

Харківський національний університет імені В. Н. Каразіна

Міністерство освіти і науки України

Кваліфікаційна наукова
праця на правах рукопису

Шеханін Кирил Юрійович

УДК 004.056.5

Дисертація

«РОЗРОБКА ТА АНАЛІЗ СТЕГANOГРАФІЧНИХ МЕТОДІВ ПРИХОВУВАННЯ
ДАНИХ В СТРУКТУРУ ФАЙЛОВИХ СИСТЕМ»

Спеціальність 125 «Кібербезпека»


(12 «Інформаційні технології»)

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей
результатів і текстів інших авторів мають посилання на відповідне джерело.

 К.Ю. Шеханін
(ініціали та прізвище здобувача)

Науковий керівник: Кузнецов Олександр Олександрович, доктор технічних
наук, професор.

*Усі примірники дисертації ідентичні
за змістом
такою стилізованою вкритою
ДФ 64.051.071*  *Василь НАЗАРУК*

Харків – 2021

АНОТАЦІЯ

Шеханін К.Ю. Розробка та аналіз стеганографічних методів приховування даних в структуру файлових систем. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 125 «Кібербезпека» (Галузь знань 12 – «Інформаційні технології»). – Харківський національний університет імені В. Н. Каразіна Міністерство освіти і науки України, Харків, 2021.

Дисертація присвячена розробці та удосконаленню стеганографічних методів приховування інформації у структуру файлової системи шляхом перемішування кластерів. Розробці способів оцінки зазначених методів та можливого способу використання.

Метою дисертаційної роботи є підвищення пропускну здатності кластерних стеганосистем при забезпеченні необхідної стійкості до несанкціонованого детектування прихованої інформації.

У першому розділі дисертації (*Дослідження технологій носіїв інформації та властивостей файлових систем*) виконано аналіз актуального стану розвитку технологій фізичних носіїв інформації. Зокрема, проаналізовані такі технології як HDD та SSD. Надано прогноз щодо розвитку технологій носіїв інформації. Також проаналізовані розповсюджені файлові системи, виконано їх порівняльний аналіз. За результатами аналізу обрано файлову систему для подальшого дослідження. Вирішена *перша часткова задача дослідження*: дослідження сучасних і перспективних методів зберігання інформації, властивостей фізичних носіїв та типів файлових систем, аналіз існуючих методів стеганографічного приховування у структурі файлових систем.

У другому розділі (*Розробка методу підвищення пропускної здатності кластерних стеганосистем та дослідження властивостей*) детально проаналізована файлова система FAT32, як еталона система із сімейства кластерних файлових систем. Також досліджені та проаналізовані властивості структури файлової системи що сприяють приховування повідомлення у структуру файлової системи. Надано математичну модель методу приховування інформації шляхом перемішування кластерів покриваючих файлів. Вирішено *другу часткову задачу дослідження*: розробка методу підвищення пропускної здатності кластерних стеганосистем. *Вперше отримано* метод підвищення пропускної здатності кластерних стеганосистем на основі урахування додаткової залежності місць розміщення кластерів у межах одного покриваючого файлу системи.

У третьому розділі (*Аналіз методів приховування інформації та удосконалення математичної моделі оцінки основних параметрів кластерних стеганосистем*) проаналізовано досліджувані методи на можливий розмір приховуваного повідомлення у залежності від різних вихідних параметрів. Отримано формули, за якими можливо оцінити максимально можливий розмір стеганограми у залежності від ключових параметрів (кількість, порядок та розмір покриваючих файлів). Наведено графіки, які наочно демонструють залежність розміру стеганограми від ключових параметрів методів.

Також оцінено рівень захищеності прихованого повідомлення до детектування, шляхом аналізу середнього рівня фрагментації файлової системи та фрагментації кожного покриваючого файлу. Результати оцінки отримано шляхом статистичного аналізу рівня фрагментації комп'ютерних систем із лабораторій Харківського національного університету імені В. Н. Каразіна. За результатами статистичного аналізу надано зліпок файлових систем з точки зору рівня фрагментації, найбільш фрагментованих типів файлів та способу використання комп'ютерних систем. За результатами даного аналізу надано дані щодо можливого розміру приховуваного

повідомлення у такий спосіб, щоб рівень фрагментації покриваючих файлів був у межах середнього рівня файлової системи.

У даному розділі була надана оцінка часових параметрів методів приховування повідомлення, виконана за допомогою розробленої програми «SteganoFAT». Дана оцінка залежить від технічних властивостей комп'ютерної системи та фізичного носія інформації, отже було виведено рівняння загальної обчислювальної складності у залежності від кількості покриваючих файлів та кількості стеганоблоків. Загальний час на приховування залежить від кількості переміщень зчитуваючої головки, кількості зчитувань та записів даних у кластери файлів. Результатом даного дослідження стали методи приховування інформації із послідовним та почерговим записом даних до оперативної пам'яті для базового методу (ПЗОП, ПчЗОП-I/II/III) та відповідні методи для модифікованого методу приховування інформації (ПЗОПм, ПчЗОПм-I/II/III). Для кожного методу теоретично отримано оцінку обчислювальної складності. Найоптимальнішим за часовими показниками та рівнем фрагментації є розроблений у даній дисертаційній роботі метод ПчЗОП-II (почергове завантаження даних із кластерів до оперативної пам'яті із послідовним впорядкування лише задіяних у повідомленні кластерів).

Виконано *третю часткову задачу дослідження*: удосконалено математичну модель оцінки основних параметрів кластерних стеганосистем за рахунок додаткового урахування елементів конфігурації стеганосистеми, що дозволяє більш повно оцінити пропускну здатність системи.

Також вирішено *четверту часткову задачу дослідження*: удосконалення методу приховування інформації у структурі кластерних стеганосистем. Удосконалення полягає у раціональному використуванні оперативної пам'яті. Також інше удосконалення полягає у спеціальному розрахунку таблиць перестановок таким чином, щоб зменшити кількість переміщуваних кластерів. Це дозволило значно зменшити час на приховування повідомлення.

Отримано *другий науково-обґрунтований результат*: удосконалено математичну модель оцінки основних параметрів кластерних стеганосистем за рахунок додаткового урахування елементів конфігурації стеганосистеми, що дозволяє більш повно оцінити пропускну здатність системи.

Та отримано *третій науково-обґрунтований результат*: удосконалено метод приховування інформації у структуру кластерних стеганосистем за рахунок генерації відповідного набору перестановок кластерів, що дозволяє зменшити час приховування інформації.

У четвертому розділі (*Розробка програмної реалізації методів приховування інформації*) розроблено програмну симуляцію, що наочно дозволяє продемонструвати принцип роботи методів приховування інформації у структуру файлової системи шляхом перемішування кластерів покриваючих файлів та емпірично оцінити ефективність способів приховування (ПЗОП, ПчЗОП-I/II/III). Надано повний опис розробленої програмної реалізації із посиланням на відкритий репозиторій, описано алгоритмічні особливості методів приховування інформації та надано фрагменти коду функціоналу приховування повідомлення.

За допомогою розробленої програми було отримано емпіричну оцінку обчислювальної складності стосовно кожного методу приховування інформації, кожного способу. За результатами порівняння теоретично отриманої та емпірично розрахованої оцінки обчислювальної складності виявлено, що способи модифікованого методу у даній програмній реалізації мають гіршу обчислювальну складність ніж теоретично очікувалось. Та було описано вдосконалений алгоритм роботи модифікованого методу, який дозволяє зменшити рівень обчислювальної складності.

Була виконана *п'ята часткова задача дослідження*: розробка програмної реалізації запропонованих методів та проведення експериментальних досліджень.

У п'ятому розділі (*Напрямки використання методів приховування інформації*) описані можливі системи що використовують методи приховування інформації у структуру файлової системи. Такими системами є:

- система прихованого збереження даних;
- система прихованої передачі даних;
- система верифікації фізичного носія інформації чи файлів, що збережено на носії.

По кожній системі надано опис із аналізом ефективності такої системи у залежності від конфігураційних параметрів.

Також надано опис розповсюджених програмних реалізації що імплементують описані вище системи використання методів приховування інформації. Зроблено порівняльний аналіз та надано оцінку відомих програмних реалізацій як би вони також використовували розроблені у даній роботі методи приховування інформації, як компонент у свої алгоритмах. Надано висновок, що описані методи дозволяють значно розширити функціонал реалізованих програм та/або методи можуть надати альтернативу, у разі коли внутрішні інструменти програм використовувати недоречно.

Ключові слова: метод приховування інформації, стеганографічні методи, файлова система, носій інформації, FAT, статистичний аналіз, структура файлової системи, пропускна здатність, обчислювальна складність, програма симуляція, StarForce.

ABSTRACT

Kyryl Yu. Shekhanin. Development and analysis of steganographic methods of hiding data in the structure of file systems. – Qualification scientific work is as a manuscript.

Thesis submitted for obtaining the Doctor of Philosophy degree in Information Technologies, Speciality 125 – Cybersecurity. – V. N. Karazin Kharkiv National University, Ministry of Education and Science of Ukraine, Kharkiv, 2021.

The dissertation is devoted to the development and improvement of steganographic methods of hiding information in the structure of the file system by mixing clusters. Development of a model for evaluating these methods and possible uses.

The purpose of the dissertation is to increase the bandwidth of cluster steganosystems while ensuring the necessary resistance to unauthorized detection of hidden information.

The first chapter of the dissertation (*Research of media technologies and properties of file systems*) analyzes the current state of development of technologies of physical media. In particular, technologies such as HDD and SSD are analyzed. The forecast on development of technologies of data carriers is given. Common file systems are also analyzed, their comparative analysis is performed. Based on the results of the analysis, a file system is selected for further research. *The first partial research problem* is solved: conducted research on modern and promising methods of information storage, properties of physical media and types of file systems, analysis of existing methods of steganographic hiding information in the structure of file systems.

The second chapter (*Development of a method to increase the bandwidth of cluster steganosystems and study their properties*) analysis in detail the FAT32 file system as a root system from the family of cluster file systems. Possible properties of the file system structure that help to hide the message in the file system structure are also investigated and analyzed. A mathematical model of the method of hiding information by mixing clusters of cover files is given. *The second partial problem of the research* is solved: development of a method to

increase the bandwidth of cluster steganosystems. *For the first time is obtained* a method of increasing the bandwidth of cluster steganosystems on the basis of taking into account the additional dependence of cluster locations within single cover file of the system.

In the third chapter (*Analysis of methods of hiding information and improving the mathematical model for estimating the basic parameters of cluster steganosystems*) the researched methods on the possible size of the hidden message depending on various initial parameters are analyzed. The formulas by which it is possible to estimate the maximum possible size of the steganogram depending on key parameters of methods are received (number, order and size of cover files). Diagrams are given that clearly show the dependence of the size of the steganogram on the key parameters of the methods.

The level of security of the hidden message before detection is also assessed by analyzing the average level of file system fragmentation and fragmentation of each cover file. The evaluation results are obtained by statistical analysis of the level of fragmentation of computer systems from laboratories of V. N. Karazin Kharkiv National University. According to the results of statistical analysis, a cast of file systems in terms of the level of fragmentation, the most fragmented file types and how to use computer systems. Based on the results of this analysis, data on the possible size of the hidden message are provided, so that the level of fragmentation of the cover files is within the average level of the file system.

This chapter provides an estimate of the time parameters of the methods of hiding the message, performed using the developed program “SteganoFAT”. This estimate depends on the technical properties of the computer system and the physical storage medium, so the equation of the total computational complexity is derived depending on the number of cover files and the number of steganoblocks. The total hiding time depends on the number of read head movements, the number of reads and writes data in clusters. The result of this study are methods of hiding information: ПЗОП, ПЧЗОП-I/II/III for the basic method and ПЗОПМ, ПЧЗОП-I/II/IIIM for a modified method of hiding information, respectively. For each method, an estimate of the computational complexity is theoretically obtained. The most optimal in terms of time and level of fragmentation is the method developed in this

dissertation ПЧЗОП-II (alternately reading data from clusters to RAM with sequential ordering of only the clusters involved in the message).

The third partial task of the research is fulfilled: improvement of the mathematical model of estimation of the basic parameters of steganosystems. This is achieved by a comprehensive assessment of bandwidth, assessment of resistance to detection and assessment of computational complexity.

The fourth partial problem of the research is also solved: improvement of the method of hiding information in the structure of cluster steganosystems. The improvement is the rational use of RAM, which has reduced the amount of memory required, which has allowed the implementation of hiding methods on low-resource systems. Another improvement is the special calculation of permutation tables so as to reduce the number of moving clusters. This significantly reduced the time to hide the message.

The second scientifically substantiated result is obtained: the mathematical model of estimation of the basic parameters of cluster steganosystems due to additional consideration of elements of a configuration of a steganosystem that allows to estimate more completely capacity of steganosystem capacity is improved.

And *the third scientifically substantiated result* is obtained: improved method of hiding information in the structure of cluster steganosystems by generating an appropriate set of permutations of clusters, which reduces the time of hiding information.

In the fourth chapter (*Development of software implementation of methods of hiding information*), a software simulation is developed, which clearly demonstrates the principle of methods of hiding information in the structure of the file system by mixing clusters of cover files and empirically evaluate the effectiveness of hiding methods (ПЗОП, ПЧЗОП-I/II/III). A full description of the developed software implementation is provided with reference to the public repository, algorithmic features of methods of hiding information are described and fragments of code of functionality of hiding of the message are given.

With the help of the developed program the empirical estimation of computational complexity concerning each method of hiding of the information, each way is received.

According to the results of comparison of theoretically obtained and empirically calculated estimation of computational complexity, it is found that the methods of the modified method in this software implementation have worse computational complexity than theoretically expected. However, an improved algorithm of the modified method is described, which allows to reduce the level of computational complexity.

