(int r)
    {
      this.hue = MeanFilter.process(this.hue, r);
      this.saturation = MeanFilter.process(this.saturation, r);
      this.brightness = MeanFilter.process(this.brightness, r);
    }
  }
}
```

HSV.ja

```
using System;
using System.Drawing;

public class HSV
{
  public static void ColorToHSV(Color color, out double hue, out
double saturation, out double value)
  {
    int max = Math.Max(color.R, Math.Max(color.G, color.B));
    int min = Math.Min(color.R, Math.Min(color.G, color.B));

    hue = color.GetHue();
    saturation = (max == 0) ? 0 : 1d - (1d * min / max);
    value = max / 255d;
  }

  public static Color ColorFromHSV(double hue, double saturation,
double value)
  {
    int hi = Convert.ToInt32(Math.Floor(hue / 60)) % 6;
    double f = hue / 60 - Math.Floor(hue / 60);

    value = value * 255;
    int v = Convert.ToInt32(value);
    int p = Convert.ToInt32(value * (1 - saturation));
```

```
    int q = Convert.ToInt32(value * (1 - f * saturation));
    int t = Convert.ToInt32(value * (1 - (1 - f) * saturation));

    if (hi == 0)
      return Color.FromArgb(255, v, t, p);
    else if (hi == 1)
      return Color.FromArgb(255, q, v, p);
    else if (hi == 2)
      return Color.FromArgb(255, p, v, t);
    else if (hi == 3)
      return Color.FromArgb(255, p, q, v);
    else if (hi == 4)
      return Color.FromArgb(255, t, p, v);
    else
      return Color.FromArgb(255, v, p, q);
  }
}
```

MeanFilter.cs

```
namespace filter
{
  public class MeanFilter
  {
    public static Matrix process(Matrix matrix, int r)
    {
      Matrix resultMatrix = matrix.clone();

      int offset = (r - 1) / 2;
      for (int i = offset; i < matrix.width - offset; i++)
      {
        for (int j = offset; j < matrix.height - offset; j++)
        {
          resultMatrix.matrix[i, j] = 0;
          for (int k = i - offset; k <= i + offset; k++)
          {
            for (int g = j - offset; g <= j + offset; g++)
            {
              resultMatrix.matrix[i, j] += matrix.matrix[k, g];
            }
          }
          resultMatrix.matrix[i, j] /= r * r;
        }
      }
      return resultMatrix;
    }
  }
}
```

MedianFilter.cs

```
using System;

namespace filter
{
  public class MedianFilter
  {
    public static Matrix process(Matrix matrix, int r)
    {
      Matrix resultMatrix = matrix.clone();

      int offset = (r - 1) / 2;
      int windowOffset = (r * r - 1) / 2;
      for (int i = offset; i < matrix.width - offset; i++)
      {
        for (int j = offset; j < matrix.height - offset; j++)
        {
          int windowIndex = 0;
          var window = new double[r * r];
          for (int k = i - offset; k <= i + offset; k++)
          {
            for (int g = j - offset; g <= j + offset; g++)
            {
              window[windowIndex++] = matrix.matrix[k, g];
            }
          }
          Array.Sort(window);
          resultMatrix.matrix[i, j] = window[windowOffset];
        }
      }
      return resultMatrix;
    }
  }
}
```

MedianMatrixEncryptor.cs

```
using System;
using System.Collections.Generic;

namespace filter
{
  public class MedianMatrixEncryptor
  {
    private Random generator;

    public MedianMatrixEncryptor()
```

```
    {
      this.generator = new Random();
    }

    public Matrix encryptMatrix(Matrix matrix, out
Dictionary<double, double> encryptedValues) {
      int randomKey = this.generator.Next();
      int globalRandomShift = this.generator.Next();

      encryptedValues = new Dictionary<double, double>();

      Matrix resultMatrix = new Matrix(matrix.width,
matrix.height);
      for (int i = 0; i < matrix.width; i++)
      {
        for (int j = 0; j < matrix.height; j++)
        {
          var randomShift = this.generator.Next(randomKey);
          resultMatrix.matrix[i, j] = matrix.matrix[i, j] *
randomKey + randomShift + globalRandomShift;
          encryptedValues.TryAdd(resultMatrix.matrix[i, j],
matrix.matrix[i, j]);
        }
      }

      return resultMatrix;
    }

    public Matrix decryptMatrix(Matrix encryptedMatrix,
Dictionary<double, double> encryptedValues) {
      Matrix resultMatrix = new Matrix(encryptedMatrix.width,
encryptedMatrix.height);
      for (int i = 0; i < encryptedMatrix.width; i++)
      {
        for (int j = 0; j < encryptedMatrix.height; j++)
        {
          encryptedValues.TryGetValue(encryptedMatrix.matrix[i,
j], out resultMatrix.matrix[i, j]);
        }
      }

