

Харківський національний університет радіоелектроніки
(повне найменування вищого навчального закладу)

Факультет Комп'ютерної інженерії та управління
(повне найменування інституту, назва факультету (відділення))

Кафедра Безпеки інформаційних технологій
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до атестаційної роботи

бакалавра

(освітньо-кваліфікаційний рівень)

на тему «Розробка та дослідження алгоритмів приховування інформації у
кластери файлових систем»

Виконав: студент 4 курсу, групи БІКС-12-2
напряму підготовки (спеціальності)
6.170101 Безпека інформаційних та
комунікаційних систем
(шифр і назва напряму підготовки, спеціальності)

Шеханін К.Ю.

(прізвище та ініціали)

Керівник Кузнецов О.О.

(прізвище та ініціали)

Рецензент _____

(прізвище та ініціали)

Допускається до захисту
Зав. кафедри

(підпис)

Халімом Г.З.
(прізвище, ініціали)

Харків – 2016 року

Харківський національний університет радіоелектроніки

(повне найменування вищого навчального закладу)

Інститут, факультет, відділення Комп'ютерної інженерії та управління

Кафедра, циклова комісія Безпека інформаційних технологій

Освітньо-кваліфікаційний рівень Бакалавр

Напрямок підготовки 6.170101 Безпека інформаційних та комунікаційних систем

(шифр і назва)

ЗАТВЕРДЖУЮ

в. о. завідувача кафедри БІТ

_____ д.т.н проф. Халімов Г. З.

“ ____ ” _____ 2016 року

З А В Д А Н Н Я
НА АТЕСТАЦІЙНУ РОБОТУ СТУДЕНТУ

Шеханіну Кирилу Юрійовичу

(прізвище, ім'я, по батькові)

1. Тема роботи: Розробка та дослідження алгоритмів приховування інформації у кластери файлових систем

керівник проекту (роботи) Кузнецов Олександр Олександрович, проф., д.т.н.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від “16” травня 2016 року
№530 СТ

2. Строк подання студентом проекту (роботи) 08 червня 2016 року

3. Вихідні дані до проекту (роботи): специфікація файлової системи FAT32, інформаційне повідомлення у вигляді масиву стеганоблоків.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):

1) аналіз та дослідження властивостей файлових систем;

2) розробка та дослідження алгоритму приховування та вилучення інформації у кластери файлової системи FAT32;

3) розробка програмної реалізації алгоритму приховування інформації та експертне дослідження;

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): презентаційні матеріали у вигляді слайдів, блок-схеми алгоритму приховування та вилучення інформації з кластерів файлової системи, інтерфейс програмної реалізації розробленого алгоритму, результати досліджень алгоритму приховування та вилучення інформації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розробка дослідження алгоритму	Кузнецов О.О, проф., д.т.н.		
Розробка програмної реалізації	Олешко О.І. доц., к. т. н.		

7. Дата видачі завдання “16” квітня 2016 р.

Керівник роботи

(підпис)

проф., д.т.н. Кузнецов О.О.
(посада, прізвище, ім'я, по батькові)

Завдання прийняв до виконання

(підпис студента)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1.	Отримання завдання	22.05.2014	виконано
2.	Аналіз актуальності даної роботи	24.05.2014	виконано
3.	Опис та порівняльний аналіз основних файлових систем	26.05.2014	виконано
4.	Аналіз стеганографічних властивостей файлової системи FAT32	28.05.2014	виконано
5.	Розробка та оцінка методу приховування інформації шляхом перемішування кластерів	30.05.2014	виконано
6.	Розробка та оцінка програмної реалізації методу приховування інформації	02.06.2014	виконано
7.	Оформлення пояснювальної записки	10.06.2014	виконано

Студент

(підпис)

Шеханін К.Ю.
(прізвище та ініціали)

Керівник роботи

(підпис)

Кузнецов О.О.
(прізвище та ініціали)

РЕФЕРАТ

Звіт містить: 81 ст., 25 рис., 11 табл., 6 формул, 18 посилань, 2 додатка.

Об'єктом дослідження є метод приховування інформації в структурі файлової системи.

Предметом дослідження – алгоритм приховування та вилучення інформації із структури файлової системи FAT32, оцінка та аналіз ефективності методу приховування інформації, програмна реалізація методу приховування інформації шляхом перемішування кластерів файлової системи FAT32, аналіз ефективності програмної реалізації.

Метою даної роботи є розробка та аналіз програмної реалізації алгоритму приховування та вилучення інформації в структурі файлової системи FAT32 шляхом перемішування кластерів покриваючих файлів у певному порядку.

ФАЙЛОВА СИСТЕМА, ПОРІВНЯЛЬНИЙ АНАЛІЗ ФС, ТЕХНІЧНА СТЕГANOГРАФІЯ, ПРОПУСКНА ЗДАТНІСТЬ, ПЕРЕМІШУВАННЯ КЛАСТЕРІВ, АНАЛІЗ АЛГОРИТМУ ПРИХОВУВАННЯ ТА ВИЛУЧЕННЯ ІНФОРМАЦІЇ.

ABSTRACT

Explanatory note: 81 p., 25 figures, 11 tables, 6 formulas, 18 links, 2 applications.

Object of research is method for hiding information in the structure of the file system.

The subject of the study – hiding and extract algorithm information from the structure of the FAT32 file system, assessment and analysis of the effectiveness of the method of hiding information, software implementation method of hiding information by mixing cluster file system FAT32, analysis of the effectiveness of program implementation.

The purpose of this work is the development and analysis software implementation of the algorithm hiding and extracting information in the FAT32 file system structure by mixing covering clusters of files in a specific order.

FILE SYSTEM, FS COMPARATIVE ANALYSIS, TECHNICAL STEGANOGRAPHY, THROUGHPUT, MIXING CLUTERS, ANALYSIS ALGORITHM HIDING AND EXTRACTING INFORMATION.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ	8
ВСТУП.....	9
1 АНАЛІЗ ТА ДОСЛІДЖЕННЯ ВЛАСТИВОСТЕЙ ФАЙЛОВИХ СИСТЕМ...	11
1.1 Аналіз властивостей та параметрів файлової системи FAT32	12
1.2 Аналіз властивостей та параметрів файлової системи NTFS	16
1.3 Аналіз властивостей та параметрів файлової системи exFAT	17
1.4 Теоретична та практична оцінка файлових систем FAT32, NTFS, exFAT.....	18
2 РОЗРОБКА ТА ДОСЛІДЖЕННЯ ВЛАСТИВОСТЕЙ АЛГОРИТМУ ПРИХОВУВАННЯ ТА ВИЛУЧЕННЯ ІНФОРМАЦІЇ У КЛАСТЕРИ ФАЙлової СИСТЕМИ	22
2.1 Опис властивостей FAT32, що можуть сприяти приховуванню інформації	22
2.2 Опис методу приховування даних шляхом перемішування кластерів	26
2.3 Аналіз пропускної здатності методу приховування даних шляхом перемішування кластерів	30
3 РОЗРОБКА ПРОГРАМНОЇ РЕАЛІЗАЦІЇ АЛГОРИТМУ ПРИХОВУВАННЯ ІНФОРМАЦІЇ ТА ЕКСПЕРТНЕ ДОСЛІДЖЕННЯ	36
3.1 Опис програмної реалізації методу приховування та вилучення даних.....	36
3.2 Опис алгоритму роботи програми «SteganoFAT»	41
3.3 Інструкція користувача та рекомендації при роботі із програмою «SteganoFAT»	48
3.4 Аналіз часових параметрів та пропускної здатності програмної реалізації методу приховування та вилучення інформації у кластери файлової системи	53

ВИСНОВКИ.....	59
ПЕРЕЛІК ПОСИЛАНЬ	61
ДОДАТОК А ПРОГРАМНИЙ КОД «STEGANOFAT»	63
ДОДАТОК Б НАУКОВА ДІЯЛЬНІСТЬ	76

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ,
СКОРОЧЕНЬ І ТЕРМІНІВ

Гб – гігабайт (10^9 байт);

Гіб – гібібайт (2^{30} байт);

Еіб – ексбібайт (2^{60} байт);

Кб – кілобайт (10^3 байт);

Мб – мегабайт (10^6 байт);

Тіб – тебібайт (2^{40} байт);

ФС – файлова система;

API – application programming interface;

EOC – end of clusterchain;

exFAT – extended file allocation table;

FAT – file allocation table;

LFN – long file name;

MFT – master file table;

NTFS – new technology file system;

USB – universal serial bus;

USN – update sequence number.

ВСТУП

Сьогодні інформацію розглядають як один із основних ресурсів розвитку суспільства, а інформаційні системи і технології, як засоби підвищення продуктивності та ефективності роботи персоналу підприємств, організацій, установ.

Інформаційні технології визначають процеси передачі і розповсюдження, зберігання та обробки інформації, а також її використання у певних цілях. Інколи, факт виконання цих процесів повинен бути прихований від сторонніх осіб. Цим і займається галузь науки стеганографія.

Суспільству з давна відомо більшість стеганографічних методів заснованих на фізичних явищах природи, чи фізіологічних особливостей людського організму. Але технології не стоять на місці, із відкриттям нових засобів обробки та зберігання інформації з'являються нові методи приховування інформації, що засновані на технічних особливостях технологічних засобів і методів обробки інформації, дана галузь науки називається технічною стеганографією.

На даний час відомо декілька методів технічної стеганографії. Приховування інформації у модель під час 3D-друку, дана галузь приховування інформації має певні переваги та недоліки, а саме: відносно більшу коштовність при створенні прихованого повідомлення, та складності при зчитуванні інформації. Другий напрямок технічної стеганографії пов'язаний із мережевим трафіком. У даному методі інформація може приховуватись, наприклад, у поля заголовків протоколів, чи, наприклад, передача прихованого повідомлення шляхом посилення певної послідовності пакетів.

У даній роботі представлено метод технічної стеганографії, що базується на структурній особливості файлових систем у носіях інформації. А саме, приховування інформації у файловій системі FAT32 шляхом перемішування кластерів певних, ключових, файлів. Аналіз даного алгоритму є актуальним, бо

на даний час існує незначна кількість алгоритмів, що використовують цю технічну особливість.

Метою цієї роботи є розробка та аналіз алгоритму приховування інформації у кластери файлової системи, розробка програмної реалізації, та створення вихідних даних для подальшого аналізу стеганографічних методів, що базуються на даній технічній особливості.

Подальша галузь використання результатів даної роботи охоплює широкий спектр застосувань, а саме: для зберігання прихованої інформації на електронному носії, для прихованої передачі даних, для подальшого аналізу та розробки більш досконалих методів технічної стеганографії.

1 АНАЛІЗ ТА ДОСЛІДЖЕННЯ ВЛАСТИВОСТЕЙ ФАЙЛОВИХ СИСТЕМ

Файлова система – порядок, що визначає спосіб організації, збереження та іменування даних на носіях інформації. Файлова система визначає формат змісту та спосіб фізичного збереження інформації, яку прийнято групувати у вигляді файлів. Конкретна файлова система визначає розмір імен файлів та каталогів, максимальний можливий розмір файлу та розділу, набір атрибутів файлу. Деякі файлові системи надають сервісні можливості, наприклад розмежування доступу або шифрування файлів.

Файлова система зв'язує носія інформації з одного боку та API для доступу до файлів – з другого. Коли прикладна програма звертається до файлу, вона не має жодного уявлення про те, яким чином розташована інформація в певному файлі, так само як і на якому фізичному типі носія він збережений. Все, що знає програма, – це ім'я файлу, його розмір та атрибути. Ці данні вона отримує від драйвера файлової системи.

Файлова система не обов'язково зв'язана з фізичними носієм інформації. Існують віртуальні файлові системи, а також мережеві файлові системи, котрі є лише способом доступу до файлів, що знаходяться на віддаленому комп'ютері.

Загалом можна привести такі основні задачі файлової системи:

- а) іменування файлів;
- б) програмний інтерфейс роботи з файлами для API програми;
- в) відображення логічної моделі файлової системи на фізичну організацію сховища даних;
- г) організація стійкості файлової системи до збоїв електроживлення, помилок апаратних і програмних засобів;
- д) зміст параметрів файлу, необхідних для правильної його взаємодії з іншими об'єктами системи.

У даній роботі реалізовано стеганографічний метод приховування інформації у структуру файлової системи USB флеш-накопичувачів, тому

розглянуто та проаналізовано такі файлові системи: FAT32, NTFS, exFAT. Так як ці файлові системи є основними файловими системами, що використовуються у флеш-накопичувачах [1, 2, 3, 5].

1.1 Аналіз властивостей та параметрів файлової системи FAT32

FAT32 – це файлова система, розроблена компанією Microsoft, використовує класичну архітектуру файлової системи, а саме таблиці розміщення файлів. Через таку просту реалізацію FAT все ще широко використовується у флеш-накопичувачах.

У файловій системі FAT суміжні сектора носія інформації об'єднуються у логічні одиниці, що називаються кластерами. Кількість секторів у кластері дорівнює ступеню двійки. Для зберігання даних файлу відводиться ціла кількість кластерів, так що, наприклад, якщо розмір файлу складає 40 байт, а розмір кластера 4 Кб, дійсно зайнятий інформацією файлу буде лише 1% відведеного для нього місця. Для уникнення подібних ситуацій доцільно зменшувати розмір кластеру, а для зменшення об'єму адресної інформації та підвищення швидкості файлових операцій – навпаки. Загальна структура файлової системи FAT зображена на рисунку 1.1.

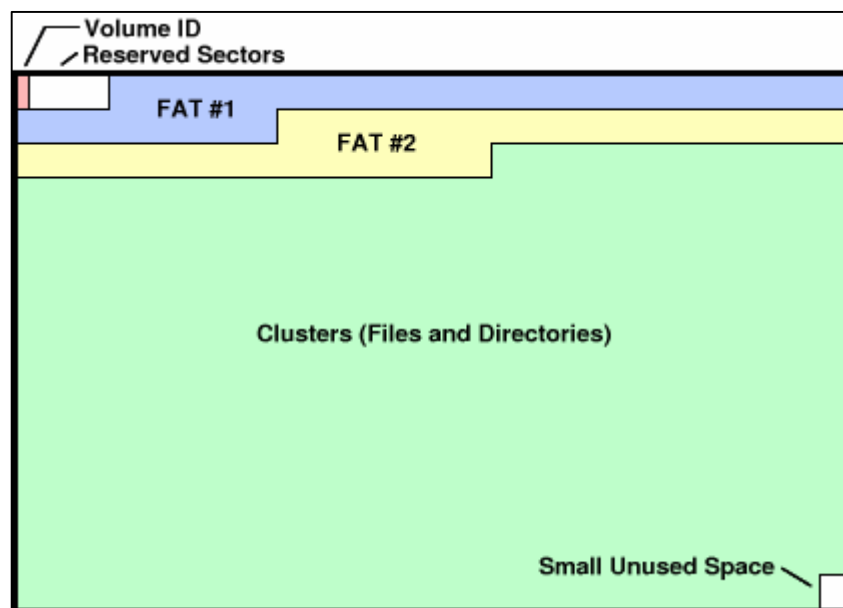


Рисунок 1.1 – Логічні області структури FAT

Простір тому FAT32 логічне розділений на три суміжні області:

- а) зарезервована область – містить службові структури, які належать до завантажувального запису розділу та використовуються при ініціалізації тому;
- б) область таблиці FAT, що містить масив індексів покажчиків, відповідних кластерів області даних, у двох екземплярах;
- в) область даних, де записано власне вміст файлів, а також метадані – інформація відносно імен файлів та папок, їх атрибутів, часу створення та зміни, розміру і розміщення на носії.

У зарезервованій області в першому секторі знаходиться загрузочна інформація, яка містить усі необхідні данні задля роботи драйвера із флеш-накопичувачем:

- а) кількість байт на сектор;
- б) кількість секторів на кластер;
- в) кількість зарезервованих секторів;
- г) кількість FAT таблиць;
- д) номер кластеру кореневого каталогу;
- е) загальна кількість секторів на носії;
- ж) кількість секторів, що займає одна FAT таблиця;

та інша необхідна інформація.

Перший сектор таблиці FAT знаходиться після зарезервованих секторів. Таблиця має вигляд списку 28-ми бітних слів, розташованих у порядку зростання. Значення слова вказує на індекс наступного слова. Таким чином API отримує ланцюг слів, які вказують на кластери де записані данні відповідного файлу. Перші два слова у таблиці зарезервовано, та відповідних кластерів у розділі даних файлів не існує, тобто відлік кластерів у розділі даних починається з другого кластеру. Приклад FAT-таблиці зображено на рисунку 1.2.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000304000	F8	FF	FF	0F	FF	FF	FF	FF	FF	FF	FF	0F	FF	FF	FF	0F
000304010	FF	FF	FF	0F	06	00	00	00	07	00	00	00	08	00	00	00
000304020	09	00	00	00	0A	00	00	00	0B	00	00	00	FF	FF	FF	0F

Рисунок 1.2 – Таблиця індексів розміщення файлів

Слова у таблиці можуть приймати такі значення:

- а) вільний кластер – показчик обнулено;
- б) кластер зайнятий файлом и не є останнім кластером файлу — значення показчика — це номер наступного кластера файлу;
- в) кластер є останнім кластером файлу – показчик має мітку ЕОС (0x0FFFFFFF);
- г) кластер пошкоджено – показчик містить мітку 0x0FFFFFFF7.

Наступною необхідною інформацією для коректної роботи API із накопичувачем є записи у кореновому каталозі. У файловій системі FAT32 кореневий каталог обробляється так само як і інші файли, на відміну від FAT16/12 де для кореневого каталогу виділяється фіксована кількість кластерів одразу після останнього сектора таблиці індексів розміщення файлів.

Кожен запис у кореновому каталозі має фіксований розмір у 32 байти, та містить метадані файлу (рисунок 1.3).

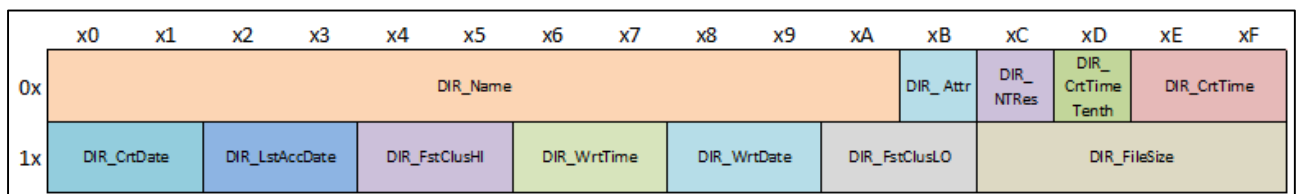


Рисунок 1.3 – Структура записаних метаданих файлу у каталозі

Метадані, які зображені на рисунку 1.3 мають такі значення:

- DIR_Name – 11-ти байтове поле по відносному адресу 0, містить коротке ім'я файлу (в рамках стандарту 8.3);
- DIR_Attr – байт за адресом 0x0B, відповідає за атрибут файлу;
- DIR_NTRes – байт за адресом 0x0C, використовується у Windows NT;

– DIR_CrtTimeTenth – байт за адресом 0x0D, лічильник десятків мілісекунд часу створення файлу, допустимі значення 0-199 (поле найчастіше ігнорується);

– DIR_CrtTime – 2 байти за адресом 0x0E, час створення файлу з точністю до 2-ух секунд;

– DIR_CrtDate – 2 байти за адресом 0x10, дата створення файлу;

– DIR_LstAccDate – 2 байти за адресом 0x12, дата останнього доступу до файлу (тобто останнього зчитування чи запису – в останньому випадку прирівнюється до DIR_WrtDate), аналогічне поле для часу не передбачається;

– DIR_FstClusHI – 2 байти за адресом 0x14, номер першого кластера файлу (старше слово);

– DIR_WrtTime – 2 байти за адресом 0x16, час останньої модифікації файлу;

– DIR_WrtDate – 2 байти за адресом 0x18, дата останньої модифікації файлу, у тому числі створення;

– DIR_FstClusLO – 2 байти за адресом 0x1A, номер першого кластера файлу (молодше слово);

– DIR_FileSize – 4 байти за адресом 0x1C, розмір файлу в байтах.