The fifth partial task of the study is performed: development of software implementation of the proposed methods and conducting experimental research.

The fifth chapter (*Directions for using methods of hiding information*) describes possible systems that use methods to hide information in the file system structure.

Such systems are:

- system of hidden data storage;
- system of hidden data transmission;
- a system for verifying the physical medium of information or files stored on the medium.

For each system, a description is provided with an analysis of the effectiveness of such a system depending on the configuration parameters.

Also provided is a description of common software implementations that implement the above-described systems of methods of hiding information. A comparative analysis is made and an assessment of known software implementations is given as if they also used the methods of hiding information developed in this paper as a component in their algorithms. It is concluded that the described methods allow to significantly expand the functionality of the implemented programs and / or methods can provide an alternative in case it is unable to use internal program tools.

Key words: method of hiding information, steganographic methods, file system, storage device, FAT, statistical analysis, file system structure, bandwidth, computational complexity, simulation software, StarForce.

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Публікації у періодичних наукових закордонних виданнях, проіндексованих у базах даних Web of Science Core Collection та/або Scopus:

1. Shekhanin K., Kuznetsov A., Krasnobayev V., Smirnov O. Detecting hidden information in FAT. *International Journal of Computer Network and Information Security*. 2020. Vol. 12, Iss. 3, P. 33 – 43 (Hong Kong, Scopus).

DOI: 10.5815/ijcnis.2020.03.04

URL: <https://www.scopus.com/record/display.uri?eid=2-s2.0-85086029655&origin=resultslist>;

(Особистий внесок здобувача: дослідження статистичних властивостей розподілу файлів за рівнем фрагментації, дослідження стійкості до детектування методів приховування інформації).

2. Shekhanin K.Yu., Kolhatin A.O., Demenko E.E., Kuznetsov A. A. On Hiding Data Into the Structure of the FAT Family File System. *Telecommunications and Radio Engineering*. 2019. Vol. 78, Iss. 11, P. 973 – 985 (United States, Scopus).

DOI: 10.1615/TelecomRadEng.v78.i11.50;

URL <https://www.scopus.com/record/display.uri?eid=2-s2.0-85070406462&origin=resultslist>;

(Особистий внесок здобувача: розробка та аналіз методу підвищення пропускної здатності у кластерних стеганосистемах).

Публікації у виданнях, включених до переліку фахових видань України з присвоєнням категорії «Б»:

3. Шеханін К.Ю., Пшенична С.В., Кузнецов О.О. Дослідження обчислювальної складності методів приховування інформації у кластерні стеганосистеми. *Радіоелектроніка*. 2021. Вип. 206. С. 77 – 87.

URL: <http://rt.nure.ua/issue/archive>;

(Особистий внесок здобувача: розробка удосконалених методів приховування інформації).

4. Шеханін, К., Горбенко, Ю., Горбачова, Л., Кузнецов, О. Дослідження властивостей носіїв інформації для стеганографічного приховування даних в кластерних файлових системах. *Радіотехніка*. 2020. Вип. 203. С. 109 – 120.

DOI: 10.30837/rt.2020.4.203.10

URL: <http://rt.nure.ua/article/view/229343>;

(Особистий внесок здобувача: дослідження технологій фізичних носіїв інформації).

Публікації у виданнях, включених до переліку фахових видань України

5. Кузнецов О., Тимченко В., Лисицький К., Родінко М., Луценко М., Шеханін К., Колгатін А.. Дослідження швидкодії та статистичної безпеки алгоритмів криптографічного гешування. *Радіотехніка*. 2019. Вип. 198. С. 75 – 95.

DOI: 10.30837/rt.2019.3.198.06;

URL: <http://rt.nure.ua/article/view/184661>;

(Особистий внесок здобувача: аналіз швидкодії та обчислювальної складності алгоритмів гешування).

6. Шеханін К., Колгатін А., Деменко Є., Кузнецов О. Приховування даних у структуру файлової системи сімейства FAT. *Радіотехніка*. 2018. Вип. 193. С. 169 – 178.

URL: <http://rt.nure.ua/article/view/175804>;

(Особистий внесок здобувача: опис та порівняльний аналіз існуючих стеганографічних методів у структурі кластерних файлових систем).

Публікації, які засвідчують апробацію матеріалів дисертації:

7. Steganographic hiding information in a file system structure / K. Shekhanin, A. Kolhatin, K. Kuznetsova, S Kavun // 2018 International Conference on Information and Telecommunication Technologies and Radio Electronics : Proceeding of UkrMiCo 2018, 10–14 September 2018, Odessa. (Scopus).

DOI: 10.1109/UkrMiCo43733.2018.9047551

URL: <https://www.scopus.com/record/display.uri?eid=2-s2.0-85083488842&origin=resultslist>;

(Особистий внесок здобувача: опис методу підвищення пропускну́ї здатності методів приховування у кластерні файлові системи).

8. Hiding data in the structure of the FAT family file system / Kuznetsov A., Shekhanin K., Kolhatin A. [etc.] // Dependable Systems, Services and Technologies (DESSERT) : conference proceedings of 2018 IEEE 9th International Conference, 24 – 27 May 2018, Kyiv, 2018 / Kyiv, 2018. – P. 337 – 342. (Scopus).

DOI: 10.1109/DESSERT.2018.8409155;

URL: <https://www.scopus.com/record/display.uri?eid=2-s2.0-85050684345&origin=resultslist>;

(Особистий внесок здобувача: порівняльний аналіз методів приховування інформації у структуру файлових систем FAT).

Публікації, які додатково відображають наукові результати дисертації:

Публікації у виданнях України:

9. Kuznetsov A., Shekhanin K., Kolgatin A., Kuznetsova K., Demenko Y. Hiding data in the file structure. *Комп'ютерні науки та кібербезпека*. 2018. № 1 (9). С. 43 – 52.

URL: <https://periodicals.karazin.ua/cscs/article/view/12013>

(Особистий внесок здобувача: описані та досліджені методи приховування інформації у структуру файлових систем).

патенти:

10. Спосіб стеганографічного приховування інформації за допомогою технологій 3d-друку: пат. 131986, Україна: № u201808262; заявл. 26.07.2018; опубл. 11.02.2019.

(Особистий внесок здобувача: дослідження стеганографічних властивостей заявленого методу).

11. Спосіб стеганографічного приховування даних в кластерних файлових системах: пат. 132302, Україна: № u201808261; заявл. 26.07.2018; опубл. 25.02.2019.

(Особистий внесок здобувача: розробка способу стеганографічного приховування даних в кластерних файлових системах).

розділ у колективній монографії:

12. Detecting Hidden Information in FAT: collective monograph / Kuznetsov A., Shekhanin K., Smirnov O., Chepurko I. – *ISCI'2019: Information Security in Critical Infrastructures* : collective monograph. USA. 2019. P. 412 – 429.

DOI 10.17605/OSF.IO/E8PY6;

URL: <https://osf.io/u3bt4>;

(Особистий внесок здобувача: розробка методу оцінки пропускної здатності та стійкості до детектування методів приховування)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	18
ВСТУП.....	20
РОЗДІЛ 1 ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ НОСІІВ ІНФОРМАЦІЇ ТА ВЛАСТИВОСТЕЙ ФАЙЛОВИХ СИСТЕМ.....	26
1.1 Дослідження технологій носіїв інформації	26
1.2 Дослідження файлових систем	32
1.3 Аналіз властивостей та параметрів файлової системи FAT32	33
1.4 Аналіз властивостей та параметрів файлової системи NTFS.....	37
1.5 Аналіз властивостей та параметрів файлової системи exFAT	39
1.6 Теоретична та практична оцінка файлових систем FAT32, NTFS, exFAT ..	40
Висновки до розділу 1	43
РОЗДІЛ 2 РОЗРОБКА МЕТОДУ ПІДВИЩЕННЯ ПРОПУСКНОЇ ЗДАТНОСТІ КЛАСТЕРНИХ СТЕГАНОСИСТЕМ ТА ДОСЛІДЖЕННЯ ВЛАСТИВОСТЕЙ	44
2.1 Опис властивостей FAT32, що можуть сприяти приховуванню інформації	44
2.2 Математична модель методу приховування даних у структуру файлової системи	48
2.3 Математична модель модифікованого методу приховування даних у структуру файлової системи	55
Висновки до розділу 2	64
РОЗДІЛ 3 АНАЛІЗ МЕТОДІВ ПРИХОВУВАННЯ ІНФОРМАЦІЇ ТА УДОСКОНАЛЕННЯ МАТЕМАТИЧНОЇ МОДЕЛІ ОЦІНКИ ОСНОВНИХ ПАРАМЕТРІВ КЛАСТЕРНИХ СТЕГАНОСИСТЕМ	66
3.1 Аналіз пропускної здатності методів приховування даних шляхом переміщування кластерів.....	66

3.2 Аналіз захищеності методів приховування даних шляхом перемішування кластерів до детектування	75
3.3 Аналіз часових параметрів та обчислювальної складності методу приховування даних у структуру файлової системи FAT	111
3.4 Математична модель оцінки основних параметрів кластерних стеганосистем	117
3.5 Удосконалення методів приховування інформації у структуру кластерних файлових систем.....	119
Висновки до розділу 3	142
РОЗДІЛ 4 РОЗРОБКА ПРОГРАМНОЇ РЕАЛІЗАЦІЇ МЕТОДІВ ПРИХОВУВАННЯ ІНФОРМАЦІЇ.....	144
4.1 Опис програмної реалізації	146
4.2 Розробка програмної реалізації	151
4.2.1 Модуль збереження інформації.....	151
4.2.2 Модуль обробки інформації	153
4.3 Емпірична оцінка обчислювальної складності методів приховування інформації.....	158
Висновки до розділу 4	170
РОЗДІЛ 5 НАПРЯМКИ ВИКОРИСТАННЯ МЕТОДІВ ПРИХОВУВАННЯ ІНФОРМАЦІЇ.....	172
5.1 Система прихованого збереження даних	172
5.2 Система прихованої передачі даних	175
5.3 Система верифікації фізичних носіїв та файлів.....	178
5.4 Опис та дослідження компонентів програмного комплексу StarForce	181
Висновки до розділу 5	184
ВИСНОВКИ.....	186
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	189
ДОДАТОК А. СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІІ	200

ДОДАТОК Б. ОПИС КЛАСУ ФАЙЛОВОЇ СИСТЕМИ	204
ДОДАТОК В. ОПИС КЛАСУ ФАЙЛУ	206
ДОДАТОК Г. ОПИС КЛАСУ КЛАСТЕРУ	208
ДОДАТОК Ґ. ОПИС КЛАСУ СТАТИСТИКИ.....	209
ДОДАТОК Д. ФУНКЦІЇ ОБРОБКИ ПОВІДОМЛЕННЯ У СТЕГАНОВЛОКИ.....	213
ДОДАТОК Е. ФУНКЦІЇ РОБОТИ ІЗ ПЕРЕСТАНОВКОЮ.....	215
ДОДАТОК Є. ФУНКЦІЇ БАЗОВОГО МЕТОДУ ПРИХОВУВАННЯ	217
ДОДАТОК Ж. ФУНКЦІЇ МОДИФІКОВАНОГО МЕТОДУ ПРИХОВУВАННЯ.....	221
ДОДАТОК З. АКТИ ВПРОВАДЖЕННЯ	226

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API	–	Application programming interface
Avg	–	Average
CD	–	Compact disc
CPU	–	Central processing unit
CSS	–	Cascading style sheets
DVD	–	Digital versatile disc
FAT	–	File allocation table
FS	–	File system
GPU	–	Graphics processing unit
HDD	–	Hard (magnetic) disk drive
HTML	–	Hypertext markup language
JSON	–	JavaScript object notation
Max	–	Maximum
MFT	–	Master file table
Min	–	Minimum
NTFS	–	New technology file system
RAM	–	Random access memory
SD	–	Secure digital (memory card)
SSD	–	Solid-state drive
UI	–	User interface
USB	–	Universal serial bus
USN	–	Update sequence number
K3	–	Контрольована зона

КПФ	–	Кількість покриваючих файлів
НСД	–	Несанкціонований доступ
ПЗОП	–	Повне завантаження кластерів до оперативної із впорядкованим розміщенням залишку кластерів
ПК	–	Персональний комп'ютер
ПчЗОП-I	–	Почергове завантаження кластерів до оперативної пам'яті із впорядкованим розміщенням залишку кластерів
ПчЗОП-II	–	Почергове завантаження кластерів до оперативної пам'яті із переміщенням лише інформативних кластерів у нормальну послідовність
ПчЗОП-III	–	Почергове завантаження кластерів до оперативної пам'яті із оптимальним переміщенням лише інформативних кластерів
РК	–	Розмір кластеру
СВФН	–	Система верифікації фізичного носія
СДЛПЗ	–	Система дистрибуції ліцензійного програмного забезпечення
СДФН	–	Система дистрибуції фізичних носії
СЕД	–	Система електронного документообігу
ФС	–	Файлова система

ВСТУП

Обґрунтування вибору теми дослідження. На сьогоднішній день інформацію розглядають як один з основних ресурсів для розвитку сучасного суспільства, а інформаційні системи та технології, як засоби підвищення ефективності та продуктивності роботи сучасних систем [1].

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 [2, 3, 4].

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 [3].