      return resultMatrix;
    }
  }
}
```

MedianImageEncryptor.cs

```
using System.Collections.Generic;

namespace filter
{
  public class MedianImageEncryptor
  {
    private MedianMatrixEncryptor medianMatrixEnctyptor;

    public MedianImageEncryptor(){
      this.medianMatrixEnctyptor = new MedianMatrixEncryptor();
    }

    public Image encryptImage(Image image, out
Dictionary<double, double>[] encryptedValues) {
      encryptedValues = new Dictionary<double, double>[3];

      var encryptedHue =
medianMatrixEnctyptor.encryptMatrix(image.hue, out
encryptedValues[0]);
      var encryptedSaturation =
medianMatrixEnctyptor.encryptMatrix(image.saturation, out
encryptedValues[1]);
      var encryptedBrightness =
medianMatrixEnctyptor.encryptMatrix(image.brightness, out
encryptedValues[2]);

      return new Image(
        encryptedHue,
        encryptedSaturation,
        encryptedBrightness
      );
    }

    public Image decryptImage(Image image, Dictionary<double,
double>[] encryptedValues) {
      var decryptedHue =
medianMatrixEnctyptor.decryptMatrix(image.hue,
encryptedValues[0]);
      var decryptedSaturation =
medianMatrixEnctyptor.decryptMatrix(image.saturation,
encryptedValues[1]);
      var decryptedBrightness =
medianMatrixEnctyptor.decryptMatrix(image.brightness,
encryptedValues[2]);

      return new Image(
```

```
        decryptedHue,
        decryptedSaturation,
        decryptedBrightness
      );
    }
  }
}
```

GroupMatrixEncryptor.cs

```
using System;

namespace filter
{
  public class GroupMatrixEncryptor
  {
    private const int RANDOM_MATRIX_COUNT = 10;

    private Matrix encryptionMatrix;
    private Matrix decryptionMatrix;

    private Matrix[] randomMatrices;
    private Matrix[] filteredRandomMatrices;

    private int groupSize;
    private int matrixWidth;
    private int matrixHeight;

    private Random generator;

    public GroupMatrixEncryptor(
      Matrix encryptionMatrix,
      Matrix decryptionMatrix,
      int matrixWidth,
      int matrixHeight,
      int windowSize
    )
    {
      if (
        encryptionMatrix.width != decryptionMatrix.width ||
        encryptionMatrix.height != decryptionMatrix.height
      ) {
        throw new Exception("Matrices have different size");
      }

      this.encryptionMatrix = encryptionMatrix;
      this.decryptionMatrix = decryptionMatrix;
```

```
      this.matrixWidth = matrixWidth;
      this.matrixHeight = matrixHeight;
      this.groupSize = encryptionMatrix.height;

      this.randomMatrices = new Matrix[RANDOM_MATRIX_COUNT];
      this.filteredRandomMatrices = new
Matrix[RANDOM_MATRIX_COUNT];

      for (int i = 0; i < RANDOM_MATRIX_COUNT; i++) {
        var randomMatrix = new Matrix(matrixWidth,
matrixHeight);
        randomMatrix.fillRandomValues();

        var filteredRandomMatrix =
MeanFilter.process(randomMatrix, windowSize);

        this.randomMatrices[i] = randomMatrix;
        this.filteredRandomMatrices[i] = filteredRandomMatrix;
      }

      this.generator = new Random();
    }

    public Matrix[] encryptMatrices(Matrix[] matrices, out int[]
groupMatrixIndexes) {
      groupMatrixIndexes = new int[this.groupSize];

      for (int k = 0; k < this.groupSize; k++)
      {
        int randomIndex =
generator.Next(this.randomMatrices.Length);
        var randomMatrix = this.randomMatrices[randomIndex];

        matrices[k] = matrices[k] + randomMatrix;
        groupMatrixIndexes[k] = randomIndex;
      }
      return this.encodeMatrices(this.encryptionMatrix,
matrices);
    }

    public Matrix[] decryptMatrices(Matrix[] matrices, int[]
groupImageIndexes) {
      var decryptedMarteces =
this.encodeMatrices(this.decryptionMatrix, matrices);

      for (int k = 0; k < this.groupSize; k++)
      {
```

```
        int randomIndex = groupImageIndexes[k];
        var filteredRandomMatrix =
this.filteredRandomMatrices[randomIndex];

        decryptedMarteces[k] = decryptedMarteces[k] -
filteredRandomMatrix;
      }

      return decryptedMarteces;
    }

    private Matrix[] encodeMatrices(Matrix system, Matrix[]
matrices)
    {
      if (matrices.Length != this.groupSize) {
        throw new Exception("Unsupported group size");
      }