Запис імен повинен відповідати стандарту 8.3. Тобто 8 символів для імені файлу, 3 символи для розширення. Стандарт дозволяє використовувати будь-яку комбінацію букв та цифр, а також деяких спеціалізованих символів. Під час запису імені файлу усі рядкові літери замінюються заголовними, та якщо довжина імені не відповідає стандарту 8.3, то ім'я доповнюється символами пробілу (0x20). Якщо ім'я не відповідає стандарту 8.3, то його запис представлений у вигляді LFN-запису.

Файлова система FAT32 дозволяє використовувати такі атрибути файлів: скритий (0x02), системний (0x04), мітка тому (0x08), каталог (0x10), архівний (0x20), та атрибут LFN-запису (0x0F) [1, 2].

1.2 Аналіз властивостей та параметрів файлової системи NTFS

NTFS – стандартна файлова система для сімейства операційних систем Windows NT фірми Microsoft. NTFS підтримує зберігання метаданих. З метою поліпшення продуктивності, надійності та ефективного використання дискового простору для зберігання інформації о файлах у NTFS використовуються спеціалізовані структури даних. Інформація о файлах зберігається в головній файловій таблиці – Master File Table (MFT). NTFS підтримує розмежування доступу до інформації для різних користувачів і груп користувачів, а також дозволяє призначати обмеження на використання дискового простору для кожного користувача. Для підвищення надійності файлової системи в NTFS використовується система журналювання USN.

У цілому структура NTFS дещо схожа на структуру FAT32. На початку тому знаходиться завантажувальний запис тому, в якій міститься код завантаження Windows, інформація про том, адреса системних файлів. Завантажувальна займає 16 перших секторів.

В певній області тому знаходиться основна системна структура NTFS – майстер таблиця. У записах цієї таблиці міститься уся інформація про розміщення файлів на носії, а невеликі файли зберігаються безпосередньо у записах MFT.

Важливою особливістю NTFS є те, що уся інформація, як користувачів, так і системна, зберігається у вигляді файлів. Імена системних файлів починаються із символу «\$». Наприклад завантажувальний запис тому знаходиться у файлі «\$Boot», а головна таблиця файлів – у файлі «\$Mft». Така організація інформації дозволяє одночасно обробляти як з файлами користувачів, так і з системними. Приклад структури зображено на рисунку 1.4.

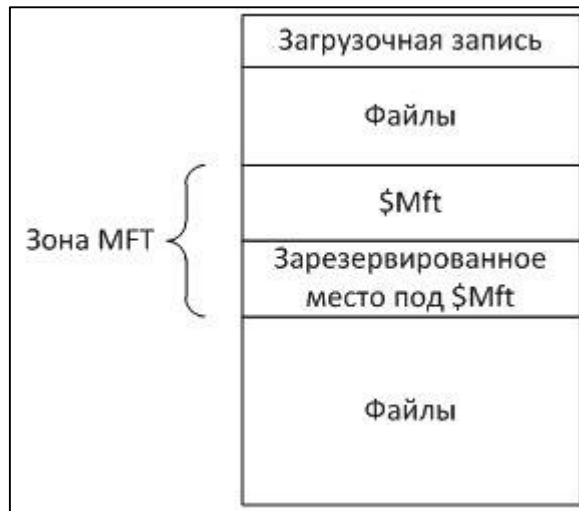


Рисунок 1.4 – Загальна структура файлової системи NTFS

Оскільки MFT є найважливішою системною структурою, до якої при операціях з носієм найбільш часто відбуваються звернення, вигідно зберігати файл «\$Mft» у безперервній області логічного диску, щоб уникнути його фрагментації, та, отже, підвищити швидкість роботи з ним. З цією метою при форматуванні тому виділяється безперервна область, звана зоною MFT. У міру збільшення головної таблиці файлів, файл «\$Mft» розширюється, займаючи зарезервоване місце в зоні.

Залишене місце на томі NTFS відводиться під файли – системні та призначені для користувача [3].

1.3 Аналіз властивостей та параметрів файлової системи exFAT

Файлова система exFAT – пропріетарна файлова система, призначена головним чином для флеш-накопичувачів. Вперше представлена фірмою Microsoft для вбудованих пристроїв.

Основними перевагами exFAT перед попередніми версіями FAT є:

а) зменшення кількості перезаписів одного і того ж сектора, що важливо для флеш-накопичувачів, у яких осередки пам'яті необоротно зношуються після певної кількості операцій запису – це послугувало основною причиною створення exFAT;

- б) теоретичний ліміт на розмір файлу становить 16 ексабайт;
- в) максимальний розмір одного кластеру становить 32 мегабайта;
- г) поліпшення розподілу вільного місця за рахунок введення біт-карти вільного місця, що може зменшувати фрагментацію диска;
- д) введена підтримка списку прав доступу;
- е) підтримка транзакцій.

Основними недоліками exFAT перед іншими файловими системами є, те що exFAT не є продуктом з відкритим кодом, що значно зменшує швидкість розширення цієї файлової системи. А також іншим недоліком є, відносно складна структура, що потребує більше розрахункових ресурсів для роботи з файлами, а саме зчитуванню та запису [4, 5].

1.4 Теоретична та практична оцінка файлових систем FAT32, NTFS, exFAT

Для оцінки даних файлових систем та подальшого їх порівняння у даній роботі для аналізу взяті такі критерії: максимальний розмір носія інформації, максимальний розмір файлу, максимальна кількість файлів, засоби захисту. Данні щодо аналізу файлових систем по даним критеріям представлені у таблиці 1.1,

Таблиця 1.1 – Аналіз параметрів файлових систем

Тип файлової системи	FAT32	NTFS	exFAT
Максимальний розмір носія інформації	8 TiB	16 EiB	∞
Максимальний розмір файлу	4 GiB	16 TiB	16 EiB
Максимальна кількість файлів	1 GiB	1 GiB	∞
Засоби захисту	АФ	АФ, Ш, АВТ	АФ, АВТ

Абревіатури із таблиці 1.1 мають такі значення:

- АФ – можливість привласнювати файлам та каталогам атрибути;

- Ш – можливість шифрувати данні;
- АВТ – можливість розмежування доступу, та введення авторизації, як міра захисту від НСД.

Роблячи висновок можна стверджувати що файлова система exFAT є найбільш ефективною з точки зору зберігання великих за розміром файлів, але на практиці технологія ще не дійшла до можливостей даної файлової системи. Основним недоліком exFAT є, те що ця файлова система є закритою програмною реалізацією компанії Microsoft, що не дає можливості широкого застосування у USB носіях.

Перевагою файлової системи NTFS є, можливості використовуватися на відносно великих за об'ємом пам'яті носіях інформації, саме тому, ця файлова система за замовчуванням використовується на жорстких дисках у персональних комп'ютерах. Наступною перевагою є особливість структури, що використовує збалансоване *n*-арне дерево пошуку зі змінними, така структура дозволяє швидко зчитувати та знаходити інформацію про файли, при великому розмірі носія інформації. Також NTFS має певні заходи захисту інформації що є критичним у роботі з інформацією на персональному робочому місці, а саме, NTFS дозволяє: використовувати шифрування файлів та архівів, проводити розмежування доступу до інформації, надавати квоти чи навпаки обмежувати використання пам'яті носія певними користувачами або групами користувачів, ведення журналу аудиту, що також дозволяє попередити деякі системні помилки, а також використовування атрибутів файлів. Значним недоліком використання NTFS у носіях з малим об'ємом інформації є, те що системні файли займають значне за об'ємом місце. Саме тому ця файлова система не є поширеною на флеш-накопичувачі.

Файлова система FAT32 є найпоширенішою файловою системою при використанні флеш-накопичувачах. Така тенденція пов'язана із тим що у свій час FAT32 був найефективнішою файловою системою, яка задовольняла потреби користувачів, та задовольняє їх і на теперішній час, хоча і поступається у ряді параметрів іншим файловим системам. Основною перевагою FAT32 є, те що

вона сумісна з більшістю операційних систем. Недоліками FAT32 можна вважати відносно малі розміри носіїв інформації та файлів, а також обмежену кількість файлів що можна зберігати у одній директорії.

Надалі представлена часова оцінка файлових систем при виконанні операцій над файлами, а саме: запису 100 файлів по 1 байту, запис 1 файлу об'ємом у 200 МБ, зчитування файлу, видалення файлу, також за ще один критерій взята середня швидкість запису файлу. Задля проведення оцінки використовувався флеш-накопичувач USB DISK 28X (236 МБ) 2006 року виготовлення, персональний ноутбук LENOVA Y-510P. Підключення флеш-накопичувача відбувалася через порт USB2.0. Розмір одного кластеру для кожної файлової системи був фіксованим – 512 байт. Результати занесені до таблиці 1.2.

Роблячи висновки із таблиці 1.2, можна стверджувати, що файлова система FAT32 – є найповільнішою, у базових операціях над файлами, це пов'язано із особливістю структури файлової системи. А NTFS – є файловою системою, яка показала себе, як найшвидша файлова система серед даних, таких результатів NTFS досягає завдяки структурі типу n -арного дерева пошуку. Файлова система exFAT, значно швидша за FAT32, але поступається NTFS, так як використовує у структурі таблицю, а не дерево пошуку [1, 6, 7].

Таблиця 1.2 – Часова оцінка файлових систем

Тип файлової системи		FAT32	NTFS	exFAT
Запис (с)	1 файлу	102,69	80,95	85,43
	100 файлів	12,32	4,17	8,65
Видалення (с)		13,67	0,8	1,2
Зчитування (с)		2,31	1,5	1,74
Середня швидкість запису (МБ/с)		2,1	2,45	2,4

Роблячи загальний висновок можна стверджувати, що кожна з даних файлових систем зайняла певну сферу використання. NTFS – є найпоширенішою

файловою системою у накопичувачах інформації на персональних комп'ютерах під операційною системою сімейства Windows. FAT32 – є найпростішою за структурою файловою системою з відкритою специфікацією, що дає можливість широкого розповсюдження у використанні флеш-накопичувачах бюджетного використання. exFAT – закритий продукт, який здобув переваги у використанні на спеціалізованих пристроях, дана файлова система на відмінно від двох інших, є найбільш затратною з точки зору розрахункових ресурсів.

У даному розділі розглянуто та описано файлові системи, що найчастіше використовуються у флеш-накопичувачах. Також, надано їх порівняльну оцінку, що є вихідними даними для обрання типу файлової системи для розробки стеганографічного методу приховування інформації.

2 РОЗРОБКА ТА ДОСЛІДЖЕННЯ ВЛАСТИВОСТЕЙ АЛГОРИТМУ ПРИХОВУВАННЯ ТА ВИЛУЧЕННЯ ІНФОРМАЦІЇ У КЛАСТЕРИ ФАЙЛОВОЇ СИСТЕМИ

Для розробки та дослідження властивостей алгоритму приховування та вилучення інформації у кластери файлової системи була обрана файлова система FAT32, так як дана файлова система є найпоширенішою з файлових систем. FAT32 має декілька потенційно можливих недоліків що можуть використовуватися у стеганографії.

2.1 Опис властивостей FAT32, що можуть сприяти приховуванню інформації

Загалом у файловій системі для стеганографії можна використовувати такі властивості:

- а) зарезервовані сектора перед першою FAT таблицею;
- б) старші 4 біти у покажчиках вільних кластерів;
- в) залишок останнього сектора FAT таблиці;
- г) перемішування кластерів у файловій системі.

Можливість першого метода полягає у недоліку структури файлової системи FAT32. Зарезервована область на початку логічного тому складає близько 6500 секторів, а вже після цих сектор починається перша FAT таблиця. Хоча із зарезервованих секторів інформаційне навантаження задають лише перші 12 секторів, які містять інформацію щодо структури файлової системи. Таким чином у інші 6488 секторів можна заносити інформацію. Головним недоліком даного метода є легке детектування прихованої інформації та її вилучення. Пропускна здатність мало залежить від розміру одного кластеру, та від загального об'єму пам'яті фізичного носія. Так наприклад, для флеш-накопичувача ємністю у 7.24 Гб можна даним стеганографічним методом

приховати близько 3.2 Мб інформації. Приклад приховування інформації даним методом зображено на рисунку 2.1.

Cluster No.:	n/a	0000019E0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
Reserved sector		0000019F0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 55 AA	UE
Snapshot taken	33 min. ago	000001A00	48 65 6C 6C 6F 20 57 6F	72 6C 64 00 00 00 00 00	Hello World
Physical sector No.:	269	000001A10	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
Logical sector No.:	13	000001A20	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
		000001A30	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	

Рисунок 2.1 – Приклад приховування інформації у зарезервованій сектор

Сутність наступного методу полягає у способі індикації показників вільних кластерів. Для стандартного API показником вільного кластера є той у якого молодші 28 біт обнулено, тобто 0xF0000000 та 0x10000000 – є показниками вільних кластерів. Основними недоліками даного метода є: відносна мала пропускна здатність та відсутність стійкості перед базовим функціоналом файлової системи, тобто додавання нового файлу чи зміна розміру існуючого призведе до зайняття вільних кластерів, що у свою чергу призведе до пошкодження прихованої інформації. Пропускна здатність даного метода напряду залежить від розміру FAT таблиць. Так наприклад, для флеш-накопичувача ємністю у 7.24 Гб, при мінімальному розмірі кластеру у 2048 байт, даним стеганографічним методом можна приховати близько 3.7 Мб інформації. А при максимальному розмірі кластеру 64Кб, можна приховати близько 120 Кб інформації. Приклад приховування інформації даним методом зображено на рисунку 2.2.

Alloc. of visible drive space:	Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Cluster No.:	n/a	000304000	F8	FF	FF	0F	FF	FF	FF	FF	FF	FF	FF	0F	FF	FF	FF
FAT 1	Cluster 5: free	000304010	FF	FF	FF	0F	00	00	00	F0	00	00	00	10	00	00	00

Рисунок 2.2 – Приклад приховування інформації у старші 4 біти показників вільних кластерів

Метод приховування інформації у залишок від останнього сектора FAT таблиці полягає у тому що, розмір таблиці не обов'язково є кратним розміру сектора. Тоді після останнього показника таблиці залишається зарезервоване

місце до кінця даного сектора. Недоліками даного метода є: легке детектування прихованої інформації та її вилучення, а також мала пропускна здатність. Пропускна здатність залежить від розміру FAT таблиці, але дана залежність не є лінійною. Наприклад кількість прихованої інформації може займати мінімум 0 байт, а максимум 1016 байт, у випадку якщо існує друга FAT-таблиця. Приклад приховування інформації даним методом зображено на рисунку 2.3.

Alloc. of visible drive space:		Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
Cluster No.:	n/a	001181F00	00	00	00	00	00	00	00	00	53	74	6E	67	61	6E	6F	67	Steganog
FAT 1	reserved	001181F10	72	61	70	68	79	20	69	73	20	74	68	65	20	70	72	61	raphy is the pra
Snapshot taken	32 min. ago	001181F20	63	74	69	63	65	20	6F	66	20	63	6F	6E	63	65	61	6C	ctice of conceal
Physical sector No.:	36111	001181F30	69	6E	67	20	61	20	66	69	6C	65	2C	20	6D	65	73	73	ing a file, mess
Logical sector No.:	35855	001181F40	61	67	65	2C	20	69	6D	61	67	65	2C	20	6F	72	20	76	age, image, or v
		001181F50	69	64	65	6F	20	77	69	74	68	69	6E	20	61	6E	6F	74	ideo within anot

Рисунок 2.3 – Приклад приховування інформації у залишок сектора

Четвертий метод реалізується за рахунок перестановки кластерів певних покриваючих файлів, таким чином, щоб спільну послідовність покажчиків цих файлів можна було би однозначно відобразити у повідомлення. Недоліком даного метода є його мала пропускна здатність, та відсутність стійкості від дефрагментації накопичувача, видалення чи переміщення файлів. Перевагами є те, що даний метод відносно складно детектувати, та наявність ключів у виді покриваючих файлів надає більшу стійкість до розшифрування. Також даний метод реалізовано за рахунок стандартного функціоналу файлової системи, а не за рахунок недоліків структури. Пропускна здатність прямопропорційно залежить від кількості покриваючих файлів, та має зворотну залежність до розміру кластеру. Приклад даного методу зображено на рисунку 2.4.

F8	FF	FF	0F	FF	FF	FF	FF	FF	FF	FF	0F	FF	FF	FF	0F
FF	FF	FF	0F	06	00	00	00	08	00	00	00	0F	00	00	00
09	00	00	00	0A	00	00	00	0B	00	00	00	0C	00	00	00
0D	00	00	00	0E	00	00	00	10	00	00	00	17	00	00	00
11	00	00	00	12	00	00	00	13	00	00	00	14	00	00	00
15	00	00	00	16	00	00	00	18	00	00	00	1F	00	00	00
19	00	00	00	1A	00	00	00	1B	00	00	00	1C	00	00	00

Рисунок 2.4 – Приклад приховування інформації шляхом перемішування кластерів

Порівнявши параметри даних методів можна зробити висновок, що методи які приховують інформацію у зарезервовану область структури файлової системи мають мінімальну стійкість від детектування та вилучення інформації. Другий метод, не захищений від стандартного функціоналу FAT32. А метод приховування інформації за рахунок перемішування кластерів є найбільш надійним з точки зору стійкості від детектування, але потребує більше розрахункових ресурсів. Результати порівняльного аналізу містяться у таблиці 2.1.

Таблиця 2.1 – Порівняльний аналіз стеганографічних методів приховування інформації у флеш-накопичувач ємністю 7.24 Гб

Відповідний номер методу	I	II	III	IV
Рівень стійкості від детектування (0, 1, 2)	0	1	0	2
Складність реалізації методу (0, 1, 2)	0	1	0	1
Розмір стеганограми	3.2 Мб	0.12 – 3.7 Мб	0 – 1 Кб	0.014 – 2.8 Мб
Операції що можуть призвести до втрати інформації (Ф, С, В)	Ф	Ф, С	Ф	Ф, В

Скорочення із таблиці 2.1 мають такі тлумачення:

а) I – метод приховування інформації у зарезервовані сектора перед першою FAT таблицею;

б) II – метод приховування інформації в старші 4 біти у покажчиках вільних кластерів;

в) III – метод приховування інформації залишок останнього сектора FAT таблиці;

г) IV - метод приховування інформації шляхом перемішування кластерів у файловій системі;

д) Ф – форматування;

е) С – створення нового файлу;

ж) В – видалення певного файлу [2, 8].