На даний час відомо декілька методів технічної стеганографії. Приховування інформації у модель під час 3D-друку [5, 6, 7, 8], дана галузь приховування інформації має певні переваги та недоліки, а саме: відносно більшу кошовність при створенні прихованого повідомлення, та складності при зчитуванні інформації [9, 10, 11, 12]. Другий напрямок технічної стеганографії пов'язаний із мережевим трафіком [13, 14, 15]. У даному методі інформація може приховуватись, наприклад, у поля заголовків протоколів, чи, наприклад, передача прихованого повідомлення шляхом посилення певної послідовності пакетів [16]. Також існують методи приховування інформації у структуру файлової системи але відомі методи або здатні приховати малу кількість інформації, або мають належний рівень стійкості до детектування [17, 18]. Таким чином, актуальною задачею є розробка методу приховування інформації, яка здатна

приховати більшу кількість інформації та має більший рівень стійкості до детектування [19, 20].

У даній роботі представлено метод технічної стеганографії, що базується на структурній особливості файлових систем у носіях інформації. А саме, приховування інформації у файловій системі FAT шляхом перемішування кластерів певних, ключових, файлів. Аналіз даного алгоритму є актуальним, бо сучасні технології збереження інформації нівелюють обмеження які накладає даний метод приховування даних, та на даний час існує незначна кількість алгоритмів, що використовують структуру файлової системи як спосіб приховування інформації.

Зв'язок роботи з науковими програмами, планами, темами. Дисертаційні дослідження проводились в рамках науково-дослідних робіт: № 1-41-18, «Аналіз, дослідження, розробка та стандартизація криптографічних систем для захисту інформації в пост-квантовому середовищі, в умовах інформаційних і гібридних війн»(акт впровадження від 21.08.2021).

Мета і завдання дослідження. Метою дисертаційної роботи є підвищення пропускної здатності кластерних стеганосистем при забезпеченні необхідної стійкості до несанкціонованого детектування прихованої інформації.

Для досягнення мети дослідження необхідно вирішити **науково-прикладну задачу**: розробка методів підвищення пропускної здатності та приховування даних в структуру кластерних файлових систем.

Часткові задачі дослідження:

1. Провести дослідження сучасних і перспективних методів зберігання інформації, властивостей фізичних носіїв та типів файлових систем, аналіз існуючих методів стеганографічного приховування у структурі файлових систем.
2. Розробка методу підвищення пропускної здатності кластерних стеганосистем.
3. Удосконалення математичної моделі оцінки основних параметрів стеганосистем.

4. Удосконалення методу приховування інформації у структурі кластерних стеганосистем.

5. Розробка програмної реалізації запропонованих методів та проведення експериментальних досліджень.

Об’єкт дослідження – процеси обробки даних у кластерних стеганосистемах.

Предмет дослідження – моделі та методи підвищення пропускної здатності та приховування інформації в кластерних стеганосистемах.

Методи дослідження. При проведенні дисертаційних досліджень використовувалися методи теорії ймовірностей та математичної статистики, методи теорії захисту інформації та стеганографії, математичного та імітаційного моделювання.

Методи теорії ймовірностей та математичної статистики використовувалися при вирішенні задачі розробки загального методу оцінки ефективності пропускної здатності, стійкості до детектування методів приховування інформації у структуру файлової системи.

Методи теорії захисту інформації та стеганографії використовувалися при розробці методу підвищення пропускної здатності кластерних стеганосистем, розробці різних варіацій виконання методів приховування інформації, розробці програмної реалізації.

Методи математичного та імітаційного моделювання використовувалися при проведенні експериментальних досліджень з метою перевірки теоретичних положень, висновків та рекомендацій, обґрунтування достовірності отриманих результатів.

Наукова новизна отриманих результатів. В дисертаційній роботі отримано теоретичне узагальнення та нове вирішення науково-прикладної задачі з розробки методів та обчислювальних алгоритмів приховування та вилучення інформації у структурі файлової системи, оцінки пропускної здатності утворюваних стеганоканалів та швидкодії стеганоперетворень.

Отримано такі нові **науково-обґрунтовані результати:**

1. *вперше* отримано метод підвищення пропускну́ї здатності кластерних стеганосистем на основі урахування додаткової залежності місць розміщення кластерів у межах одного покриваючого файлу системи, що дозволяє майже вдвічі збільшити обсяг прихованих даних;

2. *удосконалено* математичну модель оцінки основних параметрів кластерних стеганосистем за рахунок додаткового урахування елементів конфігурації стеганосистеми, що дозволяє оцінити пропуску́ здатність запропонованої системи;

3. *удосконалено* метод приховування інформації у структуру кластерних стеганосистем за рахунок генерації відповідного набору перестановок кластерів, що дозволяє зменшити час приховування інформації;

Запропоновані удосконалення дозволяють оптимізувати програмну реалізацію алгоритмів приховування/вилучення інформації у різних практичних застосуваннях, зокрема, для малоресурсних обчислювальних систем із обмеженнями оперативної пам'яті (перше удосконалення) та систем із обмеженнями на кількість можливих перезаписів окремих кластерів (друге удосконалення).

Практичне значення отриманих результатів.

1. Розроблено спеціальне програмно-математичне забезпечення, яке практично реалізує запропоновані методи приховування інформації у структуру файлової системи, та яке дозволяє експериментально досліджувати показники пропускну́ї здатності, стійкості до несанкціонованого детектування, швидкодії (кількості циклів перезапису) та обчислювальних витрат пам'яті.

2. Отримано емпіричні залежності різних показників ефективності стеганографічного перетворення (пропускну́ї здатності, стійкості до несанкціонованого детектування, швидкодії (кількості циклів перезапису) та обчислювальних витрат пам'яті). Експериментально встановлено та доведено, що:

— запропонований метод дозволяє збільшити майже у два рази пропуску́ спроможність утворюваних стеганоканалів;

- удосконалений метод дозволяє зменшити обчислювальні витрати оперативної пам'яті з лінійної залежності (від кількості кластерів покривельних файлів) до константного значення;

- удосконалений метод дозволяє зменшити кількість циклів перезапису кластерів файлової системи на фізичному носії даних від 10% до 10 разів (в залежності від вихідних співвідношень розмірів інформаційних повідомлень, розмірів кластерів та кількості покривельних файлів);

3. Розроблено практичні рекомендації щодо впровадження розроблених методів.

4. Теоретичні та практичні результати дисертаційних досліджень апробовані у статтях що входять до наукометричної бази SCOPUS, у патенті на корисну модель №132302 та у розділі монографії. Також отримані результати застосовуються у навчальному процесі Харківського національного університету імені В. Н. Каразіна (акт впровадження від 12.08.2021).

Особистий внесок здобувача. Дисертаційна робота є самостійно виконаною науковою працею, в якій викладено методи приховування інформації у структуру файлової системи та способи їх оцінки. Усі наукові результати, викладені в дисертаційній роботі, одержані автором особисто і відображені у наукових публікаціях, що становлять індивідуальний внесок автора. З наукових праць, виданих у співавторстві, у роботі використані лише ті положення, що становлять індивідуальний внесок автора. Коректний внесок здобувача в цих роботах зазначений у списку наукових публікацій, опублікованих за темою.

Апробація матеріалів дисертації. Основні результати дослідження дисертаційної роботи представлені, доповідались та обговорювались на: **3 міжнародних наукових, науково-практичних та науково-технічних конференціях, проведених в Україні:** IX-й Міжнародній конференції «Dependable Systems, Services and Technologies (DESSERT)» (м. Київ, 2018 р.); XX-й Ювілейній Міжнародній науково-практичній конференції «Безпека інформації у інформаційно-

телекомунікаційних системах» (м. Буча, 2018 р.); International Conference on Advanced Trends in Information Theory (ATIT) (м. Київ, 2019 р.); III-й IEEE International Conference on Information and Telecommunication Technologies and Radio Electronics (UkrMiCo'2018) (м. Одеса, 2018 р.);

Публікації. Основні результати дисертаційних досліджень опубліковані у 12 наукових працях, серед яких: 2 статті у періодичному науковому виданні, що входить до міжнародної наукометричної бази (Scopus), 4 статей у наукових фахових виданнях України, 2 матеріалів та тез доповідей на конференціях, 1 стаття, що додатково висвітлює результати дисертації, 2 патенти України, 1 розділ монографії, опублікований у співавторстві.

Структура та обсяг дисертації. Дисертаційна робота складається зі вступу, 5 розділів, висновків, списку використаних джерел та 10 додатків. Обсяг загального тексту дисертації складає 229 сторінку, з них 168 сторінок основного тексту. Робота ілюстрована 51 рисунком та 46 таблицями. Список використаних джерел містить 100 найменування.

РОЗДІЛ 1

ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ НОСІЇВ ІНФОРМАЦІЇ ТА ВЛАСТИВОСТЕЙ ФАЙЛОВИХ СИСТЕМ

Параметри та ефективність методів приховування інформації у структуру файлових системи у значній мірі залежить від фізичних властивостей носія інформації та від особливостей обраної файлової системи. Тож для подальшої роботи необхідно оцінити як і різні за технологією носії інформації так і різні файлові системи.

1.1 Дослідження технологій носіїв інформації

З початку розвитку інформаційних технологій людство вигадало безліч носіїв інформації більшість з яких на разі є архаїчними. До та таких носіїв інформації можна віднести:

- перфоровані картки – носій інформації, призначений для використання в ранніх (до початку 1980-х років) системах автоматизованої обробки даних. Виготовлялась з цупкого паперу;
- гнучкий магнітний диск (*floppy disk*) – портативний носій інформації, який використовується для багаторазового запису та зберігання даних, що являє собою поміщений в захисний пластиковий корпус гнучкий магнітний диск, покритий феромагнітним шаром;
- оптичний диск (*compact disc, CD*) – переносний оптичний диск для збереження інформації у цифровому вигляді, тобто формату зберігання даних. Цей формат було спочатку розроблено для записування та відтворення лише звукозаписів, але пізніше, його було пристосовано для зберігання даних: (*CD-ROM*). Пізніше, були додатково отримані кілька інших форматів дисків, у тому числі: одноразового запису аудіо та збереження даних (*CD-R*), перезаписуваний носій (*CD-RW*), відео компакт-

диск (VCD), супер-відео компакт-диск (SVCD), Photo CD, Picture CD, Enhanced Music CD. Аудіо CD та аудіо програвачі компакт-дисків, стали комерційно доступними, починаючи з жовтня 1982 року.

Таблиця 1.1

Порівняння основних характеристик SSD та HDD пристроїв

Архітектура накопичувача	SSD	HDD
Рівень шуму	майже відсутній	Значний, так як є рухомі частини
Механічна стійкість	висока	низька, при падінні рухомі частини можуть бути пошкодженні
Енергоспоживання	2-3 Ватт/годину	5-6 Ватт/годину
Магнітна чутливість	майже відсутня	Значна, так як електромагнітне поле безпосередньо впливає на магнітний диск
Розміри	менші розміри та вага	більші розміри через присутність рухомих частин
Паралельні операції	присутні, що значно пришвидшує запис/зчитування декількох файлів	відсутні
Швидкість запису/зчитування (Мб/с)	1000-3200/2000-4000	100-320/200-400
Ціна (\$/Гб)	0.5	0.1
Циклів перезапису (разів)	10000	>100000
Вплив фрагментації на роботу	швидкість доступу знижується до 10%	швидкість доступу знижується до 2%

Але на теперішній час ринок носіїв інформації заповнили інші технології які використовують сучасні способи збереження інформації, а саме:

1. SSD (*Solid-State Drive*) – це енергонезалежний немеханічний запам'ятовуючий пристрій, що базується на основі мікросхем пам'яті та контролері;
2. HDD (*Hard (magnetic) Disk Drive*) – це енергонезалежний запам'ятовуючий пристрій оснований на основі магнітного запису на жорсткий диск;
3. Flash-USB (*Universal Serial Bus*) – це енергонезалежний запам'ятовуючий пристрій на основі мікросхем, має подібну структуру до SSD накопичувачів, але має

менші показники ємкості та швидкодії, тому здебільшого використовуються для перенесення інформації з пристрою на пристрій, та відокремленого збереження незначної кількості важливої інформації (паролі, ключи доступу, документи).

Порівнюючи переваги та недоліки SSD та HDD накопичувачів [21] можна стверджувати, що SSD мають більше переваг ніж недоліків, зведена інформація наведена у таблиці 1.1. Дані таблиці мають неточний характер через те що кожен виробник має свої характеристики стосовно фізичних накопичувачів інформації SSD та HDD. Та одна й та ж технологія навіть у одного виробника може мати різні показники у залежності від цінової категорії пристрою. Головною метою було показати загальну тенденцію та різницю між фізичними накопичувачами інформації SSD та HDD [22].

Але з розвитком технологій можна зазначити що SSD накопичувачі у порівнянні з аналогами п'ятирічної давнини мають значний приріст у показниках. У той час як показники HDD пристроїв майже не змінились.