      for (int i = 0; i < this.groupSize; i++) {
        if (
          matrices[i].width != this.matrixWidth ||
          matrices[i].height != this.matrixHeight
        ) {
          throw new Exception("Unsupported matrix size");
        }
      }

      var encryptedMatrices = new Matrix[this.groupSize];
      for (int i = 0; i < this.groupSize; i++) {
        encryptedMatrices[i] = new Matrix(this.matrixWidth,
this.matrixHeight);
      }

      var xs = new double[this.groupSize];
      for (int i = 0; i < this.matrixWidth; i++)
      {
        for (int j = 0; j < this.matrixHeight; j++)
        {
          for (int k = 0; k < this.groupSize; k++)
          {
            xs[k] = matrices[k].matrix[i, j];
          }

          var ys = this.transform(xs, system);

          for (int k = 0; k < this.groupSize; k++)
          {
            encryptedMatrices[k].matrix[i, j] = ys[k];
```

```
          }
        }
      }
      return encryptedMatrices;
    }


    private double[] transform(double[] xs, Matrix system) {
      var ys = new double[this.groupSize];

      for (int i = 0; i < this.groupSize; i++)
      {
        ys[i] = 0;
        for (int j = 0; j < this.groupSize; j++)
        {
          ys[i] += system.matrix[i, j] * xs[j];
        }
      }
      return ys;
    }
  }
}
```

GroupImageEncryptor.cs

```
namespace filter
{
  public class GroupImageEncryptor
  {
    private GroupMatrixEncryptor groupMatrixEncryptor;

    public GroupImageEncryptor(
      Matrix encryptionMatrix,
      Matrix decryptionMatrix,
      int imageWidth,
      int imageHeight,
      int windowSize
    ){
      this.groupMatrixEncryptor = new GroupMatrixEncryptor(
        encryptionMatrix,
        decryptionMatrix,
        imageWidth,
        imageHeight,
        windowSize
      );
    }
```

```
    public Image[] encryptImages(Image[] images, out int[][]
groupImageIndexes) {
      var hueGroup = new Matrix[images.Length];
      var saturationGroup = new Matrix[images.Length];
      var brightnessGroup = new Matrix[images.Length];

      for (int i = 0; i < images.Length; i++)
      {
        hueGroup[i] = images[i].hue;
        saturationGroup[i] = images[i].saturation;
        brightnessGroup[i] = images[i].brightness;
      }

      groupImageIndexes = new int[3][];

      var encryptedHueGroup =
groupMatrixEncryptor.encryptMatrices(hueGroup, out
groupImageIndexes[0]);
      var encryptedSaturationGroup =
groupMatrixEncryptor.encryptMatrices(saturationGroup, out
groupImageIndexes[1]);
      var encryptedBrightnessGroup =
groupMatrixEncryptor.encryptMatrices(brightnessGroup, out
groupImageIndexes[2]);

      var encrypedImages = new Image[images.Length];
      for (int i = 0; i < images.Length; i++)
      {
        encrypedImages[i] = new Image(
          encryptedHueGroup[i],
          encryptedSaturationGroup[i],
          encryptedBrightnessGroup[i]
        );
      }
      return encrypedImages;
    }

    public Image[] decryptImages(Image[] encrypedImages, int[][]
groupImageIndexes) {
      var encrypedHueGroup = new Matrix[encrypedImages.Length];
      var encrypedSaturationGroup = new
Matrix[encrypedImages.Length];
      var encrypedBrightnessGroup = new
Matrix[encrypedImages.Length];

      for (int i = 0; i < encrypedImages.Length; i++)
      {
        encrypedHueGroup[i] = encrypedImages[i].hue;
```

```
        encrypedSaturationGroup[i] =
encrypedImages[i].saturation;
        encrypedBrightnessGroup[i] =
encrypedImages[i].brightness;
      }

      var decrypedHueGroup =
groupMatrixEncryptor.decryptMatrices(encrypedHueGroup,
groupImageIndexes[0]);
      var decrypedSaturationGroup =
groupMatrixEncryptor.decryptMatrices(encrypedSaturationGroup,
groupImageIndexes[1]);
      var decrypedBrightnessGroup =
groupMatrixEncryptor.decryptMatrices(encrypedBrightnessGroup,
groupImageIndexes[2]);

      var decrypedImages = new Image[encrypedImages.Length];
      for (int i = 0; i < encrypedImages.Length; i++)
      {
        decrypedImages[i] = new Image(
          decrypedHueGroup[i],
          decrypedSaturationGroup[i],
          decrypedBrightnessGroup[i]
        );
      }
      return decrypedImages;
    }
  }
}
```