2.2 Опис методу приховування даних шляхом перемішування кластерів

Даний стеганографічний метод дозволяє приховувати та вилучати інформацію із структури файлової системи. Для початку виконується зчитування загрузочного сектора, та визначення параметрів необхідних для коректного виконання алгоритму: розмір одного сектора, кількість секторів на кластер, кількість зарезервованих секторів, розмір FAT таблиці у секторах, кількість таблиць. Далі необхідно задати ключ у вигляді послідовності покриваючих файлів.

Покриваючий файл – файл кластери якого перемішуються у відповідній до стеганограми послідовності. Кількість файлів вказує на кількість бітів що позначаєм одним кластером, у співвідношенні як логарифм двійковий. Нехай N – кількість покриваючих файлів, M – кількість інформаційних біт, що відображає один кластер, тоді $M = \log_2 N$. Імена покриваючих файлів повинні бути унікальними, та задані у певній послідовності. Надалі, після задання ключа, виконується або приховування інформації, або її вилучення із структури файлової системи

Приховування інформації виконується у 6 етапів, якщо не враховувати створення повідомлення, що буде приховано у структуру файлової системи. Таким чином, обравши покриваючі файли, наступним кроком буде вилучення початкових кластерів для цих файлів. Загальна схема роботи алгоритму даного етапу зображена на рисунку 2.5.

Після отримання необхідних даних про покриваючі файли переходимо до наступного етапу – вилучення ланцюгів кластерів для кожного файлу. Довжина

кожного ланцюга може бути різною, та значення показчиків повинно бути унікальним для загальної групи ланцюгів, у протилежному випадку – помилка.

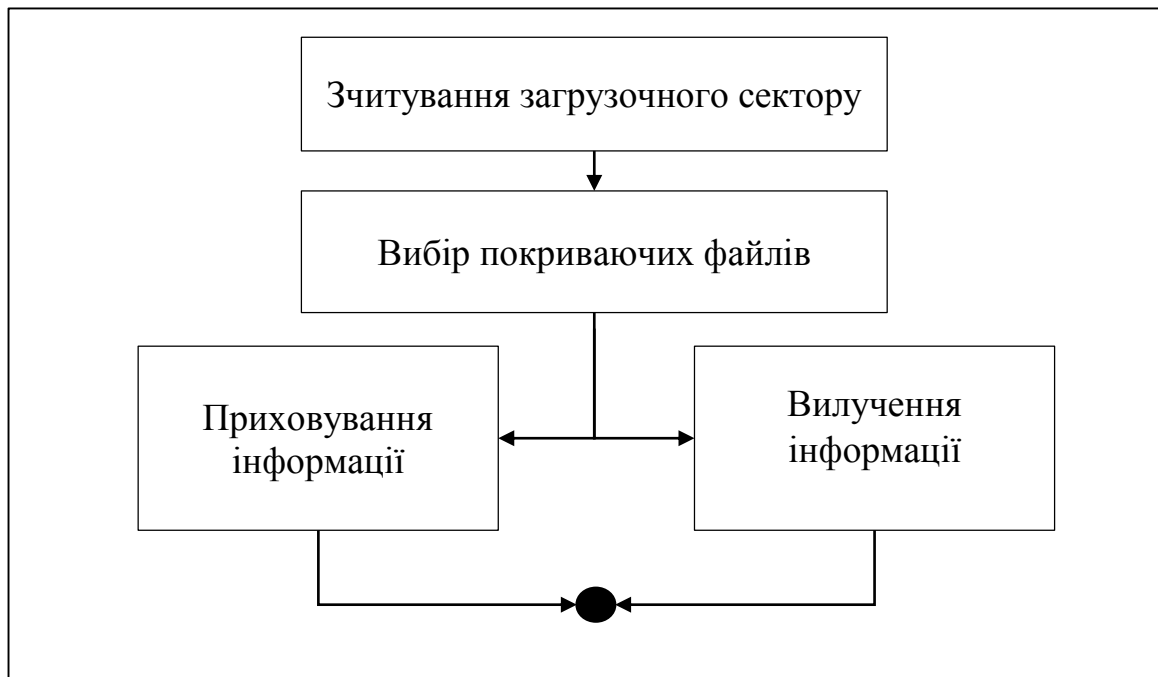


Рисунок 2.5 – Загальний алгоритм роботи приховування та вилучення інформації методом перемішування кластерів

Знаючи початковий кластер, переходимо до FAT таблиці. Встановлюємо каретку на показник початкового кластеру необхідного файлу. Потім до масиву ланцюга кластерів заносимо індекс показника на який вказує каретка. Далі зчитуємо значення показника та переносим каретку на показник індекс якого дорівнює зчитаному значенню. Потім знову заносимо індекс показника до масиву ланцюга кластерів. Цикл повторюється доки зчитане значення не буде рівним мітці ЕОС (0x0FFFFFFF). Приклад вилучення ланцюга кластерів із FAT-таблиці зображено на рисунку 2.6.

Наступним етапом є вилучення даних із покриваючих файлів та збереження їх у сторонньому місці (оперативна пам'ять або тимчасове копіювання на іншій носій). Ця операція є необхідною, інакше інформація що записана на файлах буде втрачена. Далі необхідно перемішати кластери покриваючих файлів у відповідності до блоків стеганограми.

Отримання інформації про початковий кластер																
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
002000100	45	58	41	4D	50	4C	7E	31	44	4F	43	20	00	94	92	64
002000110	BB	48	BB	48	00	00	90	64	BB	48	03	00	27	AF	00	00

Вилучення з FAT таблиці ланцюга кластерів																
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000304000	F8	FF	FF	0F	FF	FF	FF	FF	FF	FF	FF	0F	04	00	00	00
000304010	05	00	00	00	06	00	00	00	07	00	00	00	08	00	00	00
000304020	09	00	00	00	0A	00	00	00	0B	00	00	00	0C	00	00	00
000304030	0D	00	00	00	0E	00	00	00	0F	00	00	00	10	00	00	00
000304040	11	00	00	00	12	00	00	00	13	00	00	00	14	00	00	00
000304050	15	00	00	00	16	00	00	00	17	00	00	00	18	00	00	00
000304060	FF	FF	FF	0F	FF	FF	FF	0F	FF	FF	FF	0F	00	00	00	00

$$Chain_Cluster_{Exempl.doc} = \{3, 4, 5, \dots, 22, 23, 24\};$$

Рисунок 2.6 – Приклад отримання ланцюгу кластерів

Для прикладу, якщо використовується два покриваючих файли то значення блоку стеганограми може бути «0» або «1». Де «0» відповідає кластеру, що належить до першого файлу, а «1» – до другого. Нехай покриваючі файли *A.txt*, та *B.txt*, кожен з них займає по 10 кластерів у структурі файлової системи, та ці кластери фрагментовані, тобто ланцюги кластерів файлів мають такі значення: для файлу *A.txt* – $Chain_Cluster_{A.txt} = \{3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$ та для файлу *B.txt* – $Chain_Cluster_{B.txt} = \{13, 14, 15, 16, 17, 18, 19, 20, 21, 22\}$. Нехай стеганограмою буде повідомлення 0x47. Розбивши це повідомлення на відповідні стеганоблоки отримаємо даний масив $STG = \{0, 1, 0, 0, 0, 1, 1, 1\}$. Отже виконавши етап перемішування кластерів у ланцюгах із відповідністю до масиву із стеганоблоків, отримаємо відповідні ланцюги кластерів: для файлу *A.txt* – $New_Chain_Cluster_{A.txt} = \{3, 5, 6, 7, 11, 12, 13, 14, 15, 16\}$, та для файлу *B.txt* – $New_Chain_Cluster_{B.txt} = \{4, 8, 9, 10, 17, 18, 19, 20, 21, 22\}$. Якщо використовується чотири покриваючих файлів, то значення блоку стеганограми може бути «0», «1», «2», «3». Де «0» – відповідає кластеру, що належить до першого файлу, «1» – другого, «2» – третього, «3» – четвертого.

Після того, як отримано нові ланцюги кластерів, необхідно занести дані покриваючих файлів у відповідні кластери. Таким чином, для користувача та

API, вказані файли не зазнають ніяких змін. Далі необхідно оновити FAT таблицю, та для більшої стійкості до детектування необхідно оновити і другу таблицю, якщо вона є, приклад FAT-таблиці до та після вбудовування повідомлення зображено на рисунку 2.7.

До вбудовування																
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000304000	F8	FF	FF	0F	FF	FF	FF	FF	FF	FF	FF	0F	04	00	00	00
000304010	05	00	00	00	06	00	00	00	07	00	00	00	08	00	00	00
000304020	09	00	00	00	0A	00	00	00	0B	00	00	00	0C	00	00	00
000304030	0D	00	00	00	0E	00	00	00	0F	00	00	00	10	00	00	00

0x47

↓

Після вбудовування																
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000304000	F8	FF	FF	0F	FF	FF	FF	FF	FF	FF	FF	0F	05	00	00	00
000304010	08	00	00	00	06	00	00	00	07	00	00	00	0B	00	00	00
000304020	09	00	00	00	0A	00	00	00	1F	00	00	00	0C	00	00	00
000304030	0D	00	00	00	0E	00	00	00	0F	00	00	00	10	00	00	00

Рисунок 2.7 – Фрагмент FAT таблиці до та після встраювання

Останнім етапом при приховуванні інформації шляхом перемішування кластерів є поновлення метаданих покриваючих файлів, а саме заміна початкового кластеру на дійсний, що отримано після перемішування ланцюгів кластерів, відповідно до стеганограми, у разі якщо не замінити початковий кластер то API не зможе зчитати файл із файлової системи.

Таким чином з точки зору користувача та API структура файлової системи не змінилася, але програми що забезпечують спеціальну оцінку носія та рівня фрагментації файлової системи можуть виявити надлишкову фрагментацію покриваючих файлів.

При вилученні інформації із структури файлової системи для початку необхідно задати покриваючі файли. Далі – отримати ланцюги кластерів для кожного покриваючого файлу. Та останнім етапом, проглянути FAT таблицю, та порівняти показчик кластеру із даними у ланцюгах. Якщо даний показчик

присутній у ланцюгу кластерів одного із покриваючих файлів то до масиву стеганограми заносим відповідне значення, інакше картку зміщуємо на наступний покажчик і так доки не будуть прочитані усі покажчики, що є у ланцюгах кластерів [9, 10, 11].

2.3 Аналіз пропускної здатності методу приховування даних шляхом перемішування кластерів

Пропускна здатність даного методу залежить від розміру одного кластеру, та від кількості покриваючих файлів. Задля знаходження максимальної пропускної здатності допустимо, що:

- покриваючі файли займають усе вільне місце у файловій системі;
- розміри покриваючих файлів рівні по відношенню один до одного;
- кількість кожного типу стеганоблоку однакова, наприклад, якщо стеганоблоки: «0» та «1», то у повідомленні 50% – «1», та 50% – «0».

Нехай $Data$ – загальний об’єм інформації, в байтах, у файловій системі, що займають покриваючі файли, $Data = Const.$ $Size_{cl}$ – розмір одного кластеру у байтах, $Size_{cl} = \{2048, 4096, 8192, \dots, 65536\}$. Num_{cl} – загальна кількість кластерів, займаємих покриваючими файлам. $Size_{fl}$ – розмір, у байтах, одного покриваючого файлу. Num_{fl} – кількість покриваючих файлів. STG_{SIZE} – розмір стеганограми у байтах.

За даними позначеннями пропускна здатність дорівнює співвідношенню STG_{SIZE} до $Data$.

$$\text{Пропускна здатність} = \frac{STG_{SIZE}}{Data} \quad (2.1)$$

Так як $Data$ прийнято за константу, а покриваючі файли рівнозначної величини, то можна стверджувати, що

$$Data = Size_{fl} \times Num_{fl}; \quad (2.2)$$

$$Data = Size_{CL} \times Num_{CL}. \quad (2.3)$$

Так як одним кластером можна відобразити один стеганоблок, а розмір одного стеганоблоку залежить від кількості покриваючих файлів, як логарифм двійковий, то можна стверджувати, що розмір стеганограми у байтах займає:

$$STG_{SIZE} = \frac{Num_{CL} \times \log_2(Num_{FL})}{8}. \quad (2.4)$$

Оцінімо залежність пропускної здатності від розміру одного кластеру, для цього зафіксуємо кількість покриваючих файлів, тобто $Num_{FL} = Const$. А так як розмір покриваючих файлів однаковий то й $Size_{FL} = Const$. Із формул 2.2 та 2.3 можна вивести таку залежність:

$$Num_{CL} = \frac{Size_{FL} \times Num_{FL}}{Size_{CL}}. \quad (2.5)$$

Надалі отриману залежність із формули 2.5 підставимо у формулу 2.4.

$$STG_{SIZE} = \frac{\frac{Size_{FL} \times Num_{FL}}{Size_{CL}} \times \log_2(Num_{FL})}{8}. \quad (2.6)$$

Для спрощення аналізу у формулі 2.6 константні значення необхідно прийняти за одиницю, $\log_2(Num_{FL})$ – для коректності формули, приймаємо рівним одиниці. Отримаємо що $STG_{SIZE} = 1 / (Size_{CL})$, тобто чим більший розмір кластеру, тим менше повідомлення можна приховати, і як висновок, тим менша пропускна здатність методу.

Оцінімо залежність пропускної здатності від кількості покриваючих файлів, для цього зафіксуємо розмір кластеру, тобто $Size_{CL} = Const$. А так як загальна кількість кластерів залежить від розміру кластеру то й $Num_{CL} = Const$. Узявши за основу формулу 2.4, замінімо константні значення одиницею, та отримаємо що $STG_{SIZE} = \log_2(Num_{FL})$. Тобто чим більше покриваючих файлів

тим більше повідомлення можна приховати, і як висновок, тим більша пропускну здатність методу.

Виявивши залежності, для більш коректної оцінки, підставимо числові значення замість $Data$, $Size_{cl}$, Num_{Fl} . Нехай загальний розмір флеш-накопичувача займає 8 Гб, $Data = 7780302848$ байт, кількість покриваючих файлів $Num_{Fl} = 2$, та розмір одного кластеру $Size_{cl} = 2048$ байт. Тоді загальна кількість кластерів $Num_{Cl} = 7780302848/2048 = 3799952$. Підставивши отримані значення у формулу 2.4 знайдемо довжину стеганограми у байтах. $STG_SIZE = 3799952/8 = 474994$ байт. Зафіксувавши кількість файлів, знайдемо довжину стеганограми відповідно для кожного розміру кластеру, $Size_{cl} = \{2048, 4096, 8192, \dots, 65536\}$. Занесемо отримані результати до таблиці 2.2.

Таблиця 2.2 – Залежність розміру стеганограми від розміру кластеру

Розмір кластеру, байт	Кількість кластерів	Розмір стеганограми, байт
2048	3798976	474872
4096	1899488	237436
8192	949744	118718
16384	474872	59359
32768	237436	29680
65536	118718	14840

Побудувавши графік використовуючи дані з таблиці 2.2, можна впевнитись, що дійсно, розмір стеганограми залежить від розміру одного кластеру у зворотній пропорційності, як це зображено на рисунку 2.8.

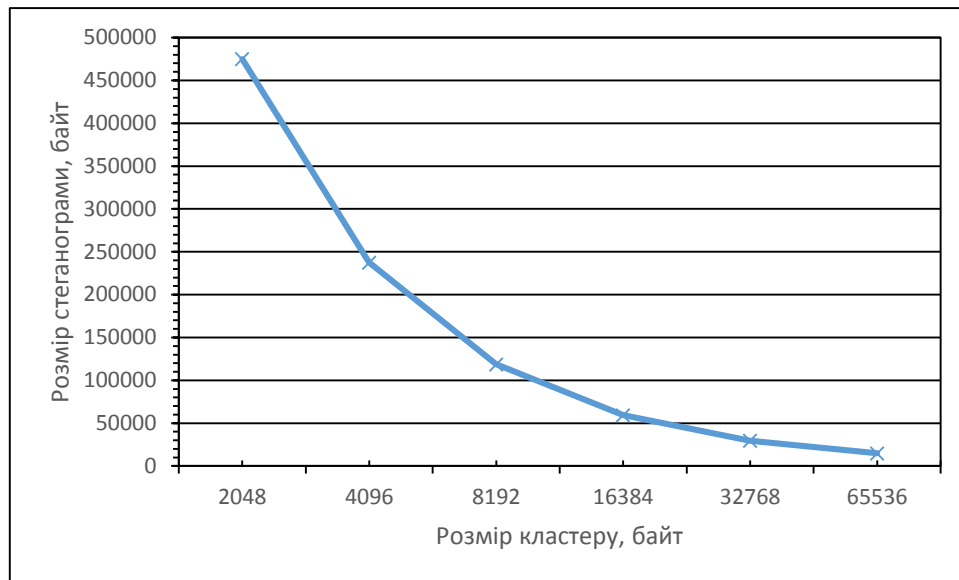


Рисунок 2.8 – Графік залежності розміру стеганограми від розміру кластеру

Аналогічну залежність виявимо і від кількості покриваючих файлів, для цього зафіксуємо загальний розмір накопичувача – $Data = 7780302848$, та розмір одного кластеру – $Size_{cl} = 2048$, кількість покриваючих файлів прийматиме такі значення – $Num_{Fl} = \{2, 4, 8, 16, 32, 64\}$. Хоча максимальна кількість файлів близька до загальної кількості кластерів, але таке велике значення недоцільне при використанні даного методу, так як користувачу буде складніше задавати покриваючі файли та їх значна кількість потребуватиме більше розрахункових ресурсів. Занесемо отримані результати до таблиці 2.3.

Таблиця 2.3 – Залежність розміру стеганограми від кількості покриваючих файлів

Кількість покриваючих файлів	Розмір одного файлу, байт	Розмір стеганограми, байт
2	3 890 151 424	474872
4	1945075712	949744
8	972537856	1424616
16	486268928	1899488
32	243134464	2374360
64	121567232	2849232

Згідно з даними із таблиці 2.3 побудуємо графік залежності розміру стеганограми від кількості покриваючих файлів – рисунок 2.9.