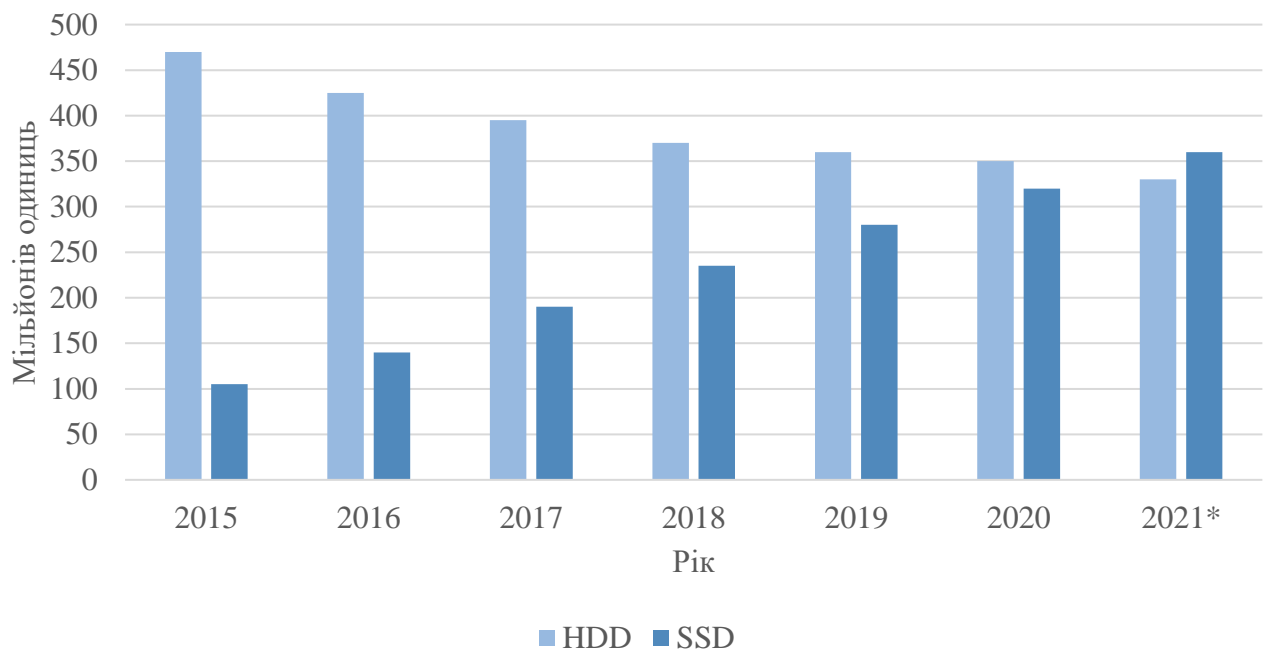


Рис. 1.1 Реалізація випущеної продукції накопичувачів інформації по всьому світі (у мільйонах одиниць)

Якщо ще у 2015 році кількість HDD пристроїв значно переважала на ринку накопичувачів, то вже у 2020 кількість пристроїв HDD та SSD порівнялась. Зведена гістограма наведена на рисунку 1.1.

Таким чином можна стверджувати, що з кожним роком кількість SSD накопичувачів росте, забираючи долю ринка у HDD накопичувачів. Динаміка кількості HDD та SSD накопичувачів зазначена на рисунку 1.2 [22].

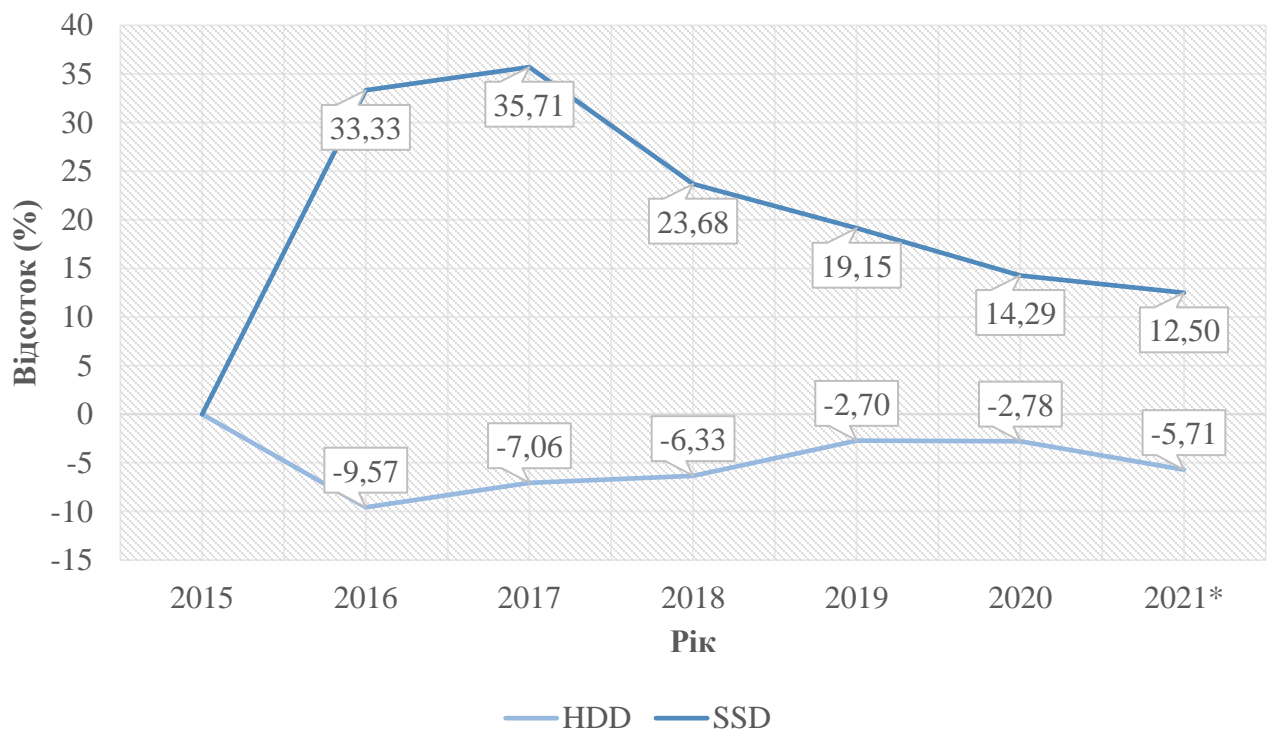


Рис. 1.2 Динаміка кількості HDD та SDD пристроїв

Для впливу технології накопичувача на можливості методу приховування інформації у структуру файлових систем [17, 18, 19] більш детально оцінимо швидкодію кожної технології. Так як єдиної специфікації по SSD, HDD технологіям не існує а швидкодія кожного пристрою залежить від виробника та цінової категорії тож порівняльна оцінка зроблена спираючись на результати електронного ресурсу *UserBenchmark* (<https://www.userbenchmark.com>). Даний ресурс збирає дані по

компонентам комп'ютера (CPU, GPU, SSD, HDD, RAM, USB) проводячи тестування «opensource» програмним забезпеченням. Дані з UserBenchmark є об'єктивними так як на ресурсі вказані результати більш ніж 160 мільйонів користувачів.

У даній роботі нас цікавлять лише SSD та HDD накопичувачі, так як для методу приховування інформації [19] важливим є параметр швидкості запису/зчитування з носія інформації, для цього виберемо найкращі пристрої за цими показниками. Для SSD накопичувачів це – Gigabyte GP-ASM2NE6100TTTD Aorus NVMe PCIe M.2 1TB (190\$). Для HDD накопичувачів це – WD WD6001FZWX Black 6TB (225\$).

Тестування даних пристрої проводилось методами:

1. Послідовного запису (*Sequential*) – це модель доступу до диску, при якій великі блоки даних записуються у сусідні блоки на поверхні пристрою з глибиною черги, рівній одиниці. Даний термін використовується здебільшого у контексті порівняльного аналізу, швидкість вимірюють у МБ/с. Даний тип доступу часто використовують при зчитуванні/записі великих за розміром файлів, таких як відео, музика та зображення. Як демонструє практика близько 50% звичайного доступу користувача до диску на ПК буде складатись з послідовних операцій зчитування/запису. Накопичувачі, що в цілому використовуються для великих мультимедійних файлів та/або резервних копій повинні мати відносно велику швидкість послідовного доступу до файлів;

2. Випадковий запис 4K (*Random 4k*) – це шаблон доступу до диску, при якому невеликі (4k) блоки даних записуються у випадкові місця на поверхні пристрою, що тестується, з глибиною черги рівній одиниці. Даний термін використовується здебільшого у контексті порівняльного аналізу, швидкість вимірюють у МБ/с. Даний метод оцінки швидкодії можна використовувати як те наскільки ефективно пристрій зчитує/записує невеликі фрагменти даних із випадкових місць. Даний шаблон значно розповсюджений під час запуску операційної системи, коли з накопичувача необхідно зчитати велику кількість файлів конфігурації та драйверів. Як демонструє практика,

близько 20% звичайного доступу користувача до диску на ПК буде складатись з запису/зчитування з випадкових блоків накопичувача.

Результат тестування *Random 4k* є більш важливим для методу приховування інформації шляхом перемішування кластерів у структурі файлової системи так як при приховуванні даних запис виконується у «випадкові» блоки накопичувача у залежності від приховуваної інформації. Зведені результати тестувань пристроїв наведені у таблиці 1.2.

Таблиця 1.2

**Зведені результати швидкодії SSD та HDD пристроїв за результатами
UserBenchmark**

T	SSD (МБ/с)						HDD (МБ/с)					
Mt	Sequential			Random 4k			Sequential			Random 4k		
	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.
R	809	1950	2318	33	49.5	58.5	102	167	215	1.67	4.74	7.2
W	1705	3115	3783	103	176	221	138	202	246	2.23	3.13	3.5
M	704	1998	2315	50.7	78.2	92.3	67.5	102	215	0.7	1.02	1.1

де:

- T – технологія пристрою;
- Mt – метод оцінювання;
- R (read) – швидкість зчитування даних;
- W (write) – швидкість запису даних;
- M (mixed) – швидкість почергового зчитування/запису даних.

Таким чином якщо порівнювати швидкості накопичувачів, то безпосередньо SSD є більш швидким, у абсолютних значеннях. Якщо порівнювати спад швидкості Random 4k та Sequential, то SSD при фрагментованому записі має швидкість у

5-10% у той час як HDD втрачають свою швидкість до 1.5-3%. Звісно 5-10% це мало, але значно краще ніж у HDD [22].

Отже роблячи висновок можна стверджувати що технологія SSD має значні переваги, завдяки чому накопичувачі з даною технологією стають більш розповсюдженими. SSD має у рази більшу швидкість доступу до файлів/секторів, навіть при значному показнику фрагментованості – це сприятливий показник для використання стеганографічних методів приховування даних. Також SSD має обмежену кількість циклів перезапису, і хоча це й недолік, але для методу приховування даних цей показник сприятливий, адже виконання дефрагментації є небажаною операцією для SSD.

Таким чином завдяки розповсюдженню SSD метод приховування інформації у структуру файлових систем шляхом перемішування кластерів покриваючих файлів є актуальним.

1.2 Дослідження 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 [23, 24].

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 дисертаційній роботі реалізовано та проаналізовано стеганографічний метод приховування інформації у структуру файлової системи накопичувачів, тому розглянуто та проаналізовано такі файлові системи: FAT32, NTFS, exFAT. Так як ці файлові системи є найпоширенішими файловими системами, що використовуються у накопичувачах [24, 25, 26].

1.3 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 [23, 25, 27].

У файловій системі FAT суміжні сектори 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 1.3.

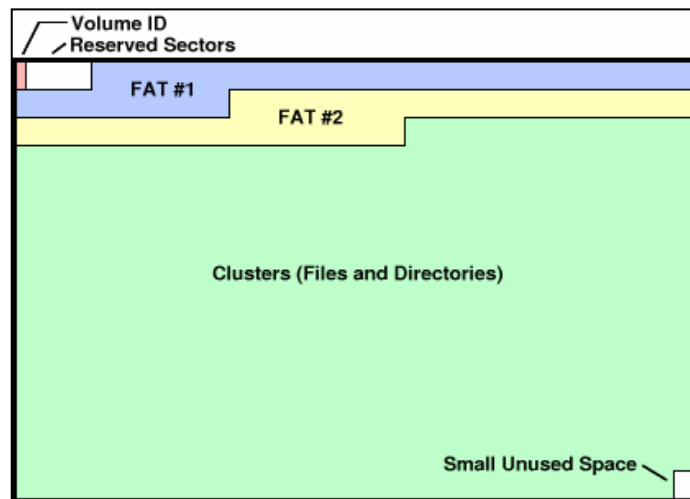


Рис.1.3 Логічні області структури FAT

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 [28, 29, 30].

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 1.4.

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

Рис. 1.4 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 кореневому 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 кореневому каталозі має фіксований розмір у 32 байти, та містить метадані файлу як це зазначено на рисунку 1.5.

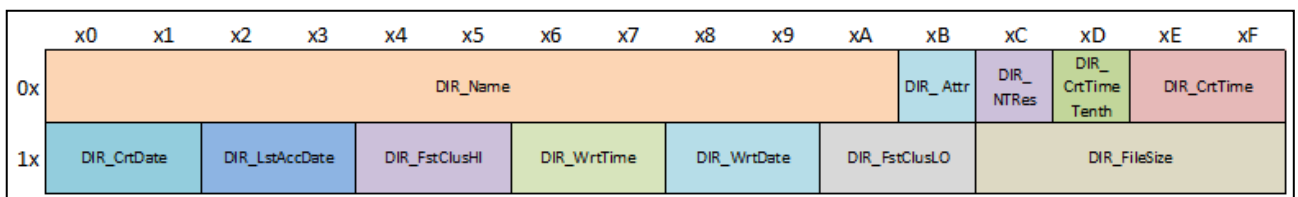


Рис. 1.5 Структура записаних метаданих файлу у каталозі

Метадані, які зображені на рисунку 1.5 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) [29, 30].

1.4 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 [24, 31].

В певній області тому знаходиться основна системна структура NTFS – майстер таблиця. У записах цієї таблиці міститься уся інформація про розміщення фалів 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 одноманітно обробляти як з файлами користувачів, так і з системними. Приклад структури зображено на рисунку 1.6.

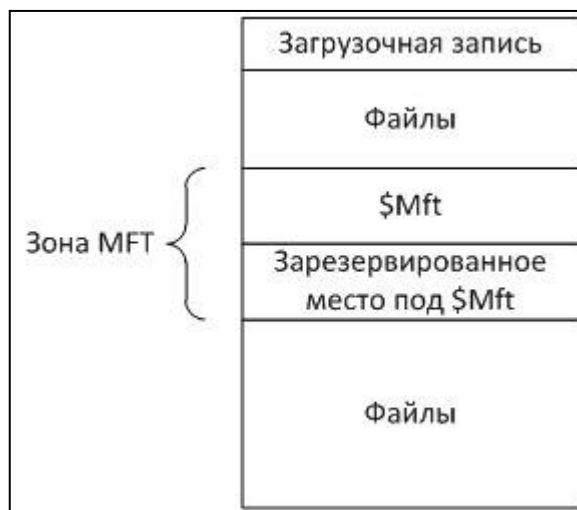


Рис. 1.6 Загальна структура файлової системи NTFS

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 [24, 31, 32].

1.5 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, тобто така файлова система є пропріетарною, що унеможлиблює модифікацію чи дослідження файлової системи. А також іншим

недоліком є, відносно складна структура, що потребує більше розрахункових ресурсів для роботи з файлами, а саме зчитуванню та запису [26, 27].

1.6 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 1.3.

Таблиця 1.3

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 1.3 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 носіях інформації.

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 [23].

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 1.4.

Роблячи висновки із таблиці 1.4, 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 [24, 25, 26, 31].

Таблиця 1.4

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 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 накопичувачах. Також, надано їх порівняльну оцінку, що є

вихідними даними для обрання типу файлової системи для розробки стеганографічного методу приховування інформації.

Висновки до розділу 1

У даному розділі проаналізовано технології носіїв інформації, а саме: SSD, HDD, Flash-USB.

Та проаналізовано файлові системи: FAT, NTFS, exFAT. Файлову систему FAT, для подальшого дослідження, обрано через те що, FAT є повністю відкритою файловою системою, FAT не має надлишкового функціоналу (шифрування, обмеження доступу і так далі), що дозволяє повністю сконцентруватися на меті дисертаційної роботи. Також файлова система FAT-32 є базисною по відношенню до інших кластерних файлових систем, що означає що дослідження проведені над FAT-32 будуть й актуальні і по відношенню до інших, більш сучасних, кластерних файлових систем. Аналіз проводився використовуючи Flash-USB носії інформації так як це є найпоширенішими носіями інформації, та ця технологія носія інформації є більш ефективно ніж HDD носії. Таким чином використовуючи Flash-USB носії можна стверджувати, що результати мають усереднені результати [32, 33].

Таким чином була виконана перша часткова задача дослідження, а саме було проведено дослідження сучасних і перспективних методів зберігання інформації, властивостей фізичних носіїв та типів файлових систем.

Результати досліджень даного розділу наведено в публікації здобувача: [34].

РОЗДІЛ 2

РОЗРОБКА МЕТОДУ ПІДВИЩЕННЯ ПРОПУСКНОЇ ЗДАТНОСТІ КЛАСТЕРНИХ СТЕГАНОСИСТЕМ ТА ДОСЛІДЖЕННЯ ВЛАСТИВОСТЕЙ

Для розробки та дослідження властивостей алгоритму приховування та вилучення інформації у кластери файлової системи було обрано файлову систему FAT32, так як дана файлова система є однією з найпоширеніших файлових систем, та досить простою за структурою для розробки та аналізу стеганографічних методів.

У даній роботі було знайдено та описано властивості файлової системи FAT32 які потенційно можуть бути використані для стеганографічного приховування інформації.

2.1 Опис властивостей FAT32, що можуть сприяти приховуванню інформації

Загалом у файловій системі для стеганографічного приховування можна використовувати такі властивості [27]:

- 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
		000001A00	48 65 6C 6C 6F 20 57 6F 72 6C 64 00 00 00 00 00	Hello World
Snapshot taken	33 min. ago	000001A10	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
Physical sector No.:	269	000001A20	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
Logical sector No.:	13	000001A30	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

Рис. 2.1 Приклад приховування інформації у зарезервований сектор

Сутність наступного методу полягає у способі індикації показчиків вільних кластерів. Для стандартного API показчиком вільного кластера є той у якого молодші 28-м біт обнулено, тобто $0xF0000000$ та $0x10000000$ – є показчиками вільних кластерів. А отже старші 4 біти можуть бути використані для приховування інформації. Основними недоліками даного методу 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 методу 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	0F	FF	FF	FF	0F
FAT 1	Cluster 5: free	000304010	FF	FF	FF	0F	00	00	00	F0	00	00	00	10	00	00	00

Рис. 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 [35, 36]. 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	53	74	6E	67	61	6E	6F	67	Steganog
FAT 1	reserved	001181F10	72	61	70	68	79	20	69	20	74	68	65	20	70	72	61	raphy is the pra
		001181F20	63	74	69	63	65	20	6F	66	20	63	6F	6E	63	65	61	ctice of conceal
Snapshot taken	32 min. ago	001181F30	69	6E	67	20	61	20	66	69	6C	65	2C	20	6D	65	73	ing a file, mess
Physical sector No.:	36111	001181F40	61	67	65	2C	20	69	6D	61	67	65	2C	20	6F	72	20	age, image, or v
Logical sector No.:	35855	001181F50	69	64	65	6F	20	77	69	74	68	69	6E	20	61	6E	6F	ideo within anot

Рис. 2.3 Приклад приховування інформації у залишок сектора

Четвертий метод реалізується за рахунок перестановки кластерів певних покриваючих файлів, таким чином, щоб спільну послідовність показників 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 [18, 37, 38]. 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

Рис. 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 [39, 40]. 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) І – метод приховування інформації у зарезервовані сектора перед першою FAT таблицею;
- б) ІІ – метод приховування інформації в старші 4 біти у покажчиках вільних кластерів;
- в) ІІІ – метод приховування інформації залишок останнього сектора FAT таблиці;
- г) ІV – метод приховування інформації шляхом перемішування кластерів у файловій системі;
- д) Ф – форматування;
- е) С – створення нового файлу;
- ж) В – видалення певного файлу.

2.2 Математична модель методу приховування даних у структуру файлової системи

Перші методи технічної стеганографії, які засновані на приховуванні повідомлення у структуру файлової системи, розглянуто у роботах [17, 39]. Найпростіші методи застосовують вільні кластери (або службові поля даних) для запису прихованої інформації, але такий спосіб не є надійним [41, 42, 43]. Інші методи застосовують надмірність, яка виникає штучно, у способі нумерації застосованих кластерів. Змінюючи нумерацію окремих кластерів певних файлів (їх називають покриваючими файлами) вдається приховати невелику кількість інформаційних бітів [17]. Розглянемо найбільш ефективний спосіб такого приховування з метою його подальшого розвитку та вдосконалення.

Для базового методу [17] застосуємо наступні позначення. Нехай повідомлення M , яке буде вбудовано, позначається через масив $M = [b_0, b_1, \dots, b_{n-1}]$, де n – довжина повідомлення (кількість стеганоблоків), b_i – окремий стеганоблок (набір з m бітів),

$i = 0, 1, \dots, n - 1$. Покрівельні файли позначимо як F_0, F_1, \dots, F_{p-1} , де p – кількість покрівельних файлів. Кількість покрівельних файлів строго підпорядковується обмеженню та може бути лише ступінь двійки, тобто $p = 2^m$, де $m \in N$, тобто належить множині натуральних чисел. Натуральне число m є однією з ключових інформацій необхідних для однозначного приховування та вилучення інформації. Параметр m безпосередньо впливає на розмір кожного із стеганоблоків b_i , $b_{ilen} = m$.

Іменами (назвами) покрівельних файлів F_i , є строки t_i , де $i = 0, 1, \dots, p - 1$. Задана послідовність покрівельних файлів F_i , $i = 0, 1, \dots, p - 1$, (або їх назв t_i , $i = 0, 1, \dots, p - 1$) також є ключовою інформацією. Тобто ключом для однозначного вилучення та приховування даних є кількість та порядок покрівельних файлів.

Для приховування даних необхідно визначити стеганоконтейнер та стеганоблок. Для цього зазначимо множину усіх кластерів файлової системи FAT – $C = [0, 1, \dots, FAT_{len}]$, FAT_{len} – довжина файлової системи у штук кластерів (довжина кожного кластеру залежить від налаштування файлової системи та може бути 4-64 КБ). Кожен покрівельний файл F_i розміщений у наборі кластерів – ланцюгу кластерів $C_{F_i} = [\{j | 0 \leq j \leq FAT_{len}\}]$, $0 < C_{F_i len} < FAT_{len}$. Тобто ланцюг кластерів покрівельного файлу F_i складається з кластерів що входять до множини усіх кластерів файлової системи. Та довжина ланцюга кластерів може бути від 1 кластера до усіх кластерів файлової системи. Кожен кластер може бути представлений як $c_{i,j}^{F_i}$, де i – порядок кластера у множині кластерів файлової системи C , j – порядок кластера у множині кластерів певного покрівельного файлу C_{F_i} . Ми визначили необхідні вихідні параметри: повідомлення яке необхідно приховати, кількість та набір покрівельних файлів із відповідними ланцюгами кластерів.

Для приховування повідомлення необхідно виконати наступну послідовність дій. Повідомлення M , необхідно розбити на масив стеганоблоків по m біт кожен, $M = [b_0, b_1, \dots, b_{n-1}]$. Якщо останній стеганоблок не має достатньої довжини то його необхідно доповнити нульовими або одиничними бітами (у залежності від реалізації

методу). Кожен стеганоблок b_i необхідно інтерпретувати у натуральне число, як зазначено у формулі 1. Перетворення стеганоблоку у натуральне число рекомендовано виконувати шляхом перетворення двійкового коду у десятинне число

$$b_i \in N, i = 0, 1, \dots, n - 1; 0 \leq b_i \leq p - 1 \quad (2.1)$$

Кожен блок b_i відповідає кластеру покрівельного файлу F_i , тобто $F_i, i = b_i$. Тобто повідомлення M , можна представити у вигляді набору кластерів покрівельних файлів у відповідній послідовності, формула 2.2.

$$M = [F_{i_0}, F_{i_1}, \dots, F_{i_{n-1}}] \quad (2.2)$$

Також файлова система вже має певну послідовність кластерів покриваючих файлів у множині кластерів файлової системи, формула 2.3

$$M^* = [c_{0j}, c_{1j}, \dots, c_{ij}, \dots, c_{FATLenj}] \quad (2.3)$$

Таким чином для приховування повідомлення M , необхідно теперішній стан кластерів файлової системи M^* перетворити на необхідний стан – $M = [F_{i_0}, F_{i_1}, \dots, F_{i_{n-1}}]$. Це робиться шляхом перестановки кластерів покрівельних файлів – $\pi(c_{ij}^*) = c_{ij}$. Схематично процес приховування повідомлення зображено на рисунку 2.5

Як приклад методу приховування інформації у структуру файлової системи FAT шляхом перемішування кластерів покриваючих файлів задамо вихідні параметри, а саме:

- M – повідомлення яке необхідно приховати. $M = [0, 1, 1, 0, 1, 1, 0, 0]$.
- p, F – кількість та набір покрівельних файлів. $p = 4$, $F_0 = [4, 5, 6, 7]$, $F_1 = [11, 12, 16]$, $F_2 = [17, 18]$, $F_3 = [22, 23, 24]$.

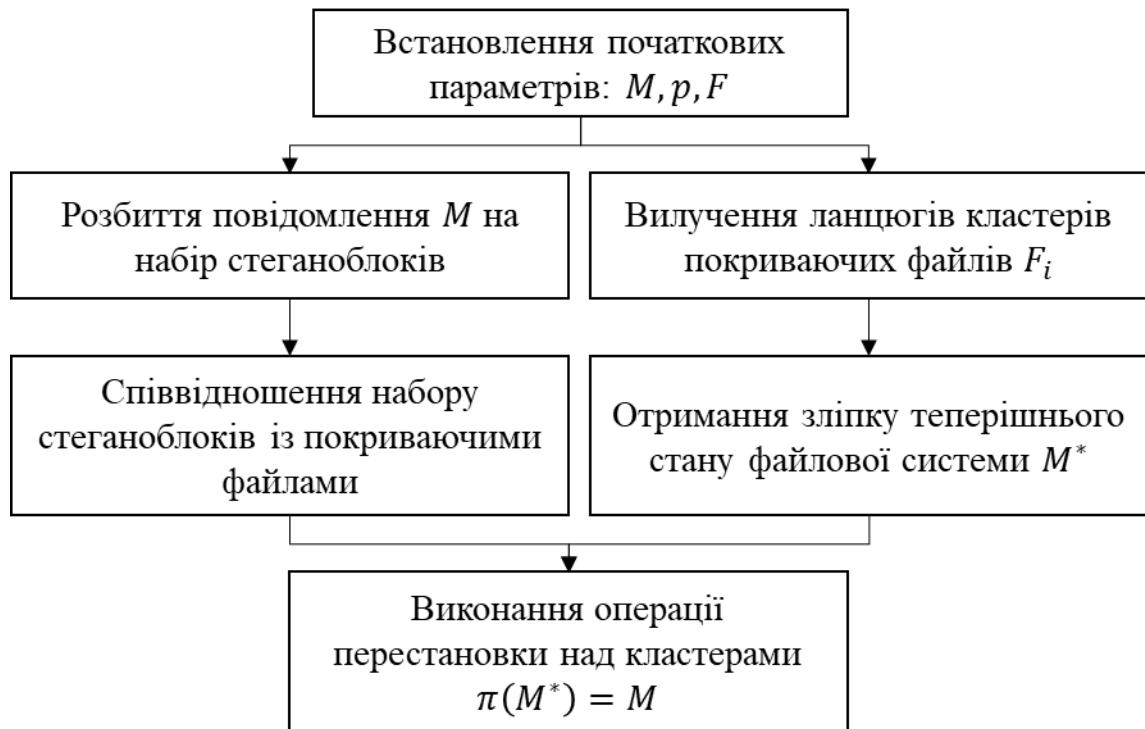


Рис. 2.5 Схематичне зображення методу приховування інформації у структуру файлової системи.