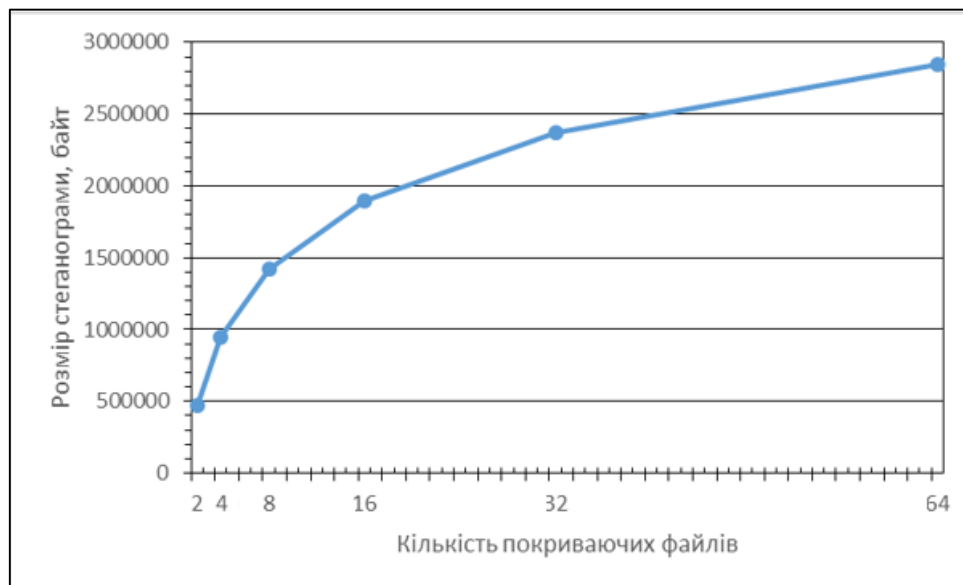


Рисунок 2.9 – Графік залежності розміру стеганограми від кількості покриваючих файлів

Отримавши залежності, наступним кроком буде отримання значень пропускної здатності при усіх можливих розмірах одного кластеру та усіх заданих кількостях покриваючих файлів. Отримані результати занесено до таблиці 2.4.

Таблиця 2.4 – Розмір стеганограми у залежності від кількості покриваючих файлів та розміру одного кластеру

КПФ РК	2	4	8	16	32	64
2048	474872	949744	1424616	1899488	2374360	2849232
4096	237436	474872	712308	949744	1187180	1424616
8192	118718	237436	356154	474872	593590	712308
16384	59359	118718	178077	237436	296795	356154
32768	29680	59359	89039	118718	148398	178077
65536	14840	29680	44519	59359	74199	89039

У таблиці 2.4 аббревіатури мають такі тлумачення:

- а) КПФ – кількість покриваючих файлів;
- б) РК – розмір кластеру у байтах.

Аналізуючи отримані дані із таблиці 2.4, можна стверджувати, що при збільшені розміру кластера у двічі, задля утримання розміру стеганограми, необхідно збільшити кількість покриваючих файлів у другу ступінь.

Роблячи загальний висновок, можна стверджувати, що мінімальний розмір стеганограми – 14.8 Кб, при мінімальній кількості покриваючих файлів – 2, та при максимальному розмірі одного кластеру – 65536 байт. А максимальний розмір стеганограми – 2.8 Мб, при максимальній кількості файлів – 64, та при мініимальному розмірі кластеру – 2048 байт.

У даному розділі описано та проаналізовано властивості файлової системи FAT32, які надають можливість приховати повідомлення, та надано їх порівняльний аналіз. Також надано детальний опис методу приховування інформації шляхом перемішування кластерів. Останнім кроком, виконано дослідження пропускної здатності даного методу, надано закономірності які дозволяють розрахувати розмір повідомлення, яке можна приховати маючи відповідні параметри файлової системи. Отже, отримано вихідні дані задля розробки програмної реалізації метода, що базується на описаній властивості структури FAT32 [11, 12].

3 РОЗРОБКА ПРОГРАМНОЇ РЕАЛІЗАЦІЇ АЛГОРИТМУ ПРИХОВУВАННЯ ІНФОРМАЦІЇ ТА ЕКСПЕРТНЕ ДОСЛІДЖЕННЯ

У даному розділі описано та проаналізовано програмну реалізацію методу приховування та вилучення інформації шляхом перемішування кластерів у файловій системі FAT32, ця програма має назву «SteganoFAT».

Дана програма реалізована із використанням інструментарію мов програмування C++ та C завдяки засобу компіляції Microsoft Visual Studio 2015. Ця програма створена для оцінки часових та кількісних параметрів методу приховування та вилучення інформації шляхом перемішування кластерів.

3.1 Опис програмної реалізації методу приховування та вилучення даних

При написанні програмного коду використовувалися такі стандартні бібліотеки:

- `iostream` – для функцій вводу та виводу даних з консолі;
- `Windows.h` – для функцій маніпулювання консоллю;
- `conio.h` – для функцій керування потоком вводу;
- `math.h` – для математичних функцій;
- `ctime` – для підрахунку часових параметрів алгоритму програми.

У програмному коді окрім стандартних типів даних також використано структури, що дозволяють ефективніше реалізувати даний метод, а саме:

- `FLASH_INFO` – містить дані першого сектору логічного тому, такі як кількість байтів на сектор, кількість секторів на кластер, кількість FAT таблиць, та інші;

- `FILE_INFO` – містить метадані файлу, такі як ім'я, тип, атрибути, перший кластер, розмір у байтах та інші.

Загальну кількість функції можна розбити на чотири категорії: для отримання даних щодо структури файлової системи, для задання та пошуку

інформації о покриваючих файлах, для приховування інформації, для вилучення інформації.

До першої категорії функцій належать:

а) `HANDLE OpenDisk(char ch)` – створює дескриптор логічного тому літера якого задана символом `ch`, повертає покажчик на дескриптор, або у разі помилки повертає «-1»;

б) `int ReadFlashInfo(FLASH_INFO &fl_info, HANDLE flash)` – зчитує інформацію першого сектора потоку на який вказує `flash` у `fl_info`, повертає кількість зчитаних байт.

До другої категорії функцій належать:

а) `FILE_INFO* ChoiceOfFiles(HANDLE fdevice, FLASH_INFO flash_info, int num_files)` – виконує пошук `num_files` файлів, які задав користувач, у тому на який вказує `fdevice`, `flash_info` містить інформацію щодо структури файлової системи, повертає покажчик на масив знайдених файлів, або `NULL` у разі помилки (ця функція використовує інші функції даної категорії);

б) `inline int InputFileNames(char* &file_name)` – виконує зчитування із потоку вводу імен файлів, та записує імена у масив `file_name`, повертає `NULL` у разі успіху, та «-1» у разі помилки;

в) `inline char* CorectName(char* file_name)` – виконує коригування імені `file_name` у відповідності до стандарту 8.3, повертає покажчик на масив відкоригованого імені.

До третьої категорії функцій належить:

а) `int WriteMesage(HANDLE fdevice, FLASH_INFO f_info, FILE_INFO* files, int num_files)` – виконує приховування інформації у логічний диск, на який вказує дескриптор `fdevice`, структурна інформація якого міститься у `f_info`, шляхом перемішування кластерів покриваючих файлів `files` у кількості `num_files` файлів, повертає `NULL` разі успіху (ця функція використовує інші функції даної категорії);

б) `void quickSort(unsigned _int32 arr[], int left, int right)` – виконує сортування масиву `arr` від меншого елемента до більшого, за алгоритмом швидкого сортування, використовуючи граничні значення `left` та `right`.

в) `inline _int32 MinimumSizeFile(FILE_INFO* files, int num_files)` – виконує пошук мінімального за розміром покриваючого файлу з `num_files` `files`, повертає розмір мінімального файлу у байтах;

г) `FILE* FileOpenToRead(int max_mes_byte)` – відкриває бінарний файл для зчитування, у разі успіху повертає покажчик на файл, або `NULL` у разі, якщо файл не вдається відкрити або його розмір більший за `max_mes_byte`;

д) `inline int SizeFile(FILE *f)` – повертає розмір файлу у байтах на якій вказує покажчик `f`;

е) `char* CreateInformArray(char* mes, int size_file, int MASK)` – розбиває вихідне повідомлення `mes` розміром `size_file` на стеганоблоки розміром по `MASK` бітів на блок, повертає масив із стеганоблоків;

ж) `inline unsigned _int32 GetFirstClusterOfFile(FILE_INFO files)` – повертає номер початкового кластеру файлу `files`;

з) `unsigned _int32* GetClusterChain(HANDLE hdevice, FLASH_INFO f_info, unsigned _int32 start_cluster, unsigned _int32 num_cluster)` – виконує вилучення ланцюга кластерів із логічного тому `hdevice`, структурна інформація якого міститься у `f_info`, починаючи із першого кластеру `start_cluster` на довжину ланцюга `num_cluster`, повертає покажчик на масив ланцюга кластерів;

и) `char ** LoadDataFilesToBuf(HANDLE hdevice, FLASH_INFO f_info, unsigned _int32** cluster_chain, unsigned _int32* num_cluster, unsigned _int32 num_files)` – виконує збереження даних `num_files` файлів із кластерів на які вказують ланцюги кластерів `cluster_chain`, відповідних довжин `num_cluster`, із логічного тому `hdevice`, структурна інформація якого міститься у `f_info`, повертає покажчик на масиви збережених даних у разі успіху, або `NULL` у разі помилки;

к) `unsigned _int32** ChainTransformation(unsigned _int32** elder_cluster_chain, unsigned _int32* &start_cluster, unsigned _int32* num_cluster, int num_files, char* stegano_mesage, unsigned _int32 size_stegano_mesage)` –

виконує перемішування `num_files` ланцюгів кластерів `elder_cluster_chain`, із відповідними початковими кластерами `start_cluster` та відповідними довжинами ланцюгів `num_cluster`, у відповідності до масиву стеганоблоків `stegano_mesage` довжиною `size_stegano_mesage`, повертає покажчик на масиви ланцюгів кластерів;

л) `unsigned _int32 LoadDataFilesFromBuf(HANDLE hdevice, FLASH_INFO f_info, unsigned _int32** cluster_chain, unsigned _int32* num_cluster, unsigned _int32 num_files, char** bufer_for_file_data)` – виконує збереження даних `num_files` файлів у кластери на які вказують ланцюги кластерів `cluster_chain`, із відповідними довжинами `num_cluster`, із логічного тому `hdevice`, структурна інформація якого міститься у `f_info`, із масивів даних `bufer_for_file_data`, повертає розмір одного кластеру у байтах у разі успіху;

м) `unsigned _int32 LoadClusterChainToFat(unsigned _int32** cluster_chain, HANDLE hdevice, FLASH_INFO f_info, unsigned _int32* num_cluster, unsigned _int32 num_files)` – виконує поновлення першої FAT таблиці логічного тому `hdevice`, структурна інформація якого міститься у `f_info`, у відповідності до ланцюгів кластерів `cluster_chain`, довжини яких `num_cluster`, у кількості `num_files`, повертає розмір одного сектору у разі успіху та інше число у разі помилки;

н) `int ReloadFileInfo(HANDLE fdevice, FLASH_INFO f_info, FILE_INFO* files, int num_files, unsigned _int32* start_cluster)` – виконує поновлення метаданих файлів `files`, кількістю `num_files`, а саме, заміна номерів початкового кластеру на `start_cluster`, логічного тому `hdevice`, структурна інформація якого міститься у `f_info`, повертає розмір кластеру у разі успіху та інше значення у разі помилки.

До четвертої категорії функцій належить:

а) `int ReadMesage(HANDLE fdevice, FLASH_INFO f_info, FILE_INFO* files, int num_files)` – виконує вилучення інформації із логічного диску, на який вказує дескриптор `fdevice`, структурна інформація якого міститься у `f_info`, шляхом зчитування індексів кластерів покриваючих файлів `files` у кількості

num_files файлів, повертає NULL разі успіху (ця функція використовує інші функції даної категорії);

б) void quicksort_2(unsigned _int32* arr[], int left, int right) – виконує сортування двовимірного масиву arr від меншого елемента до більшого за одним із вимірів, за алгоритмом швидкого сортування, використовуючи граничні значення left та right.

в) FILE* FileOpenToWrite() – відкриває бінарний файл для запису, у разі успіху повертає покажчик на відкритий файл f, або NULL у разі помилки;

г) char* ReadSteaganoFromClusterChain(unsigned _int32** cluster_chain, unsigned _int32* num_cluster, unsigned _int32 num_files, unsigned _int32 size_stegano_mesage) – виконує зчитування прихованої інформації розміром size_stegano_mesage, у відповідності до ланцюгів кластерів cluster_chain, у кількості num_files штук, із відповідними довжинами num_cluster, повертає масив із стеганоблоків у разі успіху;

д) char* CreateMesageFromStg(char* stg, int size_stg, int MASK) – компонує масив із стеганоблоків stg довжиною size_stg, із розміром кожного блоку MASK, у масив байтів, повертає масив байтів у разі успіху.

Усі ці функції реалізують стеганографічний метод приховування інформації у кластери файлової системи.

Обмеженнями при використанні даної програми є :

- розмір оперативної пам'яті має бути в два рази більшим ніж загальний розмір покриваючих файлів;
- необхідно мати 24 Кб вільного місця на дисковому просторі;
- операційна система сімейства Windows, версії Windows XP або сучасніше;
- розрядність операційної системи – 32 біти [2, 8, 12, 13].

3.2 Опис алгоритму роботи програми «SteganoFAT»

Алгоритм роботи програми умовно можна розділити на три етапи, як це показано на рисунку 3.1:

- підготовчий етап – вибір логічного тому, вибір покриваючих файлів;
- приховування повідомлення у структуру файлової системи;
- вилучення повідомлення із структури файлової системи;

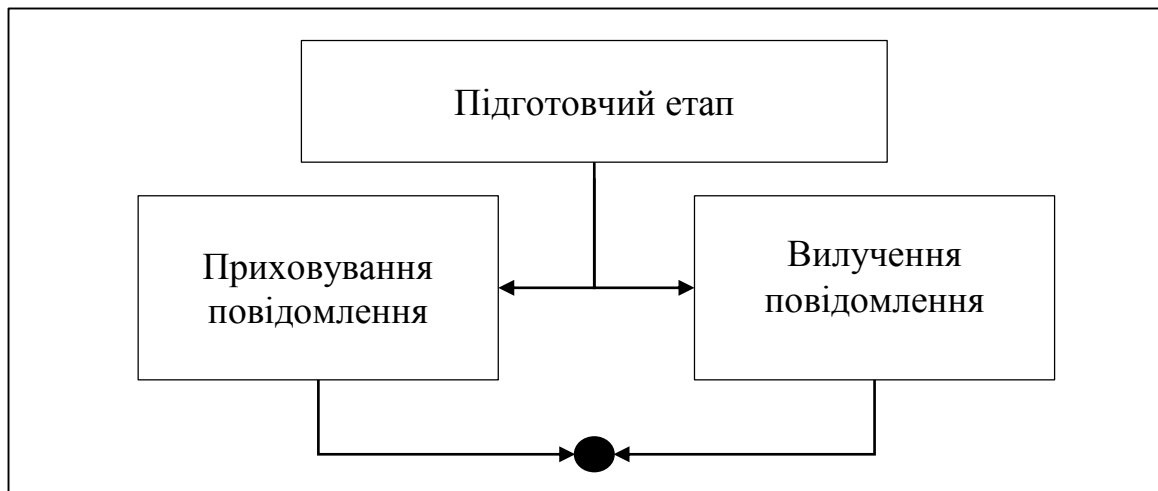


Рисунок 3.1 - Загальний алгоритм приховування та вилучення повідомлення у програмі «SteganoFAT»

До підготовчого етапу належать такі дії, як це проілюстровано на рисунку 3.2.:

- вибір логічного тому;
- зчитування завантажувального запису файлової системи;
- вибір та пошук покриваючих файлів у кореневому каталозі структури файлової системи;

При виборі логічного тому користувачеві необхідно увести відповідну латинську літеру, що відображає певний логічний том. У разі успіху програма «SteganoFAT», захоплює контроль над записом та зчитуванням із обраного тому.

При виборі покриваючих файлів, користувач спочатку задає їх кількість а вже потім імена. На результат виконання алгоритму впливають як коректність

імен файлів так і їх правильна послідовність. У разі якщо «SteganoFAT» знаходить дані файли у кореневому каталозі, то програма переходить до наступного етапу, на вибір користувача.

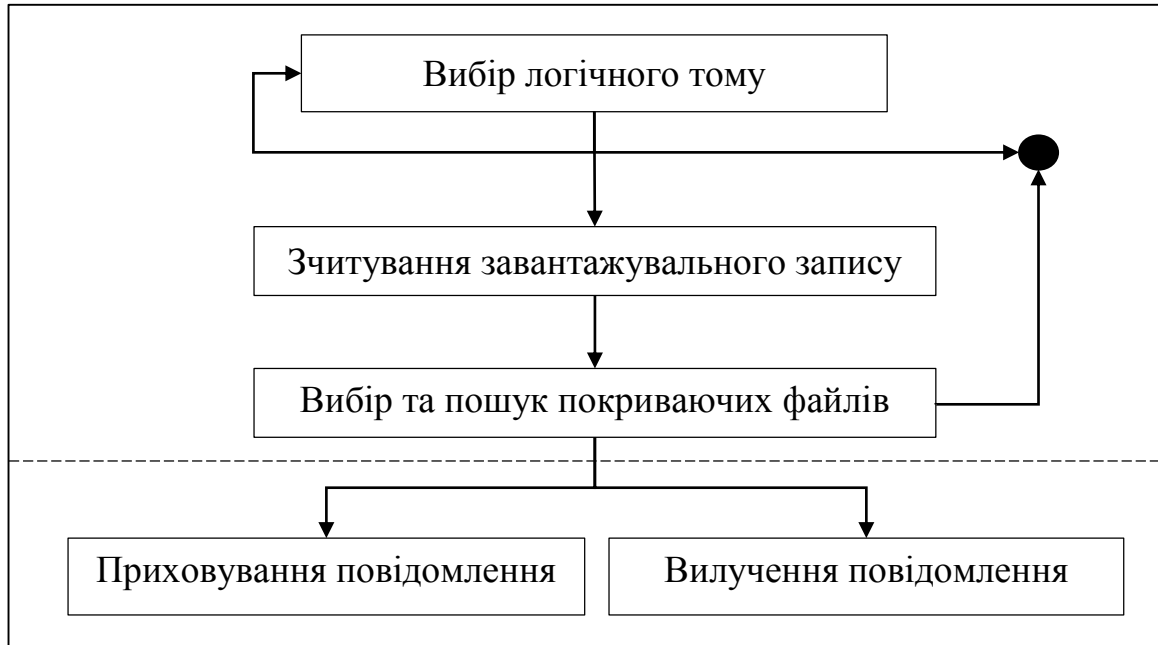
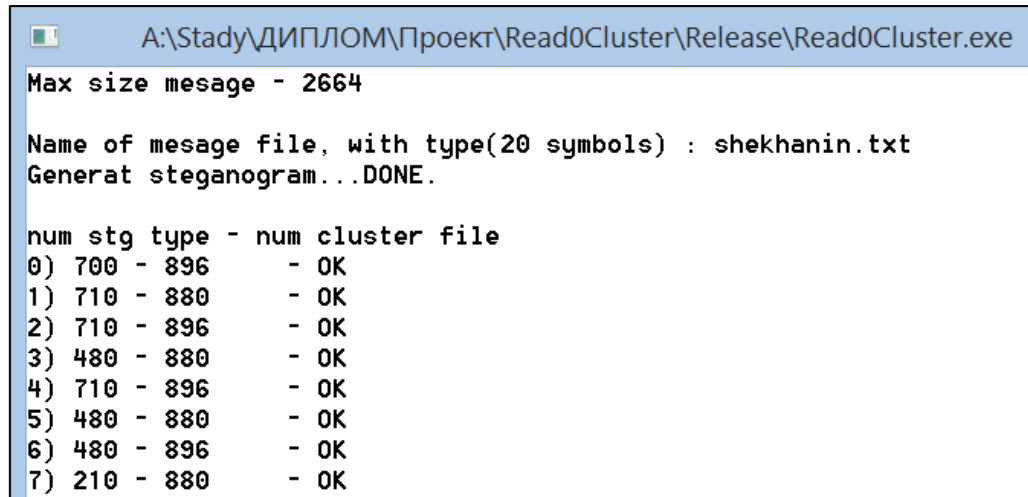


Рисунок 3.2 – Алгоритм виконання підготовчого етапу програми «SteganoFAT»

У разі, якщо користувач обрав виконання алгоритму приховування повідомлення, то програма почне виконувати відповідний етап. Даний етап містить наступні кроки:

- вибір файлу повідомлення – перед виконанням даного кроку користувач отримує повідомлення із грубою оцінкою щодо можливого розміру файлу повідомлення, на цьому кроці користувачу необхідно ввести ім'я файлу, дані якого будуть приховані, у разі некоректного імені крок повторюється спочатку;
- розбивка повідомлення на стеганоблоки – на даному кроці виконується розбивка байтів повідомлення на стеганоблоки розміром відповідним до кількості покриваючих файлів, після виконання даного кроку надається детальна оцінка розміру стеганограми, а саме відображається кількість кожного типу стеганоблоків та кількість кластерів кожного покриваючого файлу (як показано на рисунку 3.3), у разі якщо кількість стеганоблоків хоча би одного типу більша за кількість кластерів відповідного файлу, то алгоритм повертається до першого

кроку – вибір файлу повідомлення, і так доти, доки кількість кожного типу стеганоблоку не буде менша або рівною кількості кластерів відповідних покриваючих файлів;



```

A:\Stady\ДИПЛОМ\Проект\Read0Cluster\Release\Read0Cluster.exe
Max size mesage - 2664
Name of mesage file, with type(20 symbols) : shekhanin.txt
Generat steganogram...DONE.

num stg type - num cluster file
0) 700 - 896 - OK
1) 710 - 880 - OK
2) 710 - 896 - OK
3) 480 - 880 - OK
4) 710 - 896 - OK
5) 480 - 880 - OK
6) 480 - 896 - OK
7) 210 - 880 - OK
  
```

Рисунок 3.3 – Ілюстрація виконання точної оцінки програмою «SteganoFAT»

– отримання ланцюгів кластерів – на даному кроці алгоритму програма вилучає послідовність кластерів (ланцюги) кожного з покриваючих файлів;

– завантаження даних файлів – дані, у відповідності до отриманих ланцюгів кластерів, кожного покриваючого файлу завантажуються у тимчасовий буфер до оперативної пам'яті, даний крок є найзатратнішим з точки зору розрахункових ресурсів;

– перемішування ланцюгів кластерів – на даному кроці виконується, безпосередньо, стеганографічний процес, кластери у ланцюгах перемішуються у відповідності до масиву стеганоблоків, наприклад, якщо задано два покриваючих файли та масив стеганоблоків містить такі значення $stg_2 = \{0, 1, 0, 0, 1, 0, 1, 1\}$, то перший кластер буде належати ланцюгу кластерів першого файлу, другий – другому, третій – першому, четвертий – першому і так далі, як показано на рисунку 3.4;

– вивантаження даних файлів – дані, у відповідності до отриманих перемішаних ланцюгів кластерів, кожного покриваючого файлу вивантажуються із тимчасового буферу у кластери файлової системи, після даного кроку

виконується видалення непотрібних даних із буферу оперативної пам'яті, під час виконання даного кроку та після нього заборонено вилучати флеш накопичувач або припиняти виконання алгоритму, так як це призведе до фатальної помилки;



Рисунок 3.4 – Приклад перемішування ланцюгів кластерів

– оновлення FAT таблиці – на даному кроці виконується заміна значень показників відповідно до перемішаних ланцюгів кластерів;

– оновлення метаданих покриваючих файлів – на даному кроці виконується заміна початкових кластерів покриваючих файлів.

Алгоритм даного етапу відображено на рисунку 3.5.

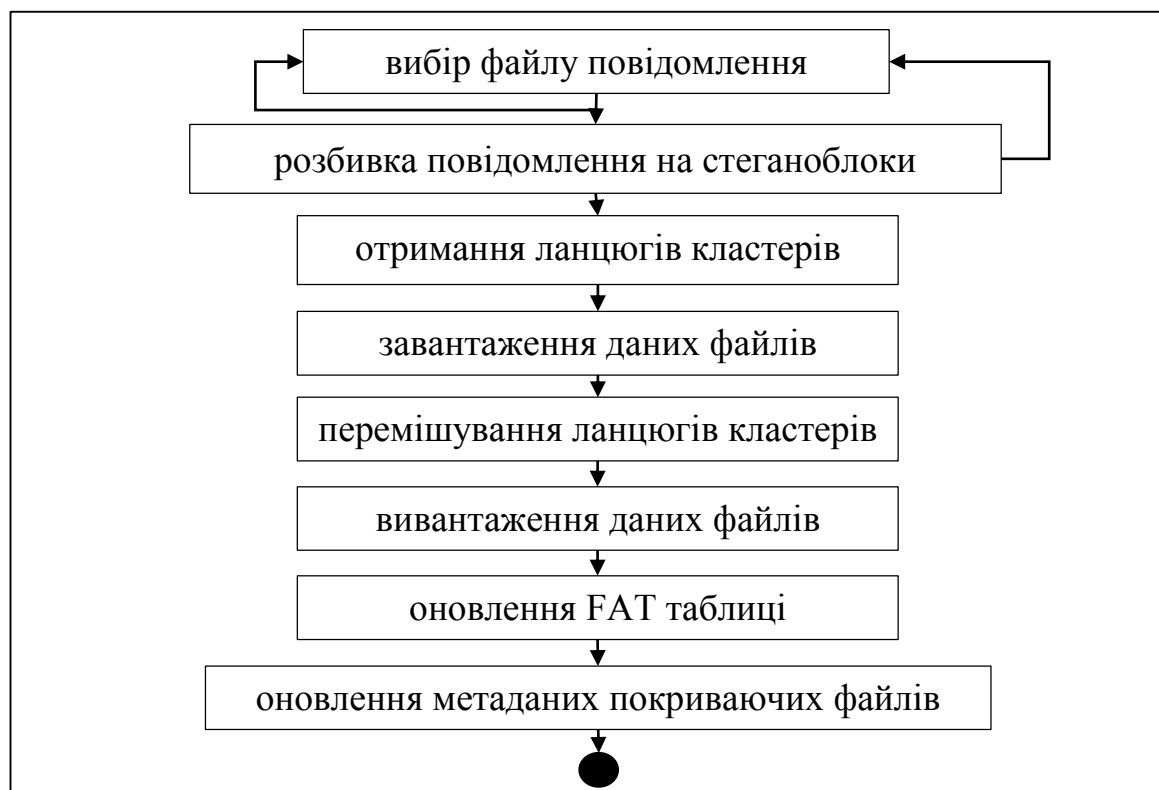


Рисунок 3.5 – Алгоритм приховування повідомлення програми «SteganoFAT»

А у разі вибору користувачем іншого етапу – вилучення інформації, то алгоритм почне виконувати наступні кроки, як показано на рисунку 3.6:

- отримання ланцюгів кластерів – на даному кроці алгоритму програма вилучає послідовність кластерів (ланцюги) кожного з покриваючих файлів;
- зчитування масиву стеганоблоків – у відповідності до ланцюгів кластерів виконується послідовне привласнювання кожному елементу масиву значення, яке вказує на порядок, у відповідності до послідовності вказання користувачем, покриваючого файлу, особливістю зчитування стеганоблоків є те що користувачеві не відома довжина прихованого повідомлення тож, він змушений зчитувати усі кластери які належать до покриваючих файлів, даний крок є умовно оборотною функцією до кроку «перемішування ланцюгів кластерів» із етапу приховування повідомлення;
- склеювання стеганоблоків – під час виконання даного кроку виконується перетворення масиву стеганоблоків у масив байтових елементів, тобто у стеганограму, яка вже несе інформаційне навантаження для користувача;
- створення файлу повідомлення – користувач задає ім'я файлу куди буде збережено стеганограму.

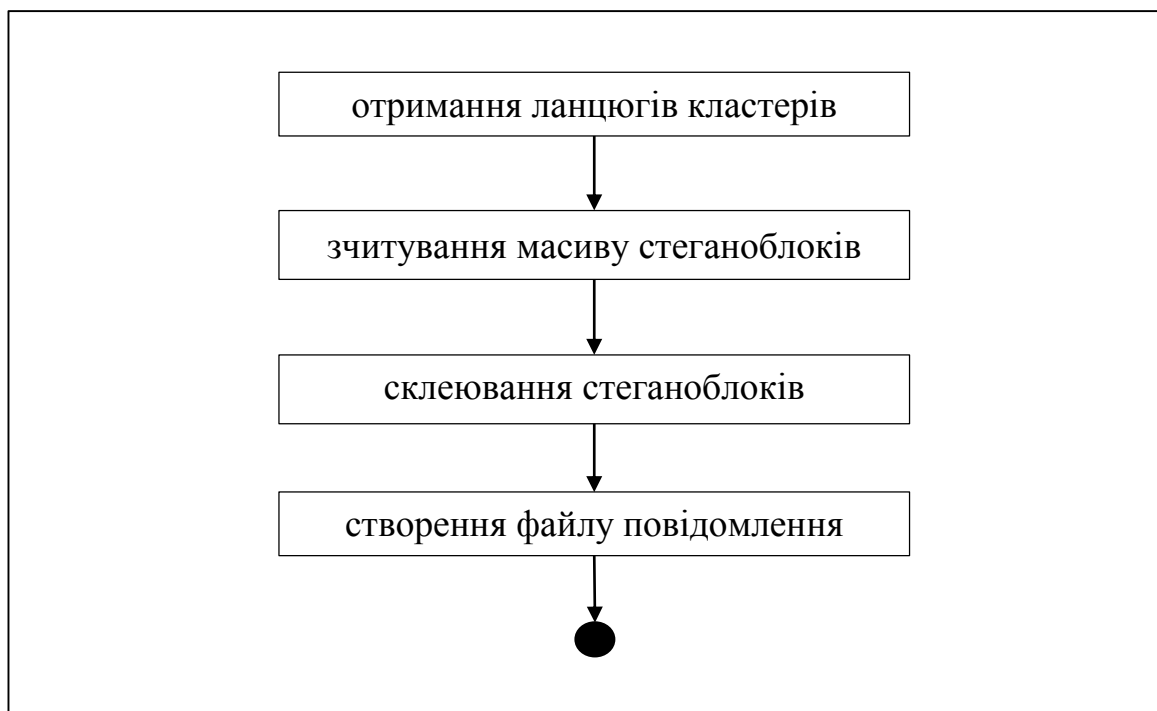


Рисунок 3.6 – Алгоритм вилучення повідомлення програми «SteganoFAT»

Основний функціонал програми «SteganoFAT» закладено на кроках «перемішування ланцюгів кластерів» та «зчитування масиву стеганоблоків» у відповідних етапах виконання алгоритму, тож доцільним буде більш детально розглянути дані кроки.

Крок «перемішування ланцюгів кластерів» приймає такі параметри: ланцюги кластерів, кількість кластерів у кожному ланцюгу, масив із стеганоблоків, довжину цього масиву, кількість покриваючих файлів – як вихідні дані.

Першими, підготовчими операціями даного кроку є:

- створення загального ланцюга кластерів, у який, по черзі, заносим значення кожного ланцюга, довжина загального ланцюга кластерів дорівнює сумі довжин ланцюгів усіх покриваючих файлів;
- сортування загального ланцюга кластерів у порядку зростання;
- створення нових ланцюгів кластерів із такими самими довжинами, але значення кожного елементу дорівнює нулю.

Наступною операцією є заповнення нових ланцюгів кластерів у відповідності до блоку стеганограми. Значення стеганоблоків відповідають за вибір ланцюга кластерів у який буде занесено нове значення із загального ланцюга кластерів. Дана ітерація повторюється доки не буде записано кожен елемент із масиву стеганоблоків. Після цього, нові ланцюги кластерів можуть мати незаповнені елементи, тож наступною операцією є доповнення цих ланцюгів значеннями, що залишилися у загальному ланцюгу кластерів. Спочатку доповнюється ланцюг кластерів, що відповідає за перший, вказаний користувачем, покриваючи файл, після його заповнення, починається заповнення другого ланцюга кластерів, і так доти, доки усі значення із загального ланцюга кластерів не будуть присвоєні новим ланцюгам кластерів. Програмний код даної операції показано на рисунку 3.7, де:

- `size_total_chain` – загальний розмір усіх ланцюгів кластерів;
- `stegano_mesage[]` – масив стеганоблоків;

- left_clusters_number[«приналежність до файлу»] – кількість кластерів, що залишились незаписаними у відповідному новому ланцюгу кластерів;
- num_cluster[«приналежність до файлу»] – кількість кластерів відповідного ланцюга кластерів;
- size_stegano_mesage – кількість стеганоблоків;
- global_chain[] – загальний ланцюг кластерів;
- new_cluster_chain[«приналежність до файлу»][] – двовимірний масив ланцюгів кластерів, перший вимір відповідає за приналежність до покриваючого файлу, другий – номер елементу у ланцюгу.

```

int choice = 0;
for (unsigned _int32 i = 0; i < size_total_chain; i++){
    if (i < size_stegano_mesage)
    {
        new_cluster_chain[stegano_mesage[i]][num_cluster[stegano_mesage[i]]-
        left_clusters_number[stegano_mesage[i]]] = global_chain[i];
        left_clusters_number[stegano_mesage[i]]--;
    }
    else if (left_clusters_number[choice])
    {
        new_cluster_chain[choice][num_cluster[choice] - left_clusters_number[choice]] =
        global_chain[i];
        left_clusters_number[choice]--;
    }
    else { choice++; i--; }
}

```

Рисунок 3.7 – Фрагмент програмного коду кроку «перемішування ланцюгів кластерів»

Крок «зчитування масиву стеганоблоків» приймає такі параметри: ланцюги кластерів, кількість кластерів у кожному ланцюгу, кількість покриваючих файлів – як вихідні дані.

Першими, підготовчими операціями даного кроку є:

- створення двовимірного загального ланцюга кластерів, у який, по черзі, заносим значення кожного ланцюга та порядковий номер покриваючого файлу, щоб знати із якого ланцюга кластерів було узято значення елементу, довжина

загального ланцюга кластерів дорівнює сумі довжин ланцюгів усіх покриваючих файлів;

- сортування загального ланцюга кластерів у порядку зростання значень;

Наступною операцією є запис значень до масиву стеганоблоків із впорядкованого масиву загального ланцюга кластерів. Програмний код даної операції показано на рисунку 3.8, де:

- `size_stegano_mesage` – кількість стеганоблоків;
- `stegano_mesage[]` – масив стеганоблоків;
- `global_chain[][«0\1»]` – загальний ланцюг кластерів, у разі якщо у другому виміру вказано «0», то значення `global_chain[][0]` – номер занесеного кластеру, а якщо «1» - порядковий номер покриваючого файлу [12, 14, 15].

```
for (unsigned _int32 i = 0; i < size_stegano_mesage; i++)
    stegano_mesage[i] = global_chain[i][1];
```

Рисунок 3.8 – Фрагмент програмного коду кроку зчитування масиву стеганоблоків

3.3 Інструкція користувача та рекомендації при роботі із програмою «SteganoFAT»

Так як інтерфейс програми із користувачем виконано у вигляді консольного вікна, то керування потребує деяких роз'яснень.

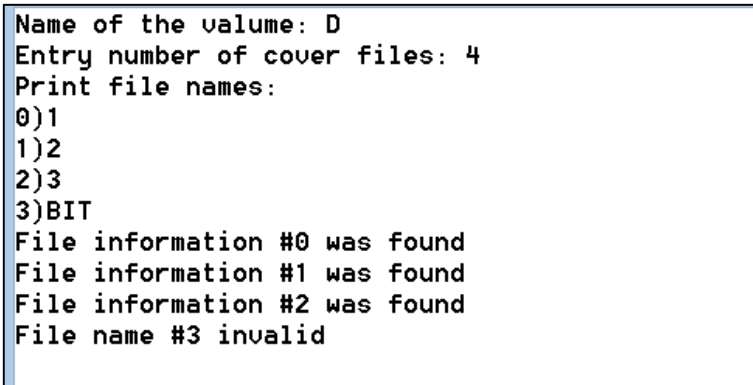
Перед початком роботи із програмою «SteganoFAT» необхідно розмістити файл «SteganoFAT.exe» у будь-якій директорії на персональному робочому місці. Також у тій самій директорії необхідно створити каталог із назвою «message». Усі файли-повідомлення як вилучені, так і ті які будуть записані у структуру файлової системи необхідно розміщати саме у цьому каталозі.

Щоб розпочати виконання алгоритму необхідно запустити файл «SteganoFAT.exe» із правами адміністратора – програма почне виконувати підготовчий етап виконання алгоритму. При появі повідомлення «Name of the

volume:» користувачу необхідно увести заголовну літеру що ідентифікує дескриптор логічного тому, попередньо вставивши у USB-порт флеш-накопичувач із файловою системою FAT32, та натиснути клавішу Enter. У разі виводу на консоль повідомлення «Invalid name | Stop reading processes (close) Press 'q' to quit» користувач або невірно увів ідентифікуючу літеру, або даний логічний том зайнятий іншим процесом, у цьому разі необхідно припинити виконання процесів пов'язаних із даним томом, у тому числі і закрити усі провідники. Після чого користувач може натиснути клавішу «q» – що припинить виконання алгоритму, або будь яку іншу клавішу – що поверне алгоритм до початкового стану.

У разі, якщо, ідентифікуюча літера введена вірно, то на консолі з'явиться повідомлення – «Entry number of cover files:». У цьому випадку користувачу необхідно увести число, ступінь двійки, що відповідає за кількість покриваючих файлів. Після чого користувачу на заявлене повідомлення «Print file names:» необхідно увести імена усіх покриваючих файлів, без задання розширення файлу.

Якщо хоча би одне з імен файлів введено не вірно, то «SteganoFAT» повідоме про це відповідним повідомленням та припинить виконання алгоритму, як це показано на рисунку 3.9.



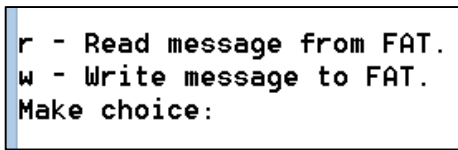
```
Name of the volume: D
Entry number of cover files: 4
Print file names:
0)1
1)2
2)3
3)BIT
File information #0 was found
File information #1 was found
File information #2 was found
File name #3 invalid
```

Рисунок 3.9 – Приклад інтерфейсу програми «SteganoFAT» при невірно заданому імені покриваючого файлу

У іншому випадку користувачу запропонують обрати один із етапів виконання алгоритму, натиснувши на відповідну клавішу:

- w – програма почне виконувати етап «приховування повідомлення у структуру файлової системи»;
- r – програма почне виконувати етап «вилучення повідомлення із структуру файлової системи»;
- будь-яка інша клавіша – програма завершить виконання алгоритму.