Отримавши вихідні параметри наступне що необхідно зробити, це розбити повідомлення на стеганоблоки b_i розміром по $b_{ilen} = t$ біт кожен, де $p = 2^m$. Тобто $4 = 2^m \rightarrow m = 2$. А значить повідомлення $M = [0,1,1,0,1,1,0,0]$ матиме таку розбивку по блоках – $M = [01,10,11,00]$.

Наступне що необхідно зробити – співвідносити кожен варіант стеганоблоку b_i із кластером покриваючого файлу. Рекомендовано виконати таке співвідношення шляхом переведення двійкового значення стеганоблоку у десятинне значення, що й відповідає порядковому номеру покриваючого файлу із набору покриваючих файлів F . Результат співвідношення стеганоблоку та покриваючого файлу вказано у таблиці 2.2.

Отже повідомлення M з точки зору кластерів покриваючих файлів матиме такий вигляд: $M = [0,1,1,0,1,1,0,0] \rightarrow M = [01,10,11,00] \rightarrow M = [F_1, F_2, F_3, F_0]$. Тобто, щоб

вважати що повідомлення M приховано, необхідно розмістити кластери покриваючих файлів саме у зазначеній вище послідовності.

Таблиця 2.2

Співвідношення стеганоблоків повідомлення M , із пориваючими файлами F

Вихідні параметри	$M = [0,1,1,0,1,1,0,0], p = 4, F = [F_0, F_1, F_2, F_3]$	
Значення стеганоблоку ₂	Значення стеганоблоку ₁₀	Відповідний до стеганоблоку покриваючий файл
00	0	F_0
01	1	F_1
10	2	F_2
11	3	F_3

Далі необхідно сформувати матрицю поточного стану кластерів файлової системи із прив'язкою до кластерів покриваючих файлів – C . Для даного прикладу така матриця матиме вигляд, як зазначено у формулі 2.4.

$$C = [c_{2-3,n}^n, c_{4,0}^{F_0}, c_{5,1}^{F_0}, c_{6,2}^{F_0}, c_{7,3}^{F_0}, c_{8-10,n}^n, c_{11,0}^{F_1}, c_{12,1}^{F_1}, c_{13-15,n}^n, c_{16,2}^{F_1}, c_{17,0}^{F_2}, c_{18,1}^{F_2}, c_{19-21,n}^n, c_{22,0}^{F_3}, c_{23,1}^{F_3}, c_{24,2}^{F_3}, \dots] \quad (2.4)$$

де $c_{i,n}^n$ – означає, що даний кластер належить невідомому файлу, тобто такому файлу що не належить до набору покриваючих файлів F . Такі кластери не приймають участь у методі приховування даних шляхом перемішування кластерів, отже можна спростити формулу 2.4 “виколовши” зайві кластери. Тоді спрощена матриця поточного стану C^* матиме вигляд формули 2.5.

$$C^* = [c_{4,0}^{F_0}, c_{5,1}^{F_0}, c_{6,2}^{F_0}, c_{7,3}^{F_0}, c_{11,0}^{F_1}, c_{12,1}^{F_1}, c_{16,2}^{F_1}, c_{17,0}^{F_2}, c_{18,1}^{F_2}, c_{22,0}^{F_3}, c_{23,1}^{F_3}, c_{24,2}^{F_3}] \quad (2.5)$$

Наступний крок це створення матриці стану кластерів із прихованим повідомленням – C^{**} , базуючись на повідомленні $M = [01, 10, 11, 00] = [F_1, F_2, F_3, F_0]$, таблиці співвідношень стеганоблоків та набору покриваючих файлів (таблиця 2.2), та спрощеній матриці поточного стану C^* . Для цього необхідно переставити кластери покриваючих файлів. Таким чином щоб перші кластери співпадали із кластерами вказаними у повідомленні M . Тобто необхідно виконати перестановку $\pi(C^*) = C^{**}$. Схематично результат виконання перестановки зображено на рисунку 2.6.

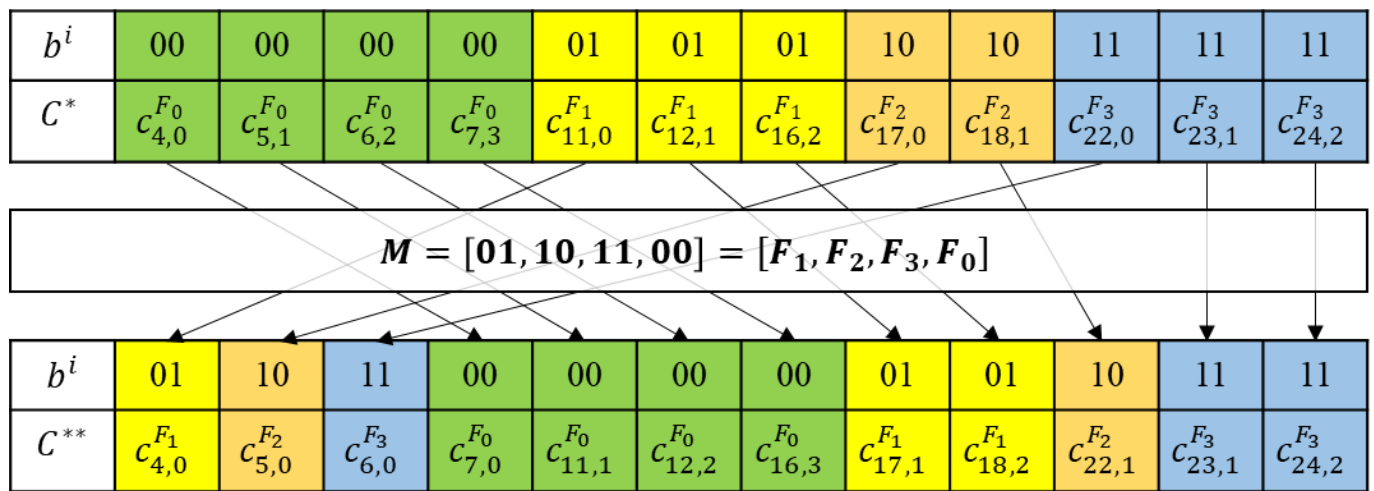


Рис. 2.6 Приклад приховування повідомлення M у множину кластерів C^* шляхом виконання перестановки кластерів

Для вилучення прихованого повідомлення необхідно зробити такі само кроки як і при приховуванні, а саме:

- визначити вихідні дані: p, F – кількість та набір покривельних файлів;
- вилучити ланцюги кластерів покриваючих файлів;
- із ланцюгів кластерів покриваючих файлів отримати матрицю поточного стану C^* ;
- із матриці C^* , знаючи параметри p, F вилучити повідомлення M^* .

Необхідно зазначити, що вилучене повідомлення M^* , може мати більшу довжину у стеганоблоках (b^i) за необхідну. А саме довжину зазначену у формулі 2.6.

$$M_{len}^* = C_{len}^* \quad (2.6)$$

Приховане повідомлення M , знаходиться у перших стеганоблоках вилученого повідомлення M^* , тобто $M \in M^*$. Для повідомлень текстового виду це не є проблемою, адже необхідне повідомлення буде читабельне, та відрізнятиметься від останніх стеганоблоків вилученого повідомлення M^* . Але для псевдовипадкових повідомлень (геши, ключі, контрольні суми), дана особливість є недоліком та потребує додаткових механізмів для задання необхідної довжини повідомлення M . До таких механізмів можуть належати:

- організаційні методи задання довжини повідомлення, тобто сторона приховання та сторона вилучення повідомлення заздалегідь знають довжину прихованого повідомлення;
- математичні, алгоритмічні методи, тобто приховане повідомлення може містити певний флаг як мітку закінчення повідомлення, або повідомлення містить інформацію про довжину прихованого повідомлення, та приймаюча сторона відсікає залишки вилученого повідомлення для отримання істинного M . Недоліком даного методу є те що додаючи флаг та/або мітку розміру повідомлення ми зменшуємо кількість біт які можуть використані для приховання самого повідомлення. Схематично метод вилучення прихованого повідомлення зображено на рисунку 2.7.

У даному підрозділі розглянуто та описано метод приховування інформації у структуру файлової системи FAT шляхом перемішування кластерів [17]. Також наведено приклад приховування повідомлення. Даний підрозділ є необхідний для подальшого аналізу методу та для розробки програмної реалізації. Головна особливість методу полягає у тому, що приховування повідомлення не додає зайвих артефактів у структуру файлової системи (нові файли чи зміна розміру існуючих файлів).

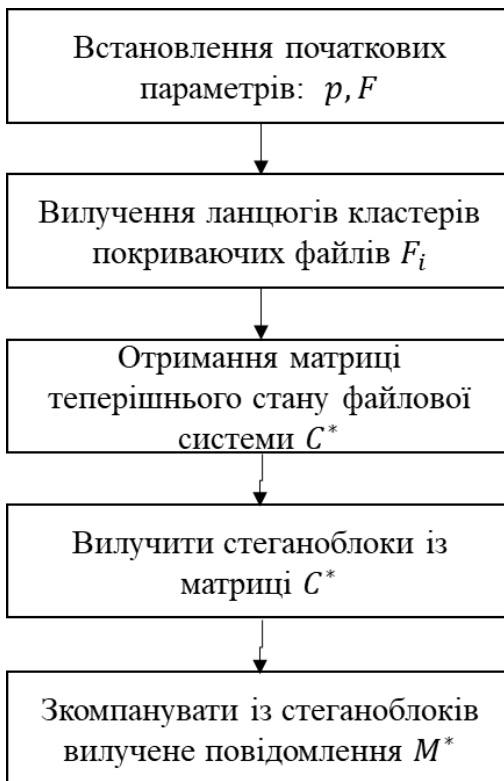


Рис. 2.7 Схематичне зображення методу вилучення інформації із структури файлової системи

2.3 Математична модель модифікованого методу приховування даних у структуру файлової системи

Даний метод базується на методі описаному у розділі 2.2 [17]. А отже для модифікованого методу застосуємо такі само позначення як і для базового методу. Нехай повідомлення M , яке буде вбудовано, позначається через масив $M = [b_0, b_1, \dots, b_{n-1}]$, де n – довжина повідомлення (кількість стеганоблоків), b_i – окремий стеганоблок (набір з $t + 1$ бітів), $i = 0, 1, \dots, n - 1$. Покрівельні файли позначимо як F_0, F_1, \dots, F_{p-1} , де p – кількість покрівельних файлів. Модифікований метод використовує додаткову особливість позначення кластеру, що дозволяє використовувати меншу кількість покрівельних файлів ніж при подібних параметрах у базовому методі. Тобто кількість покрівельних файлів може бути лише ступінь двійки, але при цьому розмір стеганоблоку можна збільшити на одиницю, тобто

$p - 1 = 2^m$, де $m \in N$, тобто належить множині натуральних чисел. Натуральне число m є однією з ключових інформацій необхідних для однозначного приховування та вилучення інформації. Параметр m безпосередньо впливає на розмір кожного із стеганоблоків b_i , $b_{ilen} = m$.

Іменами (назвами) покрівельних файлів F_i , є строки t_i , де $i = 0, 1, \dots, p - 1$. Задана послідовність покрівельних файлів F_i , $i = 0, 1, \dots, p - 1$, (або їх назв t_i , $i = 0, 1, \dots, p - 1$) також є ключовою інформацією. Тобто ключом для однозначного вилучення та приховування даних є кількість та порядок покрівельних файлів.

Для приховування даних необхідно визначити стеганоконтейнер та стеганоблок. Для цього зазначимо множину усіх кластерів файлової системи FAT – $C = [0, 1, \dots, FAT_{len}]$, FAT_{len} – довжина файлової системи у штук кластерів (довжина кожного кластеру залежить від налаштування файлової системи та може бути 4-64 КБ). Кожен покрівельний файл F_i розміщений у наборі кластерів – ланцюгу кластерів $C_{F_i} = [\{j | 0 \leq j \leq FAT_{len}\}]$, $0 < C_{F_i len} < FAT_{len}$. Тобто ланцюг кластерів покрівельного файлу F_i складається з кластерів що входять до множини усіх кластерів файлової системи. Та довжина ланцюга кластерів може бути від 1 кластера до усіх кластерів файлової системи. Кожен кластер може бути представлений як $c_{i,j}^{F_i}$, де i – порядок кластера у множині кластерів файлової системи C , j – порядок кластера у множині кластерів певного покрівельного файлу C_{F_i} . Саме порядок кластера у множині кластерів певного файлу є тією особливістю на якому засновано модифікований метод приховування даних у структуру файлової системи FAT. Ми визначили необхідні вихідні параметри: повідомлення яке необхідно приховати, кількість та набір покрівельних файлів із відповідними ланцюгами кластерів.

Модифікований метод базується на особливості порядку кластера у послідовності одного покриваючого файлу. Це означає що, можна визначити ще одну ступінь свободи, яка дозволяє збільшити пропускну здатність на одиницю. А отже виходячи з рівняння $p - 1 = 2^m$, можна розбити повідомлення на стеганоблоки

розміром у 1 біт кожен, та приховати таке повідомлення використовуючи лише 1 покрівельний файл [19, 20].

Нехай $c_{i,j}^{F_i}$ – j -ий кластер файлу F^i . Порядок кластеру у множині усіх кластерів файлової системи – i у даному випадку не є важливим. Визначимо ланцюг кластерів файлу $F^i = [c_{i1,0}^{F_i}, c_{i2,1}^{F_i}, \dots, c_{ij,j}^{F_i}, \dots, c_{in,n}^{F_i}]$, де n – довжина файлу F^i у кластерах. Введемо додаткову закономірність, зазначену у формулі 2.7.

$$c_{i,j}^{F_i} \rightarrow \begin{cases} i_1 - i_2 > 0 \text{ при } c_{i1,j}^{F_i}, c_{i2,j+1}^{F_i} \rightarrow 1 \\ i_1 - i_2 \leq 0 \text{ при } c_{i1,j}^{F_i}, c_{i2,j+1}^{F_i} \rightarrow 0 \end{cases} \quad (2.7)$$

Тобто, якщо теперішній кластер $c_{i1,j}^{F_i}$, покриваючого файлу F_i має значення індексу i_1 із множини кластерів файлової системи більший ніж за даний індекс i_2 у наступному кластері $c_{i2,j+1}^{F_i}$ файлу F_i , то нехай це відповідає 1 (єдиничному) біту, якщо навпаки, то нехай це відповідає 0 (нульовому) біту. Також така закономірність накладає обмеження у розмір повідомлення яке необхідно приховати – якщо повідомлення розбити на стеганоблоки b_i по 1 біту, то кількість кластерів файлу F^i повинна бути на один кластер більше ніж кількість стеганоблоків. Тобто $F_{len}^i = n - 1$ при $[b_0, b_1, \dots, b_{n-1}]$. Схематично дана закономірність вказана на рисунку 2.8.

Пояснюючи рисунок 2.8 необхідно зазначити що дано покриваючий файл із ланцюгом кластерів $F^i = [c_{i3,0}^{F_i}, c_{i2,1}^{F_i}, c_{i4,2}^{F_i}, c_{i1,3}^{F_i}]$, $i1, i2, i3, i4$ – будь які індекси кластерів із множини кластерів файлової системи C , при чому $i1 < i2 < i3 < i4$. У наступній таблиці рисунку кластери файлу F^i розташовані підряд з точки зору черговості кластерів множини C . Та схематично проведені лінії (1, 2, 3), порядок цих ліній відповідає порядку кластерів у файлу F^i . Закономірність формули 2.7 можна інтерпретувати як, якщо лінія між упорядковано розташованими кластерами (упорядковані за індексами множини C) одного файлу F^i прямує ліворуч (має

зворотній напрямком) то це відповідає 1 (одиничному) біту, а якщо лінія прямує праворуч (нормальний напрямок), то це 0.

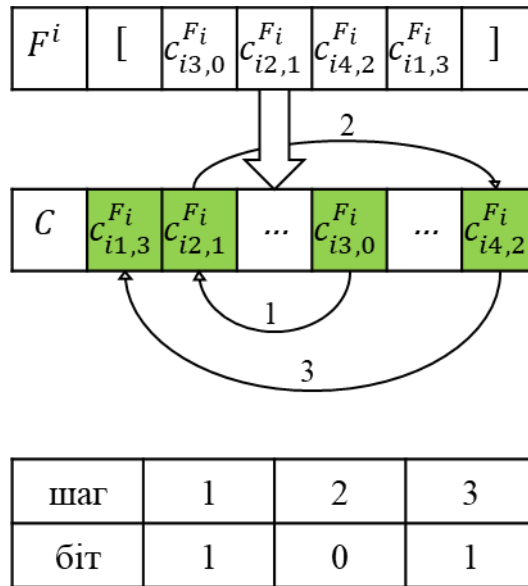


Рис. 2.8 Схематичний приклад закономірності вказаній у формулі 2.7

Файл кластери якого мають зворотній напрямок є *перемішаним* файлом. З точки зору фрагментованості файлу, перемішаний файл також є фрагментований.

Для приховування інформації модифікованим методом рекомендується зробити розбиття повідомлення за таким принципом. Нехай маємо повідомлення $M = [b_0, b_1, \dots, b_{n-1}]$ із стеганоблоками b_i де розмір кожного стеганоблоку у бітах дорівнює $m = \log_2 p + 1$. Тобто для прикладу, якщо маємо 4 покриваючих файли ($p = 4$), то повідомлення необхідно розбити по 3 біти ($b_{ilen} = 3$). Представимо кожен стеганоблок, як сукупність стеганоблоків $b_i = [b_i^B, b_i^M]$, де $b_{ilen}^B = m$, $b_{ilen}^M = 1$, тобто b_i^B – стеганоблок для базового методу приховування, b_i^M – стеганоблок для модифікованого методу. Отже повідомлення M матиме вигляд формули 2.8

$$M = \left[[b_0^B, b_0^M], [b_1^B, b_1^M], \dots, [b_{n-1}^B, b_{n-1}^M] \right] \quad (2.8)$$

Зіставимо кожен базовий стеганоблок b_i^B із відповідним покриваючим файлом із множини покриваючих файлів $b_i^B \rightarrow F^i$ [37, 44]. Тобто формула 2.8 матиме вигляд 2.9.

$$M = \left[[F^{i0}, b_0^M], [F^{i1}, b_1^M], \dots, [F^{ip-1}, b_{n-1}^M] \right] \quad (2.9)$$

Далі необхідно виконати перестановку матриці початкового стану C , так щоб новий стан відповідав повідомленню M лише із базових стеганоблоків – $M^B = \left[[F^{i0}], [F^{i1}], \dots, [F^{ip-1}] \right]$, тобто виконати приховування повідомлення базовим методом. Таким чином отримано проміжний стан матриці кластерів – $C^B = \pi^B(C)$. Наступне що необхідно виконати, це вилучити із повідомлення M набори стеганоблоків модифікованого методу із стеганоблоками базового методу b_i^B , що відповідають одному й тому покриваючому файлу, формула 2.10.

$$M^{Fi} = \left[\dots, [F^i, b_i^M], \dots, [F^i, b_j^M], \dots \right] = [b_0^M, b_1^M, \dots, b_j^M]^{Fi} \quad (2.10)$$

Да розбиття повідомлення M на повідомлення із модифікованих стеганоблоків необхідно зробити для кожного покриваючого файлу, тобто $M^M = [M^{F1}, M^{F1} \dots, M^{Fp}]$. Далі необхідно виконати перестановку кластерів кожного покриваючого файлу $F^i = [c_{i1,0}^{Fi}, c_{i2,1}^{Fi}, \dots, c_{ij,j}^{Fi}, \dots, c_{in,n}^{Fi}]$ по відповідному повідомленню M^{Fi} , використовуючи формулу 2.7. Таким чином остаточно перестановка π – це сукупність перестановок файлів F^i із відповідними M^{Fi} , що відповідає формулі 2.11.

$$C^M = \forall \pi^M(C^B)_{x,y}; x \in M^M, y \in F \quad (2.11)$$

Отже, як висновок можна стверджувати, що модифікований метод приховування інформації розширює базовий метод, це досягається шляхом використання додаткової властивості розміщення кластерів у структурі файлової системи FAT, а саме порядок кластеру у множині кластерів одного покриваючого файлу. У даній роботі розглянуто варіант розбивки повідомлення на базову частину, та модифіковані частини, шляхом розбиття кожного стеганоблоку на підстеганоблоки та групування їх за приналежністю до покриваючого файлу [45]. Але не виключено існують й інші особливості розбиття повідомлення, та використання властивості зазначеної у формулі 2.7. Даний метод схематично показаний на рисунку 2.9.

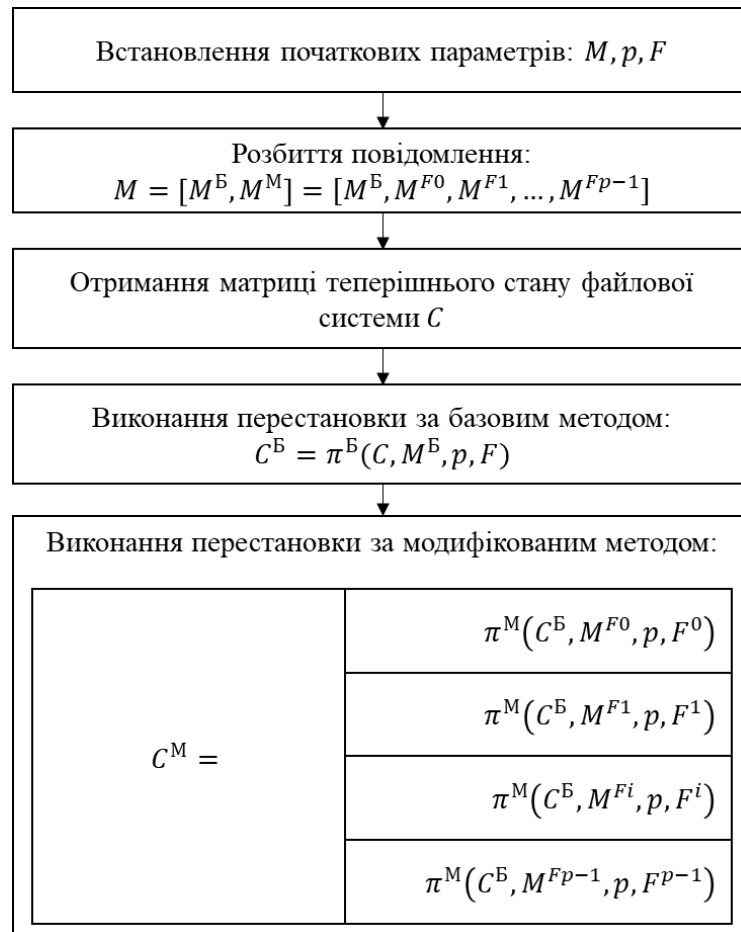


Рис. 2.9 Схематичне зображення методу приховування інформації у структуру файлової системи модифікованим методом

Вилучення повідомлення виконується подібно до вилучення повідомлення як у базовому методі, тобто необхідно скомпонувати стеганоблоки базового повідомлення (таблиця 2.2), а потім доповнити відповідні стеганоблоки значущими бітами за формулою 2.9. Останнім кроком буде конкатенація стеганоблоків у вилучене повідомлення.

Як приклад модифікованого методу приховування інформації у структуру файлової системи FAT шляхом перемішування кластерів покриваючих файлів задамо вихідні параметри, а саме (зادля наочності задамо такі само вихідні параметри як і при базовому методі):

- M – повідомлення яке необхідно приховати. $M = [0,1,1,0,1,1,0,0]$.
- p, F – кількість та набір покривельних файлів. $p = 4$, $F_0 = [4,5,6,7]$, $F_1 = [11,12,16]$, $F_2 = [17,18]$, $F_3 = [22,23,24]$.

Отримавши вихідні параметри наступне що необхідно зробити, це розбити повідомлення на стеганоблоки b_i розміром по $b_{ilen} = m + 1$ біт кожен, де $p = 2^m$. Тобто $4 = 2^m \rightarrow m = 2 \rightarrow b_{ilen} = 3$. А значить повідомлення $M = [0,1,1,0,1,1,0,0]$ матиме таку розбивку по блоках – $M = [011,011,000]$. Так як останній стеганоблок не має необхідної довжини у бітах, то доповнимо його нулями. Виділимо базове повідомлення та співвіднесемо стеганоблоки базового повідомлення із покриваючими файлами (відповідно до таблиці 2.2): $M = [[F_1, 1], [F_1, 1], [F_0, 0]]$. Вилучим базове повідомлення, та набір модифікованих повідомлень: $M^B = [F_1, F_1, F_0]$, $M^{F_0} = [0]$, $M^{F_1} = [1,1]$. Далі процес приховування повідомлення такий само як і при базовому методі. Отже необхідно сформувати матрицю поточного стану кластерів файлової системи із прив'язкою до кластерів покриваючих файлів – C . Для даного прикладу така матриця матиме вигляд: $C = [c_{2-3,n}^n, c_{4,0}^{F_0}, c_{5,1}^{F_0}, c_{6,2}^{F_0}, c_{7,3}^{F_0}, c_{8-10,n}^n, c_{11,0}^{F_1}, c_{12,1}^{F_1}, c_{13-15,n}^n, c_{16,2}^{F_1}, c_{17,0}^{F_2}, c_{18,1}^{F_2}, c_{19-21,n}^n, c_{22,0}^{F_3}, c_{23,1}^{F_3}, c_{24,2}^{F_3}, \dots]$, такий само як і у формулі 2.4. Для спрощення “виколомо” кластери не покриваючих файлів. Тоді матриця матиме

спрощений вигляд як у формулі 2.5: $C = [c_{4,0}^{F_0}, c_{5,1}^{F_0}, c_{6,2}^{F_0}, c_{7,3}^{F_0}, c_{11,0}^{F_1}, c_{12,1}^{F_1}, c_{16,2}^{F_1}, c_{17,0}^{F_2}, c_{18,1}^{F_2}, c_{22,0}^{F_3}, c_{23,1}^{F_3}, c_{24,2}^{F_3}]$.

Наступний крок це створення матриці стану кластерів із прихованою базовою складовою повідомлення – C^B , базуючись на повідомленні $M^B = [F_1, F_1, F_0]$, таблиці співвідношень стеганоблоків та набору покриваючих файлів (таблиця 2.2), та спрощеній матриці поточного стану C . Для цього необхідно переставити кластери покриваючих файлів таким чином щоб перші кластери співпадали із кластерами вказаними у повідомленні M^B . Тобто необхідно виконати перестановку $\pi(C) = C^B$. Схематично результат виконання перестановки базового повідомлення зображено на рисунку 2.10, що відповідає першій частині методу приховування. Те що кількість перестановок у даному прикладі менша, це скоріше виняток, але детальніше аналіз перестановок розглянуто далі [17, 42, 46, 47].