Приклад діалогового вікна зображено на рисунку 3.10.



```

r - Read message from FAT.
w - Write message to FAT.
Make choice:
  
```

Рисунок 3.10 – Приклад інтерфейсу програми «SteganoFAT» при виборі наступного етапу алгоритму

У разі якщо користувач обрав етап приховування повідомлення, то діалогове вікно заміниться на відповідне – на консолі з’явиться повідомлення про максимальний розмір файлу повідомлення, у байтах, та запропонують увести ім’я файлу (із розширенням). Даний файл повинен знаходитись у директорії «message». У разі введення некоректного імені файлу-повідомлення даний етап почнеться спочатку. В іншому випадку, користувач отримає інформацію про точну оцінку можливості приховування даного повідомлення у кластери обраних покриваючих файлів. Якщо точна оцінка вказує на неможливість приховування, то даний етап почнеться спочатку. У іншому випадку алгоритм роботи продовжиться автоматично. Користувач лише зможе спостерігати за виконанням алгоритму, та отримати часову оцінку, у секундах, виконання кожного кроку даного етапу. У разі якщо крок виконано швидше ніж за 1 мілісекунду, то його оцінка часу дорівнює нулю.

Після закінчення виконання етапу, на консолі з’явиться повідомлення «END», та натиснувши будь-яку клавішу клавіатури, користувач завершить

виконання програми «SteganoFAT». Приклад виконання етапу приховування інформації зображено на рисунку 3.11.

```

Max size mesage - 888

Name of mesage file, with type(20 symbols) : 1.txt
Incorect file name!

Name of mesage file, with type(20 symbols) : 1.txt
Generat steganogram...DONE.          0.000000

num stg type - num cluster file
0) 2660 - 896      - INVALID
1) 2718 - 880      - INVALID
2) 2658 - 896      - INVALID
3) 2624 - 880      - INVALID

Name of mesage file, with type(20 symbols) : shekhanin.txt
Generat steganogram...DONE.          0.000000

num stg type - num cluster file
0) 386 - 896       - OK
1) 397 - 880       - OK
2) 339 - 896       - OK
3) 178 - 880       - OK

Get cluster chain...DONE.             0.030000
Load data file to bufer...DONE.       1.986000
Transformation cluster chain...DONE.   0.001000
Load data file to flash...DONE.       9.509000
Load cluster chain to fat...DONE.      0.298000
Reoaled files metadata...DONE.        0.005000

Total time for write mesage - 11.829000

END

```

Рисунок 3.11 – Приклад інтерфейсу програми «SteganoFAT» при виконанні алгоритму етапу приховування повідомлення

У разі якщо користувач обрав етап вилучення повідомлення, то алгоритм етапу почне виконуватись автоматично, надаючи користувачу інформацію про виконані кроки алгоритму та затрачений час на їх виконання. Перед останнім кроком користувачу необхідно увести ім'я файлу у який буде записано вилучене повідомлення. Приклад інтерфейсу виконання алгоритму даного етапу зображено на рисунку 3.12.

```

Get cluster chain...DONE.                0.041000
Reading steganoblocks...DONE.            0.001000
Conglutination message...DONE.           0.000000

Name of mesage file, with type(20 symbols) : tapok.NET

Total time for read mesage - 0.043000

END

```

Рисунок 3.12 – Приклад інтерфейсу програми «SteganoFAT» при виконанні алгоритму етапу вилучення повідомлення

При використанні програми «SteganoFAT» рекомендується дотримуватись певних правил, що значно збільшать ефективність алгоритму:

- усі покриваючі файли повинні знаходитися у кореневому каталозі;
- загальна кількість файлів і директорій у кореневому каталозі не повинна перевищувати розмір кластеру поділений на 32;
- імена покриваючих файлів повинні бути записані у вигляді стандарту 8.3;
- для більшої пропускної здатності рекомендується відформувати флеш-накопичувач у FAT32 із мінімально допустимим розміром кластеру;
- розміри покриваючих файлів бажано обирати рівнозначними, так як, пропускна здатність залежить від мінімального розміру одного з покриваючих файлів;
- бажано обирати такі покриваючі файли, щоб при приховуванні повідомлення якомога більше кластерів несли інформаційне навантаження, наприклад, якщо повідомлення містить 10 байт, а загальна кількість кластерів дозволяє приховати 1000 байт, то розрахункових ресурсів буде затрачено, як для приховування 1000 байтового повідомлення;
- у разі, якщо, при створенні стеганоблоків із масиву повідомлення, точна оцінка вказує на те що деякі типи стеганоблоків майже не використовуються, а інших навпаки забагато, рекомендується обрати меншу кількість покриваючих файлів але із більшим загальним розміром;

- для більшої пропускної здатності рекомендується обирати більше покриваючих файлів;
- ім'я покриваючих файлів повинні бути унікальними по відношенню до інших файлів у обраному логічному томі [12, 14, 16, 17].

3.4 Аналіз часових параметрів та пропускної здатності програмної реалізації методу приховування та вилучення інформації у кластери файлової системи

Для аналізу часових параметрів даного методу була використана програма «SteganoFAT», файлова система FAT32 на флеш накопичувачу JetFlash 350 Transcend ємністю 8 Гб, інтерфейс підключення USB2.0 та ноутбук Lenovo Y510P.

Аналізуючи час виконання методу, будемо змінювати такі параметри:

- розмір кластеру;
- розмір файлу повідомлення;
- кількість покриваючих файлів;
- загальний розмір покриваючих файлів.

Для аналізу залежності затраченого часу, на виконання методу приховування та вилучення повідомлення, від розміру кластеру зафіксуємо розмір повідомлення – 100 байт, кількість покриваючих файлів – 2, загальний розмір покриваючих файлів 7 Мб. Та будемо змінювати розмір кластеру – 2048, 4096, 8192 байт. Дані аналізу вказані у таблиці 3.1.

Таблиця 3.1 – Залежність часових параметрів методу від розміру кластеру

Розмір одного кластеру, байт	2048	4096	8192
Час приховування повідомлення, секунд	3.341	2.87	2.37
Час вилучення повідомлення, секунд	0.022	0.012	0.008

Як видно із таблиці 3.1, при збільшені розміру кластеру час на приховування та вилучення повідомлення зменшується, але й пропускна спроможність зменшується у двічі.

Для оцінки залежності затраченого часу, на виконання методу приховування та вилучення повідомлення, від розміру повідомлення зафіксуємо розмір кластеру – 2048 байт, кількість покриваючих файлів – 2, загальний розмір покриваючих файлів 7 Мб. Та будемо змінювати розмір повідомлення – 100, 200, 400 байт. Дані аналізу вказані у таблиці 3.2.

Таблиця 3.2 – Залежність часових параметрів методу від розміру повідомлення

Розмір повідомлення, байт	100	200	400
Час приховування повідомлення, секунд	3.82	5.839	8.36
Час вилучення повідомлення, секунд	0.019	0.024	0.032

Як видно із таблиці 3.2, при збільшені розміру повідомлення, час на приховування та вилучення повідомлення збільшується.

Для оцінки залежності затраченого часу, на виконання методу приховування та вилучення повідомлення, від кількості покриваючих файлів зафіксуємо розмір кластеру – 2048 байт, розмір повідомлення – 100 байт, загальний розмір покриваючих файлів 7 Мб. Та будемо змінювати кількість покриваючих файлів – 2, 4, 8 штук. Дані аналізу вказані у таблиці 3.3.

Таблиця 3.3 – Залежність часових параметрів методу від кількості покриваючих файлів

Кількість покриваючих файлів	2	4	8
Час приховування повідомлення, секунд	4.693	2.76	2.704
Час вилучення повідомлення, секунд	0.022	0.025	0.032

Як видно із таблиці 3.3, при збільшені кількості покриваючих файлів час на приховування повідомлення зменшується, це пов'язано із тим, що кількість інформаційних кластерів, при збільшені покриваючих файлів, зменшується, та відповідно збільшується кількість кластерів, які будуть записані впорядковано, а не перемішано. При вилученні повідомлення, затрачений час збільшується із кількістю покриваючих файлів.

Для оцінки останньої залежності затраченого часу, на виконання методу приховування та вилучення повідомлення, від загального розміру покриваючих файлів зафіксуємо розмір кластеру – 2048 байт, кількість покриваючих файлів – 2, розмір повідомлення 100 байт. Та будемо змінювати загальний розмір покриваючих файлів – 1.7, 3.5, 7 Мб. Дані аналізу вказані у таблиці 3.4.

Таблиця 3.4 – Залежність часових параметрів методу від загального розміру покриваючих файлів

Загальний розмір покриваючих файлів, Мб	1.7	3.5	7
Час приховування повідомлення, секунд	2.083	2.417	3.293
Час вилучення повідомлення, секунд	0.007	0.012	0.021

Як видно із таблиці 3.4, при збільшені загального розміру покриваючих файлів час на приховування та вилучення повідомлення збільшується.

Роблячи загальний висновок з аналізу часових параметрів методу приховування та вилучення інформації можна стверджувати, що:

– при збільшені розміру кластеру метод приховування та вилучення інформації виконується швидше, але при збільшені розміру кластеру, зменшується пропускна здатність, тож не рекомендується це робити задля пришвидшення виконання алгоритму;

– при збільшені розміру повідомлення витрачений час на приховування та вилучення інформації також збільшується, здебільшого це пов'язано із тим що більша кількість кластерів буде перемішана, що у свою чергу призведе до

збільшення рівня фрагментації даних файлів та відповідно збільшення часу операції над цими файлами;

– при збільшені кількості покриваючих файлів витрачений час на приховування інформації зменшується, це також здебільшого пов'язано із рівнем фрагментації покриваючих файлів, а час на вилучення інформації збільшується, але цим параметром можна знехтувати, так як зміну цього часу користувач не помітить;

– при збільшені загального розміру покриваючих файлів витрачений час на приховування та вилучення інформації також збільшується, але у такому випадку розмір повідомлення збільшується у двічі, а час не більше ніж на 50% від попереднього значення.

Щодо оцінки пропускну здатності, необхідно впевнитись, що практичні результати будуть співпадати із теоретичними, отриманими у розділі 2. Для цього: зафіксуємо загальний розмір покриваючих файлів – 24,7 Мб, згенеруємо покриваючі файли із відповідними довжинами. Наступним кроком буде визначення розміру стеганограми у залежності від кількості покриваючих файлів та від розміру одного кластеру. Кількість покриваючих файлів будемо змінювати як – 2, 4, 8, а розмір кластеру як – 2048, 4096, 8192 байт. А також підрахуємо теоретичні значення підставивши обрані дані у формулу 2.6. Отримані результати дослідів занесені у таблицю 3.5.

Таблиця 3.5 – Розмір стеганограми у залежності від кількості покриваючих файлів та розміру одного кластеру (практичне\теоретичне значення)

КПФ РК	2	4	8
2048	1758\1757	3516\3516	5274\5273
4096	879\878	1758\1758	2637\2637
8192	439\439	879\879	1320\1318

Скорочення у таблиці 3.5 мають відповідне тлумачення:

- а) КПФ – кількість покриваючих файлів;
- б) РК – розмір кластеру у байтах.

Порівнявши отримані практичні значення із теоретичними можна зробити висновок, що формули по розрахунку пропускної здатності коректні. Погрішність у 1, 2 байти можна списати на те що при теоретичних розрахунках дробові числа після ділення округлюються у меншу сторону. Отже дані формули можна використовувати для оцінки пропускної здатності при інших розрахунках.

Узагальнюючи матеріали даного розділу, можна, стверджувати, що дана програмна реалізація має певні недоліки і переваги перед іншими варіантами реалізації. До недоліків можна віднести:

- виділення масиву під данні покриваючого файлу у вигляді монолітного блоку, даний варіант реалізації значно обмежує можливості щодо розміру покриваючого файлу, шляхів оптимізації два:

- а) створення буферу у вигляді двозв'язного списку, у разі такої реалізації уся оперативна пам'ять заповнюється повністю, але робота із таким форматом даних потребує більше часу;

- б) виконання по-кластерного перемішування даних файлів, тобто у оперативну пам'ять не завантажується увесь файл, а лише дані того кластеру, які після наступної ітерації будуть втрачені, даний варіант реалізації потребує значних змін у виконанні алгоритму приховування повідомлення;

- значне зростання витраченого часу на приховування повідомлення у разі збільшення розміру повідомлення – більшість часу займають, саме, кроки «завантаження даних файлу у буфер» та «вивантаження даних із буферу», шляхом оптимізації, також, може бути варіант «б» попереднього пункту, що дозволить у разі закінчення стеганограми припинити перемішування даних кластерів, що у свою чергу пришвидшить алгоритм приховування інформації. До переваг даного варіанту реалізації можна віднести:

- максимальний розмір файлу-повідомлення дорівнює 2^{32} байт (4 Гб) при 64-ох покриваючих файлах;

– наочність кожного кроку виконання етапу із відображення затраченого часу.

Отже для збільшення пропускної спроможності рекомендується збільшувати кількість покриваючих файлів, але це ствердження вірне, якщо повідомлення матиме рівну кількість кожного типу стеганоблоків. Для прикладу, якщо повідомлення складається лише з одиничних бітів, то збільшення кількості покриваючих файлів навпаки призведе до негативного результату. У такому випадку рекомендується архівувати повідомлення та\або зменшувати кількість покриваючих файлів при цьому збільшуючи їх загальний розмір.

У даному розділу було описано особливості програмної реалізації та алгоритм виконання методу приховування повідомлення у кластери файлової системи FAT32, надано інструкцію користувача та рекомендації щодо експлуатації даного програмного забезпечення. Також проаналізовано кількісні параметри програми, а саме: час виконання алгоритму та пропускна здатність стеганоконтейнеру. Із цих досліджень слідує що, час виконання алгоритму залежить від розміру повідомлення, загального розміру покриваючих файлів, та від розміру одного кластера, а пропуску здатність, отримана практично, відповідає теоретичним розрахункам [12, 18].

ВИСНОВКИ

У даній роботі розглянуто та описано файлові системи, що найчастіше використовуються у флеш-накопичувачах. Також, надано результати порівняльного аналізу обраних файлових систем, описано та проаналізовано властивості FAT32, які надають можливість приховати повідомлення у структурі файлової системи. За результатами порівняльного аналізу обрано властивість на основі якої розроблено та оцінено алгоритм приховування та вилучення інформації, а саме – приховування та вилучення інформації шляхом перемішування кластерів у структурі файлової системи FAT32.

Задля даного алгоритму виведено розрахункові формули що, надають можливість оцінити розмір файлу-повідомлення, який можна приховати у структуру FAT32 із відповідними параметрами.

Також розроблено програмну реалізацію даного алгоритму задля оцінки часових параметрів та практичної оцінки пропускної здатності. За результатами дослідження програмної реалізації, зроблено висновки, що, з точки зору часових параметрів, даний алгоритм є ефективним у разі невеликого за розміром повідомлення, так як при збільшенні розміру повідомлення, час приховування зростає у квадраті. За результатами практичної оцінки пропускної здатності, можна стверджувати, що розрахункові формули – коректні.

За результатами даної роботи отримано програмну реалізацію методу приховування інформації шляхом перемішування кластерів файлової системи FAT32. Отримано часові характеристики реалізованого методу, а саме час приховування інформації та час зчитування даних, також результати аналізу пропускної здатності. Усі результати зведені у таблиці і графіки.

Дана програмна реалізація має схожі аналоги, та поступається їм у сфері зручності використання програмної реалізації, та ефективності використання розрахункових ресурсів персонального комп'ютера.

Отримані результати можуть подальше використовуватись у науково-дослідницькій діяльності, так як дана робота містить дані про аналіз ефективності цього методу технічної стеганографії.

ПЕРЕЛІК ПОСИЛАНЬ

1. Обмеження файлової системи FAT32 [Електронний ресурс]/ Група технічної підтримки Microsoft – Режим доступу: [www/ URL: http://support.microsoft.com/en-en/184006/](http://support.microsoft.com/en-en/184006/) – 19.05.2016 р. – Загл. з екрану;
2. Microsoft Extensible Firmware Initiative FAT32 File System Specification Version 1.03 [Текст] : Затв. групою технічної підтримки Microsoft 12.06.2006. – Microsoft Corporation, 2006. – 34 с.;
3. Руссон Р. NTFS Documentation [Текст] / Р. Руссон , Ю. Фледел; Microsoft Corporation – 2006. – 260 с.;
4. Файлова система exFAT [Електронний ресурс] / Група технічної підтримки Microsoft. – Режим доступу : [www/ URL: http://macdaily.me/reviews/exfat-universal-file-system/](http://macdaily.me/reviews/exfat-universal-file-system/) – 26.05.2016 р. – Загл. з екрану;
5. Shullich, R. Reverse Engineering the Microsoft Extended FAT File System (exFAT) [Текст] / R. Shullich. – SANS Institute, 2009. – 86 р.;
6. Порівняння файлових систем NTFS і FAT32 [Електронний ресурс]/ Група технічної підтримки Microsoft. – Режим доступу: [www/ URL: http://windows.microsoft.com/uk-ua/windows7/comparing-ntfs-and-fat32-file-systems/](http://windows.microsoft.com/uk-ua/windows7/comparing-ntfs-and-fat32-file-systems/) – 26.05.2016 р. – Загл. з екрану;
7. Сравнение файловых систем (Windows Embedded Standard 2009) [Електронний ресурс] / Група технічної підтримки Microsoft. – Режим доступу: [www/ URL: https://msdn.microsoft.com/ru-ru/library/bb521542\(v=winembedded.51\).aspx](https://msdn.microsoft.com/ru-ru/library/bb521542(v=winembedded.51).aspx) – 26.05.2016 р. – Загл. з екрану;
8. Moulin P. Information-theoretic analysis of information hiding [Текст] / P. Moulin, J. O’Sullivan // IEEE TRANSACTIONS ON INFORMATION THEORY. – 2003. – VOL. 49, NO. 3. – p. 43 с.;
9. Petitcolas F. Information Hiding [Текст] / F. Petitcolas, R. J. Anderson , M. G. Kuhn // Proceedings IEEE, 1999. - Vol. 87, №. 7. - p. 1069-1078;

10. Anderson R. Workshop on Information Hiding: Lecture Notes in Computer Science [Текст] / R. Anderson // Springer-Verlag, Cambridge. - 1996. - Vol. 78, №. 4. - p. 578 -601;

11. Covert Channel for Cluster-based File Systems Using Multiple Cover Files [Текст] / N. Morkevičius, G. Petraitis, A. Venčkauskas, J. Čeponis // INFORMATION TECHNOLOGY AND CONTROL. – 2013. - Vol.42, No.3. – p 32;

12. Anderson R. J. The steganographic file system [Текст] / G. H. Anderson, R. M. Needham, A. Shamir // Proceedings of the Second International Workshop on Information Hiding, - London, UK, 1998. - Vol.32, No.16 - 73–82 p.;

13. Eckstein K. Data hiding in journaling file systems [Текст] / K. Eckstein, M. Jahnke // Refereed Proceedings of the 5th Annual Digital Forensic Research Workshop, New Orleans. - Louisiana, USA. – 2005 - Vol.52, No.17. - 1–8 c.;

14. Smith G. StegFS: a steganographic file system [Текст] / G. Smith // Proceedings of the 19th International Conference on Data Engineering. - 2003. - Vol. 1, No. 8. - 657–668 p.;

15. Mao D. GBDE - GEOM Based Disk Encryption [Текст] / D. Mao // Poul-Henning Kamp. – 2016. - Vol. 9, No. 1. - 57–68 p.;

16. Anderson R. J. Stretching the Limits of Steganography [Текст] / R. J. Anderson // Proceedings published by Springer as Lecture Notes in Computer Science. – 1997. - Vol 1174, No 52. - 39–48 p.;

17. Anderson R. J. Tamper Resistance — a Cautionary Note [Текст] / R. J. Anderson, M. G. Kuhn // Proceedings of the Second Usenix Workshop on Electronic Commerce. - 2008. – Vol. 32, No 96. - 1–11 p.;

18. Designing a cluster-based covert channel to evade disk investigation and forensics [Текст] / H. Khan, M. Javed, S. A. Khayam, F. Mirza // Computers & Security. – 2011. - Vol. 30. No. 41 - 35–49 p.

ДОДАТОК А

Програмний код «SteganoFAT»

```

#include <iostream>
#include <Windows.h>
#include <ctime>
#include <conio.h>
#include <math.h>
#pragma warning (disable:4996)
using namespace std;
#pragma pack (1)
struct FLASH_INFO
{
    unsigned char start[3];
    unsigned char dos_vers[8];
    unsigned _int16 bytes_per_sector;
    unsigned _int8 sector_per_cluster;
    unsigned _int16 num_reserv_sector;
    unsigned _int8 num_of_FAT;
    unsigned _int16 num_of_dir_entries;
    unsigned _int16 LOW_total_sector;
    unsigned _int8 media_descr_type;
    unsigned _int16 num_of_sector_per_FAT_12_16;
    unsigned _int16 num_of_sector_per_track;
    unsigned _int16 num_of_heads;
    unsigned _int32 num_of_hidden_sector;
    unsigned _int32 LARGE_total_sector;

    unsigned _int32 num_of_sector_per_FAT_32;
    unsigned _int16 flags;
    unsigned _int16 FAT_version;
    unsigned _int32 cluster_num_of_root_dir;
    unsigned _int16 sector_num_FSInfo;
    unsigned _int16 sector_num_backup_boot;
    unsigned char reserved[12];
    unsigned _int8 drive_num;
    unsigned _int8 winNT_flags;
    unsigned _int8 signature;
    unsigned _int32 serial_number;
    unsigned char volum_label_string[11];
    unsigned char system_id_string[8];
    unsigned char boot_code[420];
    unsigned _int16 AA55;
};
struct FILE_INFO
{
    unsigned char name[8];
    unsigned char type[3];
    unsigned _int8 attr;
    unsigned _int8 WinNT;
    unsigned _int8 CrtTimeTenth;
    unsigned _int16 CrtTime;
    unsigned _int16 CrtDate;
    unsigned _int16 LstAccDate;
    unsigned _int16 FstClusHI;
    unsigned _int16 WrtTime;
    unsigned _int16 WrtDate;
    unsigned _int16 FstClusLO;
    unsigned _int32 FileSize;
};

```

```

HANDLE OpenDisk(char ch)
{
    char device_name[20];
    sprintf(device_name, "\\.\%c:", ch);
    return CreateFile(device_name,
        GENERIC_ALL,
        FILE_SHARE_READ,
        NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL);
}

int ReadFlasfInfo(FLASH_INFO &f_info, HANDLE flash);
void quickSort(unsigned _int32 arr[], int left, int right);
void quickSort_2(unsigned _int32* arr[], int left, int right);
FILE_INFO* ChoiceOfFiles(HANDLE fdevice, FLASH_INFO f_info, int num_files);
inline char* CorectName(char* file_name);
inline int InputFileNames(char* &file_name);
int WriteMesage(HANDLE fdevice, FLASH_INFO f_info, FILE_INFO* files, int num_files);
inline _int32 MinimumSizeFile(FILE_INFO* files, int num_files);
FILE * FileOpenToRead();
inline int SizeFile(FILE *f);
char* CreateInformArray(char* mes, int size_file, int MASK);
inline unsigned _int32 GetFirstClusterOfFile(FILE_INFO files);
unsigned _int32* GetClusterChain(HANDLE hdevice, FLASH_INFO f_info, unsigned _int32
start_cluster, unsigned _int32 num_cluster);
char ** LoadDataFilesToBuf(HANDLE hdevice, FLASH_INFO f_info, unsigned _int32**
cluster_chain, unsigned _int32* num_cluster, unsigned _int32 num_files);
unsigned _int32** ChainTransformation(unsigned _int32** elder_cluster_chain, unsigned
_int32* &start_cluster, unsigned _int32* num_cluster,
    int num_files, char* stegano_mesage, unsigned _int32 size_stegano_mesage);
unsigned _int32 LoadDataFilesFromBuf(HANDLE hdevice, FLASH_INFO f_info, unsigned _int32**
cluster_chain, unsigned _int32* num_cluster,
    unsigned _int32 num_files, char** bufer_for_file_data);
unsigned _int32 LoadClusterChainToFat(unsigned _int32** cluster_chain, HANDLE hdevice,
FLASH_INFO f_info, unsigned _int32* num_cluster, unsigned _int32 num_files);
int ReloadFileInfo(HANDLE fdevice, FLASH_INFO f_info, FILE_INFO* files, int num_files,
unsigned _int32* start_cluster);
int ReadMesage(HANDLE fdevice, FLASH_INFO f_info, FILE_INFO* files, int num_files);
FILE* FileOpenToWrite();
char* ReadSteaganoFromClusterChain(unsigned _int32** cluster_chain, unsigned _int32*
num_cluster, unsigned _int32 num_files, unsigned _int32 size_stegano_mesage);
char* CreateMesageFromStg(char* stg, int size_stg, int MASK);

int main()
{
    HANDLE fdevice;
    FLASH_INFO f_info;
    do {
        cin.clear();
        system("cls");
        cout << "Name of the valume: ";
        char ch;
        scanf("%c", &ch);
        fdevice = OpenDisk(ch);
        if ((int)fdevice == 0xFFFFFFFF)
        {
            cout << "\nInvalid name | Stop reading processes (close)\nPress 'q' to
quit\n";
            if (_getch() == 'q')
                return 0;
        }
    } while (((int)fdevice == 0xFFFFFFFF));
    ReadFlasfInfo(f_info, fdevice);
    int num_of_files;
    cout << "Entry number of cover files: ";

```



```

scanf("%i", &num_of_files);
FILE_INFO* files;
files = ChoiceOfFiles(fdevice, f_info, num_of_files);
if (files)
{
    printf("\nr - Read message from FAT.\n");
    printf("w - Write message to FAT.\n");
    printf("Make choice: ");
    char ch = _getche();
    switch (ch)
    {
        case 'r':
            ReadMessage(fdevice, f_info, files, num_of_files);
            break;
        case 'w':
            WriteMessage(fdevice, f_info, files, num_of_files);
            break;
    }
    printf("\nEND");
}
CloseHandle(fdevice);
_getch();
return 0;
}

int ReadFlashInfo(FLASH_INFO &fl_info, HANDLE flash)
{
    DWORD Bytes_READ;
    SetFilePointer(flash, 0, NULL, FILE_BEGIN);
    ReadFile(flash, &fl_info, 512, &Bytes_READ, NULL);
    SetFilePointer(flash, 0, NULL, FILE_BEGIN);
    return Bytes_READ;
}

FILE_INFO* ChoiceOfFiles(HANDLE fdevice, FLASH_INFO flash_info, int num_files)
{
    FILE_INFO* files = new FILE_INFO[num_files];
    char** file_name = new char*[num_files];
    for (int i = 0; i < num_files; i++)
        file_name[i] = new char[8];
    printf("Print file names:\n");
    while (fgetc(stdin) != '\n');
    for (int i = 0; i < num_files; i++)
    {
        printf("%i", i);
        if (InputFileNames(file_name[i]) == -1)
            return 0;
    }
    unsigned _int32 dir_sector = flash_info.num_reserv_sector + (flash_info.num_of_FAT *
flash_info.num_of_sector_per_FAT_32);
    char* buffer = new char[flash_info.bytes_per_sector*flash_info.sector_per_cluster];
    DWORD Bytes_READ;
    SetFilePointer(fdevice, dir_sector * flash_info.bytes_per_sector, NULL, FILE_BEGIN);
    ReadFile(fdevice, buffer, flash_info.bytes_per_sector*flash_info.sector_per_cluster,
&Bytes_READ, NULL);
    for (int x = 0; x < num_files; x++)
    {
        bool global_true_name = false;
        for (int i = 0; i < 16 * flash_info.sector_per_cluster; i++)
        {
            bool true_name = false;
            for (int j = 0; j < 32; j++)
                files[x].name[j] = buffer[(i * 32) + j];
            if (files[x].name[0] == 0xE5 || files[x].name[0] == 0x05)
                continue;
            if (files[x].name[11] == 0x0F)

```

```

        continue;
    for (short count = 0; count < 8; count++)
    {
        if (file_name[x][count] != files[x].name[count] &&
files[x].name[count] != 0x20)
        {
            true_name = false;
            break;
        }
        else true_name = true;
    }
    if (true_name)
    {
        global_true_name = true;
        break;
    }
}
if(global_true_name)
    printf("File information #%i was found\n", x);
else {
    files = NULL;
    printf("File name #%i invalid\n", x);
}
}
for (int i = 0; i < num_files; i++)
    delete[]file_name[i];
delete[]file_name;
delete[]buffer;
return files;
}

```

```

void quickSort(unsigned _int32 arr[], int left, int right)
{
    unsigned _int32 i = left, j = right;
    unsigned _int32 tmp;
    unsigned _int32 pivot = arr[(left + right) / 2];
    while (i <= j) {
        while (arr[i] < pivot)
            i++;
        while (arr[j] > pivot)
            j--;
        if (j == 0) j++;
        if (i <= j) {
            tmp = arr[i];
            arr[i] = arr[j];
            arr[j] = tmp;
            i++;
            j--;
        }
    };

    if (left < j)
        quickSort(arr, left, j);
    if (i < right)
        quickSort(arr, i, right);
}

```

```

void quickSort_2(unsigned _int32* arr[], int left, int right)
{
    unsigned _int32 i = left, j = right;
    unsigned _int32 tmp_1, tmp_2;
    unsigned _int32 pivot = arr[(left + right) / 2][0];
    while (i <= j) {
        while (arr[i][0] < pivot)
            i++;

```

```

        while (arr[j][0] > pivot)
            j--;
        if (j == 0) j++;
        if (i <= j) {
            tmp_1 = arr[i][0];
            tmp_2 = arr[i][1];
            arr[i][0] = arr[j][0];
            arr[i][1] = arr[j][1];
            arr[j][0] = tmp_1;
            arr[j][1] = tmp_2;
            i++;
            j--;
        }
    };
    if (left < j)
        quickSort_2(arr, left, j);
    if (i < right)
        quickSort_2(arr, i, right);
}

inline char* CorectName(char* file_name)
{
    for (int i = 0; i < 8; i++)
    {
        if (file_name[i] >= 'a' && file_name[i] <= 'z')
            file_name[i] = file_name[i] - 32;
        else if (file_name[i] <= '/')
            file_name[i] = ' ';
        else if (file_name[i] > '9' && file_name[i] < 'A')
            file_name[i] = ' ';
        else if (file_name[i] > 'Z')
            file_name[i] = ' ';
    }
    return file_name;
}

inline int InputFileNames(char* &file_name)
{
    cin.getline(file_name, 8);

    file_name = CorectName(file_name);
    if ((char* &)file_name == 0)
        return -1;
    return 0;
}

int WriteMesage(HANDLE fdevice, FLASH_INFO f_info, FILE_INFO* files, int num_files)
{
    system("cls");
    int MASK = log2(num_files);
    unsigned _int32 size_one_cluster = f_info.bytes_per_sector*f_info.sector_per_cluster;
    unsigned _int32 size_stg = 0;
    unsigned _int32 sum_num_cluster = 0;
    int time_start = 0;
    int time_end = 0;
    FILE* message_file;
    char* stg;
    unsigned _int32* num_cluster = new unsigned _int32[num_files];
    unsigned _int32* start_cluster = new unsigned _int32[num_files];
    for (int i = 0; i < num_files; i++)
    {
        num_cluster[i] = ceil((double)files[i].FileSize / size_one_cluster);
        start_cluster[i] = GetFirstClusterOfFile(files[i]);
    }
}

```

```

for (int i = 0; i < num_files; i++)
    sum_num_cluster += num_cluster[i];
printf("Max size message - %i\n", (sum_num_cluster*MASK) / 8);

bool All_ok = true;
do {
    All_ok = true;
    mesage_file = FileOpenToRead();
    while (mesage_file == 0)
    {
        printf("Incorect file name!\n ");
        mesage_file = FileOpenToRead();
    }
    int size_file = SizeFile(mesage_file);
    char * mesage = new char[size_file];
    fread(mesage, 1, size_file, mesage_file);

    size_stg = ((size_file * 8) / MASK + (bool)((size_file * 8) % MASK));
    printf("Generat steganogram..."); time_start = clock();
    stg = CreateInformArray(mesage, size_file, MASK);
    time_end = clock() - time_start;
    printf("DONE.\t\t\t%f\n", (float)time_end/CLOCKS_PER_SEC);
    delete[]mesage;
    int* stg_type = new int[num_files];
    for (int i = 0; i < num_files; i++)
        stg_type[i] = 0;
    for (int i = 0; i < size_stg; i++)
        stg_type[stg[i]]++;

    printf("\nnum stg type - num cluster file\n");
    for (int i = 0; i < num_files; i++)
    {
        if(stg_type[i]<=num_cluster[i])
            printf("%i) %i - %i\t - OK\n", i, stg_type[i], num_cluster[i]);
        else
        {
            fcloseall();
            All_ok = false;
            printf("%i) %i - %i\t - INVALID\n", i, stg_type[i],
num_cluster[i]);
        }
    }
} while (All_ok == false);
int total_write_time_start = clock() + time_end;
unsigned_int32** cluster_chain = new unsigned_int32*[num_files];
printf("\nGet cluster chain..."); time_start = clock();
for (int i = 0; i < num_files; i++)
    cluster_chain[i] = GetClusterChain(fdevice, f_info, start_cluster[i],
num_cluster[i]);
time_end = clock() - time_start;
printf("DONE.\t\t\t%f\n", (float)time_end / CLOCKS_PER_SEC);
printf("Load data file to bufer..."); time_start = clock();
char** bufer_for_file_data;
bufer_for_file_data = LoadDataFilesToBuf(fdevice, f_info, cluster_chain, num_cluster,
num_files);
time_end = clock() - time_start;
printf("DONE.\t\t\t%f\n", (float)time_end / CLOCKS_PER_SEC);
printf("Transformation cluster chain..."); time_start = clock();
cluster_chain = ChainTransformation(cluster_chain, start_cluster, num_cluster,
num_files, stg, size_stg);
time_end = clock() - time_start;
printf("DONE.\t\t\t%f\n", (float)time_end / CLOCKS_PER_SEC);
printf("Load data file to flash..."); time_start = clock();
LoadDataFilesFromBuf(fdevice, f_info, cluster_chain, num_cluster, num_files,
bufer_for_file_data);
time_end = clock() - time_start;

```

```

printf("DONE.\t\t\t%f\n", (float)time_end / CLOCKS_PER_SEC);
printf("Load cluster chain to fat..."); time_start = clock();
LoadClusterChainToFat(cluster_chain, fdevice, f_info, num_cluster, num_files);
time_end = clock() - time_start;
printf("DONE.\t\t\t%f\n", (float)time_end / CLOCKS_PER_SEC);
printf("Reoaled files metadata..."); time_start = clock();
ReloadFileInfo(fdevice, f_info, files, num_files, start_cluster);
time_end = clock() - time_start;
printf("DONE.\t\t\t%f\n", (float)time_end / CLOCKS_PER_SEC);
delete[] stg;
delete[] num_cluster;
delete[] start_cluster;
for (int i = 0; i < num_files; i++)
    delete[] cluster_chain[i];
delete[] cluster_chain;
delete bufer_for_file_data;
fcloseall();
int total_write_time_end = clock() - total_write_time_start;
printf("\nTotal time for write message - %f\n", (float)total_write_time_end /
CLOCKS_PER_SEC);
return 0;
}

inline _int32 MinimumSizeFile(FILE_INFO* files, int num_files)
{
    _int32 min = files[0].FileSize;
    for (int i = 1; i < num_files; i++)
        if (min > files[i].FileSize)
            min = files[i].FileSize;
    return min;
}

FILE* FileOpenToRead()
{
    char *str = new char[20];
    printf("\nName of message file, with type(20 symbols) : ");
    gets_s(str, 20);
    string folder = "mesage\\";
    folder += str;
    FILE* f;
    if (fopen_s(&f, folder.c_str(), "rb"))
    {
        delete[] str;
        return 0;
    }
    delete[] str;
    return f;
}

inline int SizeFile(FILE *f)
{
    fseek(f, 0, SEEK_END);
    int size = ftell(f);
    fseek(f, 0, SEEK_SET);
    return size;
}

char* CreateInformArray(char* mes, int size_file, int MASK)
{
    char* inform_array = new char[((size_file * 8) / MASK + (bool)((size_file * 8) %
MASK))];
    for (int i = 0; i < ((size_file * 8) / MASK + (bool)((size_file * 8) % MASK)); i++)
        inform_array[i] = 0;
    for (int i = 0, num_stg = 0; i < (size_file * 8); i++)
    {
        if ((i%MASK) == 0 && i != 0)
            num_stg++;
    }
}

```

```

        inform_array[num_stg] = (inform_array[num_stg] << 1) | ((mes[i / 8] >> (7 - (i
% 8))) & 0x1);
    }
    return inform_array;
}

inline unsigned _int32 GetFirstClusterOfFile(FILE_INFO files)
{
    unsigned _int32 num_cluster = 0;
    num_cluster ^= files.FstClusHI;
    num_cluster = (num_cluster << 16) ^ files.FstClusLO;
    return num_cluster;
}

unsigned _int32* GetClusterChain(HANDLE hdevice, FLASH_INFO f_info, unsigned _int32
start_cluster, unsigned _int32 num_cluster)
{
    unsigned _int32 sectors_per_fat_file = (num_cluster / 128) + 2;
    unsigned _int32* table = new unsigned _int32[128];
    unsigned _int32* chain = new unsigned _int32[num_cluster];
    unsigned _int32 chain_count = 0;
    DWORD Bytes_READ;
    for (unsigned _int32 i = 0; ; i++)
    {
        int fat_sector = (start_cluster / 128);
        SetFilePointer(hdevice, f_info.bytes_per_sector * (f_info.num_reserv_sector +
fat_sector), NULL, FILE_BEGIN);
        ReadFile(hdevice, table, f_info.bytes_per_sector, &Bytes_READ, NULL);
        for (int j = 0; j < 128; j++)
        {
            chain[chain_count] = start_cluster;
            chain_count++;
            start_cluster = table[start_cluster % 128];
            if (start_cluster == 0xFFFFFFFF)
            {
                delete[] table;
                return chain;
            }
            else if (start_cluster / 128 != (unsigned _int32)fat_sector)
                break;
        }
    }
}

char ** LoadDataFilesToBuf(HANDLE hdevice, FLASH_INFO f_info, unsigned _int32**
cluster_chain, unsigned _int32* num_cluster, unsigned _int32 num_files)
{
    unsigned _int32 dir_sector = f_info.num_reserv_sector + (f_info.num_of_FAT *
f_info.num_of_sector_per_FAT_32);
    unsigned _int32 size_one_cluster = f_info.bytes_per_sector*f_info.sector_per_cluster;
    char** bufer_for_data = new char*[num_files];
    for (int i = 0; i < num_files; i++)
        bufer_for_data[i] = new char[size_one_cluster*num_cluster[i]];
    DWORD Bytes_READ;
    for (int files = 0; files < num_files; files++)
    {
        Bytes_READ = 0;
        for (unsigned _int32 i = 0; i < num_cluster[files]; i++)
        {
            SetFilePointer(hdevice, (dir_sector*f_info.bytes_per_sector) +
(size_one_cluster* (cluster_chain[files][i] - 2)), NULL, FILE_BEGIN);
            ReadFile(hdevice, &bufer_for_data[files][size_one_cluster*i],
size_one_cluster, &Bytes_READ, NULL);
        }
        if (Bytes_READ != size_one_cluster)
        {

```

```

        delete[] bufer_for_data;
        return NULL;
    }
}
return bufer_for_data;
}

unsigned _int32** ChainTransformation(unsigned _int32** elder_cluster_chain, unsigned
_int32* &start_cluster, unsigned _int32* num_cluster, int num_files, char* stegano_mesage,
unsigned _int32 size_stegano_mesage)
{
    unsigned _int32 size_total_chain = 0;
    for (int i = 0; i < num_files; i++)
        size_total_chain += num_cluster[i];
    unsigned _int32* global_chain = new unsigned _int32[size_total_chain];

    for (int i = 0, global_position=0; i < num_files; i++)
        for (int j = 0; j < num_cluster[i]; j++, global_position++)
            global_chain[global_position] = elder_cluster_chain[i][j];
    quickSort(global_chain, 0, size_total_chain - 1);
    unsigned _int32* new_cluster_chain = new unsigned _int32[num_files];
    for (int i = 0; i < num_files; i++)
        new_cluster_chain[i] = new unsigned _int32[num_cluster[i]];
    unsigned _int32* left_clusters_number = new unsigned _int32[num_files];
    for (int i = 0; i < num_files; i++)
        left_clusters_number[i] = num_cluster[i];
    int choice = 0;
    for (unsigned _int32 i = 0; i < size_total_chain; i++)
    {
        if (i < size_stegano_mesage)
        {
            new_cluster_chain[stegano_mesage[i]][num_cluster[stegano_mesage[i]]-
left_clusters_number[stegano_mesage[i]]] = global_chain[i];
            left_clusters_number[stegano_mesage[i]]--;
        }
        else if (left_clusters_number[choice])
        {
            new_cluster_chain[choice][num_cluster[choice] -
left_clusters_number[choice]] = global_chain[i];
            left_clusters_number[choice]--;
        }
        else
        {
            choice++;
            i--;
        }
    }
    for (int i = 0; i < num_files; i++)
        start_cluster[i] = new_cluster_chain[i][0];

    for (int i = 0; i < num_files; i++)
        delete[] elder_cluster_chain[i];
    delete[] elder_cluster_chain;
    delete[] global_chain;
    delete[] left_clusters_number;
    return new_cluster_chain;
}

unsigned _int32 LoadDataFilesFromBuf(HANDLE hdevice, FLASH_INFO f_info, unsigned _int32**
cluster_chain, unsigned _int32* num_cluster, unsigned _int32 num_files, char**
bufer_for_file_data)
{
    unsigned _int32 dir_sector = f_info.num_reserv_sector + (2 *
f_info.num_of_sector_per_FAT_32);
    unsigned _int32 size_one_cluster = f_info.bytes_per_sector*f_info.sector_per_cluster;
    DWORD Bytes_Write;

```

```

    for (int files = 0; files < num_files; files++)
    {
        Bytes_Write = 0;
        for (unsigned _int32 i = 0; i < num_cluster[files]; i++)
        {
            SetFilePointer(hdevice, (dir_sector*f_info.bytes_per_sector) +
(size_one_cluster* (cluster_chain[files][i] - 2)), NULL, FILE_BEGIN);
            WriteFile(hdevice, &buffer_for_file_data[files][size_one_cluster*i],
size_one_cluster, &Bytes_Write, NULL);
        }
        if (Bytes_Write != size_one_cluster)
            return NULL;
    }
    return Bytes_Write;
}

unsigned _int32 LoadClusterChainToFat(unsigned _int32** cluster_chain, HANDLE hdevice,
FLASH_INFO f_info, unsigned _int32* num_cluster, unsigned _int32 num_files)
{
    unsigned _int32 size_one_cluster = f_info.bytes_per_sector*f_info.sector_per_cluster;
    unsigned _int32* table = new unsigned _int32[128];
    DWORD Bytes_READ;
    DWORD Bytes_WRITER;
    for (int files = 0; files < num_files; files++)
    {
        int fat_sector = (cluster_chain[files][0] / (f_info.bytes_per_sector / 4));
        Bytes_READ = 0;
        Bytes_WRITER = 0;
        SetFilePointer(hdevice, f_info.bytes_per_sector * (f_info.num_reserv_sector+
fat_sector), NULL, FILE_BEGIN);
        ReadFile(hdevice, table, f_info.bytes_per_sector, &Bytes_READ, NULL);
        for (unsigned _int32 i = 0; i < num_cluster[files]; i++)
        {
            if (cluster_chain[files][i] / (f_info.bytes_per_sector / 4) !=
fat_sector)
            {
                SetFilePointer(hdevice, (-1)*f_info.bytes_per_sector, NULL,
FILE_CURRENT);
                WriteFile(hdevice, table, Bytes_READ, &Bytes_WRITER, NULL);

                fat_sector = (cluster_chain[files][i] / (f_info.bytes_per_sector
/ 4));
                SetFilePointer(hdevice, f_info.bytes_per_sector *
(f_info.num_reserv_sector + fat_sector), NULL, FILE_BEGIN);
                ReadFile(hdevice, table, f_info.bytes_per_sector, &Bytes_READ,
NULL);
            }
            if (i + 1 == num_cluster[files])
            {
                table[cluster_chain[files][i] % 128] = 0xFFFFFFFF;
                SetFilePointer(hdevice, (-1)*f_info.bytes_per_sector, NULL,
FILE_CURRENT);
                WriteFile(hdevice, table, Bytes_READ, &Bytes_WRITER, NULL);
                break;
            }
            table[cluster_chain[files][i] % 128] = cluster_chain[files][i + 1];
        }
    }
    delete[] table;
    return Bytes_WRITER;
}

int ReloadFileInfo(HANDLE fdevice, FLASH_INFO f_info, FILE_INFO* files, int num_files,
unsigned _int32* start_cluster)
{
    unsigned _int32 dir_sector = f_info.num_reserv_sector + (2 *
f_info.num_of_sector_per_FAT_32);
    char* buffer = new char[f_info.bytes_per_sector*f_info.sector_per_cluster];

```



```

DWORD Bytes_READ;
DWORD Bytes_WRITER;
SetFilePointer(fdevice, dir_sector * f_info.bytes_per_sector, NULL, FILE_BEGIN);
ReadFile(fdevice, buffer, f_info.bytes_per_sector*f_info.sector_per_cluster,
&Bytes_READ, NULL);

for (int x = 0; x < num_files; x++)
{
    bool true_name = false;
    for (int i = 0; i < 16 * f_info.sector_per_cluster; i++)
    {
        for (short count = 0; count < 8; count++)
        {
            if (buffer[(i * 32) + count] != files[x].name[count] && buffer[(i
* 32) + count] != 0x20)
            {
                true_name = false;
                break;
            }
            else true_name = true;
        }
        if (true_name)
        {
            (unsigned _int16&)buffer[(i * 32) + 20] = ((start_cluster[x] >>
16) & 0xFFFF);
            (unsigned _int16&)buffer[(i * 32) + 26] = start_cluster[x] &
0xFFFF;
            break;
        }
    }
    SetFilePointer(fdevice, (-1)*f_info.bytes_per_sector*f_info.sector_per_cluster, NULL,
FILE_CURRENT);
    WriteFile(fdevice, buffer, Bytes_READ, &Bytes_WRITER, NULL);
    delete[]buffer;
    return Bytes_READ;
}

int ReadMesage(HANDLE fdevice, FLASH_INFO f_info, FILE_INFO* files, int num_files)
{
    system("cls");
    int MASK = log2(num_files);
    unsigned _int32 size_one_cluster = f_info.bytes_per_sector*f_info.sector_per_cluster;
    char* stg;

    int time_start = 0;
    int time_end = 0;
    int total_write_time_start = clock();

    unsigned _int32* num_cluster = new unsigned _int32[num_files];
    unsigned _int32* start_cluster = new unsigned _int32[num_files];
    for (int i = 0; i < num_files; i++)
    {
        num_cluster[i] = ceil((double)files[i].FileSize / size_one_cluster);
        start_cluster[i] = GetFirstClusterOfFile(files[i]);
    }
    unsigned _int32 all_num_cluster = 0;
    for (int i = 0; i < num_files; i++)
        all_num_cluster += num_cluster[i];
    unsigned _int32** cluster_chain = new unsigned _int32*[num_files];

    printf("Get cluster chain..."); time_start = clock();
    for (int i = 0; i < num_files; i++)
        cluster_chain[i] = GetClusterChain(fdevice, f_info, start_cluster[i],
num_cluster[i]);
    time_end = clock() - time_start;

```

```

printf("DONE.\t\t\t%f\n", (float)time_end / CLOCKS_PER_SEC);

printf("Reading steganoblocks..."); time_start = clock();
stg = ReadSteaganoFromClusterChain(cluster_chain, num_cluster, num_files,
all_num_cluster);
time_end = clock() - time_start;
printf("DONE.\t\t\t%f\n", (float)time_end / CLOCKS_PER_SEC);

printf("Conglutination message..."); time_start = clock();
char* message = CreateMesageFromStg(stg, all_num_cluster, MASK);
time_end = clock() - time_start;
printf("DONE.\t\t\t%f\n", (float)time_end / CLOCKS_PER_SEC);

int total_write_time_end = clock() - total_write_time_start;

unsigned _int32 mesage_size = ((all_num_cluster*MASK) / 8) +
((bool)(all_num_cluster*MASK) % 8);
FILE* message_file = FileOpenToWrite();
    fwrite(message, 1, mesage_size, message_file);

printf("\nTotal time for read message - %f\n", (float)total_write_time_end /
CLOCKS_PER_SEC);
    fcloseall();
    return 0;
}

FILE* FileOpenToWrite()
{
    char *str = new char[20];
    printf("\nName of message file, with type(20 symbols) : ");
    gets_s(str, 20);

    string folder = "mesage\\";
    folder += str;

    FILE* f;
    if (fopen_s(&f, folder.c_str(), "wb"))
    {
        delete[]str;
        return 0;
    }
    delete[]str;
    return f;
}

char* ReadSteaganoFromClusterChain(unsigned _int32** cluster_chain, unsigned _int32*
num_cluster, unsigned _int32 num_files, unsigned _int32 size_stegano_mesage)
{
    unsigned _int32 size_total_chain = 0;
    for (int i = 0; i < num_files; i++)
        size_total_chain += num_cluster[i];
    unsigned _int32** global_chain = new unsigned _int32*[size_total_chain];
    for (int i = 0; i < size_total_chain; i++)
        global_chain[i] = new unsigned _int32[2];

    for (int i = 0, global_position = 0; i < num_files; i++)
        for (int j = 0; j < num_cluster[i]; j++, global_position++)
        {
            global_chain[global_position][0] = cluster_chain[i][j];
            global_chain[global_position][1] = i;
        }

    quickSort_2(global_chain, 0, size_total_chain - 1);

    char* stegano_mesage = new char[size_stegano_mesage];
    for (unsigned _int32 i = 0; i < size_stegano_mesage; i++)

```

```

        stegano_mesage[i] = global_chain[i][1];

    return stegano_mesage;
}
char* CreateMesageFromStg(char* stg, int size_stg, int MASK)
{
    char* mesage = new char[((size_stg*MASK) / 8) + ((bool)(size_stg*MASK) % 8)];
    for (int i = 0; i < ((size_stg*MASK) / 8) + ((bool)(size_stg*MASK) % 8); i++)
        mesage[i] = 0;

    for (int i = 0; i < (size_stg*MASK); i++)
        mesage[i / 8] = (mesage[i / 8] << 1) | ((stg[i / MASK] >> ((MASK - 1) -
(i%MASK)))) & 0x1);
    return mesage;
}

```

ДОДАТОК Б
Наукова діяльність

Міністерство освіти і науки України
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
РАДІОЕЛЕКТРОНІКИ

МАТЕРІАЛИ 20-го ЮВІЛЕЙНОГО МІЖНАРОДНОГО
МОЛОДІЖНОГО ФОРУМУ

«РАДІОЕЛЕКТРОНІКА І МОЛОДЬ В ХХІ СТОЛІТТІ»

19 – 21 квітня 2016 р.

Том 5

МІЖНАРОДНА КОНФЕРЕНЦІЯ
«ВІРТУАЛЬНИЙ ТА ФІЗИЧНИЙ КОМП'ЮТІНГ»

Харків 2016

20-й Ювілейний Міжнародний молодіжний форум «Радіoeлектроніка і молодь в ХХІ столітті». Зб. матеріалів форуму. Т. 5. – Харків: ХНУРЕ. 2016. – 236 с.

В збірник включені матеріали 20-го Ювілейного Міжнародного молодіжного форуму «Радіoeлектроніка і молодь в ХХІ столітті».

Видання підготовлено
факультетом комп'ютерної інженерії та управління
Харківського національного університету радіoeлектроніки (ХНУРЕ)

61166 Україна, Харків, просп. Науки, 14
тел.: (057) 7021397
факс: (057) 7021515

E-mail: innov@kture.kharkov.ua

© Харківський
національний університет
радіoeлектроніки (ХНУРЕ), 2016

МЕТОДЫ АНАЛИЗА ЭФФЕКТИВНОСТИ МЕТОДОВ ТЕХНИЧЕСКОЙ СТЕГАНОГРАФИИ

Шеханин К.Ю.

Научный руководитель – д.т.н., проф. Халимов Г.З.

Харьковский национальный университет радиоэлектроники
(61166, Харьков, пр. Науки, 14, каф. Безопасности информационных технологий, тел. (057) 702-13-06)
e-mail: kyryl.shekhanin@nure.ua

This work is devoted to the modern developments in the field of analysis of the effectiveness of technical-steganography techniques. Namely, the analysis method of data hiding by mixing the clusters in the FAT32 file system. To analyze the effectiveness we consider indicator data file fragmentation on the carrier. The this indicator is more, the less stable this steganographic system to determinacy hidden information. Conversely, the lower this indicator, the more effective this system. This method allows the analysis to objectively evaluate the effectiveness of the security system of the steganographic possibility of determining the presence of hidden information.

Стеганография – быстро и динамично развивающаяся наука, использующая методы и достижения криптографии, цифровой обработки сигналов, теории связи и информации. Целью данной области является скрытая передача данных по каналу связи. Причем в отличии от криптографических методов, где информация шифруется и приобретает критерий конфиденциальности, стеганография скрывает сам факт передачи информации. Таким образом, основным критерием эффективности стеганосистемы является защищённость от детерминирования наличия сокрытой информации.

Техническая стеганография – новый раздел современной цифровой стеганографии, изучающий методы и средства встраивания и извлечения информационных сообщений в различные цифровые контейнеры с использованием технических особенностей представления (форматирования), хранения, передачи и отображения контейнеров.

Каналом передачи в стеганосистеме подобного рода может выступать файловый носитель. В случае использования метода сокрытия данных в файловую систему FAT32 используется особенность формата хранения данных. Незанятый контейнер располагает кластеры с данными соответствующего файла один за одним (по возможности), а запись информации перемешивает кластеры соответствующих файлом в ключевом порядке, так что бы при чтении возможно было однозначно считать скрываемую информацию. Все эти действия повышают уровень фрагментации данных на файловом носителе.

Высокий уровень фрагментации является значимым демаскирующим признаком для данной стеганосистемы.

В данной работе предлагается использовать анализ уровня фрагментации данных на файловом носителе для определения с заданной вероятностью наличия стеганоканала.

Имя файла	A.txt	A.txt	B.txt	C.txt	FREE	D.txt	B.txt	BAD	B.txt	...
№ кластера	1	2	3	4	5	6	7	8	9	...

Рисунок 1 – «Пример FAT таблицы»

Для оценки уровня фрагментации каждому файлу необходимо присвоить весовой коэффициент по заданной формуле.

Формула 1 – «Расчет весового коэффициента уровня фрагментации, для одного файла»

$$K_{\alpha} = \sum_{\alpha=0}^{\alpha=end-1} El_{(\alpha,\alpha+1)}; \quad El = \frac{n(n+1)}{2},$$

n – количество кластеров между α и $\alpha + 1$ кластеров одного файла;

α – имя файла;

end – метка конца файла.

Посчитав весовые коэффициенты для каждого файла сравниваем с максимальным допустимым уровнем. Те файлы, чьи весовые коэффициенты больше граничного уровня, являются покрывающими файлами – ключом, для извлечения информации из стеганоконтейнера.

Для оценки общего уровня фрагментации необходимо полученные весовые коэффициенты просуммировать и сравнить с допустимым уровнем.

Данный метод позволяет оценить вероятность наличия сокрытой информации в стеганоконтейнере сокрытой путем использования методов технической стеганографии, а именно сокрытие данных путем перемешивания кластеров в файловой системе FAT32.

Список источников:

1. Конахович Г. Ф., Пузыренко А. Ю. – «Компьютерная стеганография. Теория и практика.», 2006. — 288 с.
2. Грибунин В, Оков И, Туринцев И – «Цифровая стеганография», Litres, 2015.
3. Хорошко В.О., Азаров О.Д., Шелест М.Є., Яремчук Ю.Є. – «Основи комп'ютерної стеганографії : Навчальний посібник для студентів і аспірантів.» — Вінниця: ВДТУ, 2003. — 143 с.

Ф

Федоренко К.И. 224

Х

Халимов О.Г. 103

Хомицкий И.А. 226

Ц

Цяпа О.В. 228

Ч

Чепижко С.С. 167

Чуприна А.А. 48

Ш

Шведко А.Г. 169

Шевченко А.Р. 190

Шеханин К.Ю. 93

Ю

Юхневич П.И. 171

Я

Яковенко Д.А. 173

Янко В.И. 230

Ястребов И. С. 153

Яценко О.П. 56,58

[illegible]