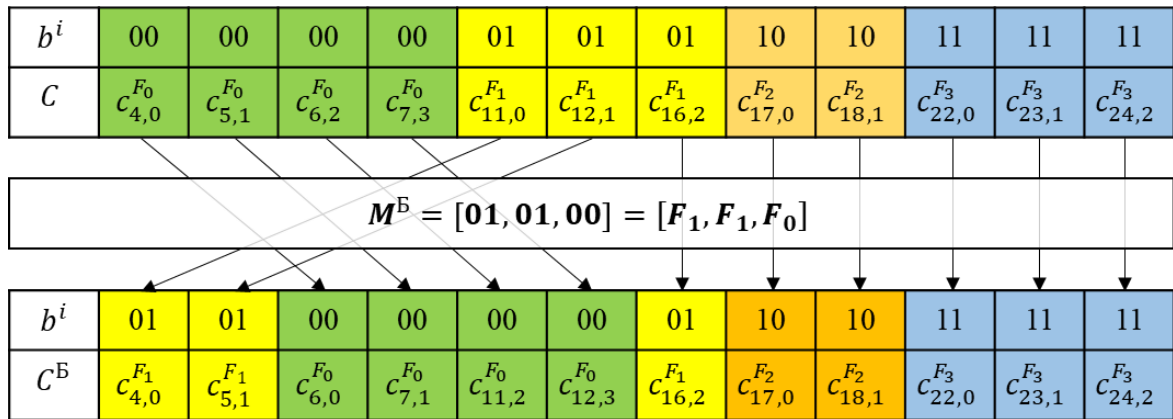


Рис. 2.10 Приклад приховування повідомлення M^B у множину кластерів C .

На цьому етапі приховано лише базову складову повідомлення. Далі необхідно виділити ланцюги кластерів покриваючих файлів, які мають відповідні модифіковані повідомлення, із матриці C^B . А саме $F^0 = [c_{6,0}^{F_0}, c_{7,1}^{F_0}, c_{11,2}^{F_0}, c_{12,3}^{F_0}]$ та $F^1 = [c_{4,0}^{F_1}, c_{5,1}^{F_1}, c_{16,2}^{F_1}]$, так як частка для модифікованого приховування містить лише стеганоблоки для цих файлів ($M^{F^0} = [0]$, $M^{F^1} = [1,1]$). Перестановка над кластерами відбувається у

відповідності до формули 2.7, у межах кластерів покриваючих файлів, тобто необхідно робити перестановку ізольовано від кластерів інших покриваючих файлів. Схематично така перестановка вказана на рисунку 2.11.

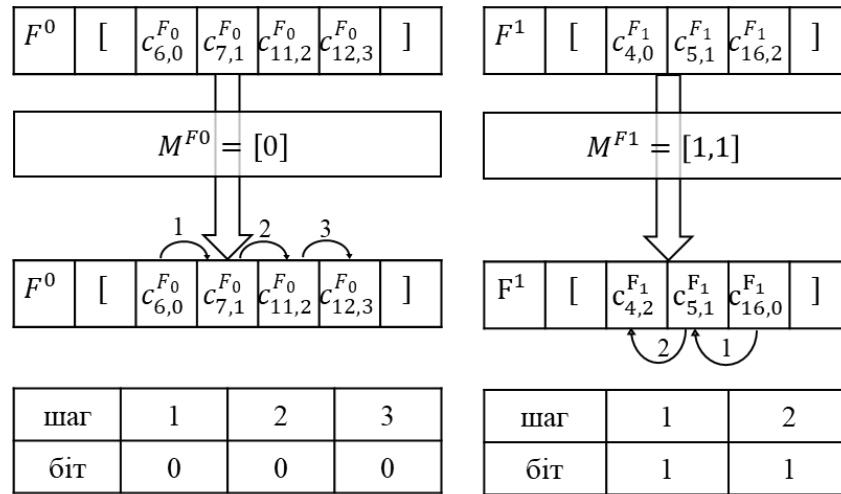


Рис. 2.11– Приклад приховування повідомлень M^{F0} , M^{F1} у множину кластерів C^B

Остаточним результатом приховування повідомлення M буде матриця стану, після перестановки за базовою складовою повідомлення та перестановками за модифікованою складовою повідомлення [19, 37, 48]. Схематично такий результат зазначено у рисунку 2.12.

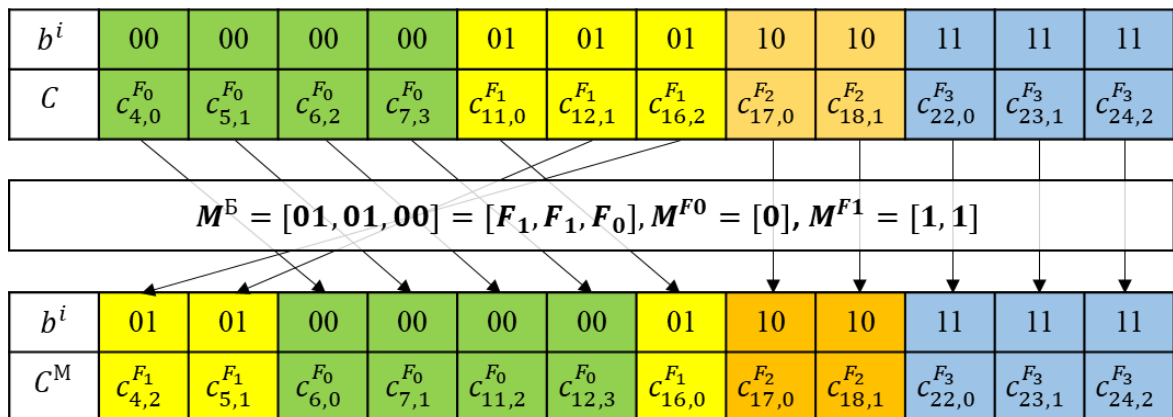


Рис. 2.12 – Приклад приховування повідомлення модифікованим методом у множину кластерів C

Порівнюючи матриці стану після приховування базової складової (рисунок 2.9) та модифікованих складових повідомлення, можна стверджувати, що при виконанні модифікованого методу фрагментованість та переплетеність покриваючих файлів може бути більша аніж при використанні базового методу приховування інформації у структуру файлової системи FAT шляхом перемішування кластерів покриваючих файлів. Але, модифікований метод дозволяє приховати більше інформації при тих самих вихідних параметрах, аналіз методів розглянуто у розділі 3 [19, 37].

Висновки до розділу 2

У даному розділі розглянуто властивості структури файлової системи FAT що можуть бути використані для приховування інформації, також проведена оцінка описаних властивостей, та обрана властивість на основі якої розроблено методи. А саме було описано метод приховування та видалення інформації шляхом перемішування кластерів покриваючих файлів. Особливість даного методу полягає у тому що, результат приховування повідомлення важко виявити, так як приховане повідомлення не призводить до артефактів у структурі файлової системи, тобто не створюються нові файли та/або не змінюється розмір існуючих файлів, а лише збільшується рівень фрагментованості деяких файлів. Що є також результатом нормального використання файлової системи. Також розроблено модифікований метод, сутність якого полягає у використанні додаткової особливості індексування кластерів покриваючих файлів, а саме використовується індекси кластерів у межах одного покриваючого файлу, та на базованні якого виконується додаткова перестановка, що дозволяє приховати більше інформації при подібних вихідних параметрах а ніж базовим методом, але й рівень фрагментованості також буде більший.

Таким чином у даному розділі була виконана перша і друга часткова задача, а саме проведено дослідження існуючих методів стеганографічного приховування

інформації у структуру файлових систем та розроблено метод підвищення пропускної здатності кластерних стеганосистем (модифікований метод приховування інформації). Та уперше отримано метод підвищення пропускної здатності кластерної стеганосистеми на основі врахування додаткової залежності місць розміщення кластерів у межах одного покриваючого файлу системи.

Результати досліджень даного розділу наведено в публікаціях здобувача: [19, 37, 39 44, 45].

РОЗДІЛ 3

АНАЛІЗ МЕТОДІВ ПРИХОВУВАННЯ ІНФОРМАЦІЇ ТА УДОСКОНАЛЕННЯ МАТЕМАТИЧНОЇ МОДЕЛІ ОЦІНКИ ОСНОВНИХ ПАРАМЕТРІВ КЛАСТЕРНИХ СТЕГАНОСИСТЕМ

Розробивши методи приховування інформації у структуру файлової системи FAT шляхом перемішування кластерів покриваючих файлів доцільним є аналіз базового методу та модифікованого методу, що дозволить зробити рекомендації щодо використання методів, та необхідних властивостей програмної реалізації, для часової оцінки роботи методів. Для цього зроблено порівняльний аналіз кількісних параметрів методів, проаналізовано файлові системи на можливість використання методів та зроблено часову оцінку методів [16, 39, 41, 49].

3.1 Аналіз пропускної здатності методів 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 стеганоблоків b_i однакова, наприклад, якщо існує лише два типи стеганоблоків $b_0 = 0$ та $b_1 = 1$, то усе повідомлення складається із стеганоблоків b_0, b_1 при чому $n = m$ при $[b_0^0, b_0^1, \dots, b_0^n], [b_1^0, b_1^1, \dots, b_1^m]$.

Нехай $Data$ – загальний об'єм інформації, в байтах, у файловій системі, що займають покриваючі файли, $Data = Const. Size_{cl}$ – розмір одного кластеру у байтах,

$Size_{Cl} \in \{2048, 4096, 8192, \dots, 65536\}$. Num_{Cl} – загальна кількість кластерів, що міститься у покриваючих файлах. $Size_{Fl}$ – розмір, у байтах, одного покриваючого файлу. Num_{Fl} – кількість покриваючих файлів. STG_{SIZE} – розмір стеганограми, тобто розмір повідомлення що необхідно приховати, у байтах.

Спираючись на надані умовності та позначення можна стверджувати, що пропускна здатність – V дорівнює співвідношенню розміру стеганограми – STG_{SIZE} до загального розміру накопичувача – $Data$, формула 3.1.

$$V = \frac{STG_{SIZE}}{Data} \quad (3.1)$$

Так як розмір файлової системи прийнято за константу, а покриваючи файли рівні за розміром, то можна стверджувати що: з одного боку розмір файлової системи це добуток кількості покриваючих файлів та розміру кожного покриваючого файлу (формула 3.2), а з іншого боку це добуток кількості кластерів файлової системи та розмір одного кластеру (формула 3.3)

$$Data = Size_{Fl} \times Num_{Fl} \quad (3.2)$$

$$Data = Size_{Cl} \times Num_{Cl} \quad (3.3)$$

Так як одним кластером можна відобразити один стеганоблок b_i , а розмір одного стеганоблоку залежить від кількості покриваючих файлів p , як логарифм двійковий ($p = 2^m \rightarrow m = \log_2 p$), то можна стверджувати, що розмір стеганограми у байтах займає:

$$STG_{SIZE} = \frac{Num_{Cl} \times \log_2(Num_{Fl})}{8} \quad (3.4)$$

Ділення на 8 необхідно саме для того щоб визначити довжину у байтах, а не у бітах, так як розмірність кожного стеганоблоку визначена у бітах. Формула 3.4 визначає розмірність стеганограми у байтах для *базового методу*.

Для модифікованого методу приховування даних у структуру файлової системи FAT розмір одного стеганоблоку також залежить від кількості обраних покриваючих файлів ($p = 2^{m-1} \rightarrow m = \log_2(p) + 1$), але так як взята за увагу додаткова властивість то кількість біт збільшується на 1 для кожного стеганоблоку. Окрім того необхідно зазначити, що при використанні властивості описаній у формулі 2.7, останній кластер кожного покриваючого файлу не несе інформативності, а лише слугує для визначення прихованого біту для попереднього кластеру. Отже можна стверджувати, що розмір стеганограми у байтах займає:

$$STG_{SIZE} = \frac{(Num_{Cl} - Num_{Fl}) \times (\log_2(Num_{Fl}) + 1)}{8} \quad (3.5)$$

Наступним кроком оцінимо залежність пропускну здатності V 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 3.2 та 3.3 можна вивести наступну залежність:

$$Num_{Cl} = \frac{Size_{Fl} \times Num_{Fl}}{Size_{Cl}} \quad (3.6)$$

Далі отриману залежність із формули 3.6 підставимо у формулу 3.4:

$$STG_{SIZE} = \frac{\frac{Size_{Fl} \times Num_{Fl}}{Size_{Cl}} \times \log_2(Num_{Fl})}{8} \quad (3.7)$$

Для спрощення аналізу у формулі 3.7 константні значення необхідно прийняти за одиницю, та значеннями наближеними до нуля – знехтувати. Значення отримане з $\log_2(Num_{FL})$ приймемо рівним одиниці. Як результат отримаємо що розмір стеганограми має зворотню залежність до розміру кластеру – $STG_{SIZE} = 1 / (Size_{CL})$. Тобто чим більший розмір кластеру покриваючого файлу, тим менше повідомлення можна приховати використовуючи метод приховування інформації шляхом перемішування кластерів і як висновок тим менша пропускна здатність методу – $V \uparrow \leftrightarrow Size_{CL} \downarrow$. Дана залежність однакова як для базового методу так і для модифікованого [49].

Оцінимо 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 кількість кластерів також константна – $Num_{CL} = Const$. Узявши за основу формулу 3.4, замінимо константні значення одиницею, та отримаємо, що $STG_{SIZE} = \log_2(Num_{FL})$. Тобто чим більше покриваючих файлів використовується у роботі методу як вихідні параметри тим більше повідомлення можна приховати, і як висновок, тим більша пропускна здатність методу – $V \uparrow \leftrightarrow Num_{FL} \uparrow$. Для модифікованого методу залежність така сама але у розмір стеганограми у два рази більший при відповідних умовах. Тобто при рівних умовах, використовуючи модифікований метод можна приховати у два рази більше інформації (якщо знехтувати константними величинами). Можна стверджувати що дана рівність $STG_{SIZE} = \log_2(Num_{FL} + 1)$ вірна для модифікованого методу.

Наступним кроком, виявивши залежності пропускної здатності V від розміру кластеру та від кількості покриваючих файлів, для отримання числового значення оцінки, підставимо числові значення замість умовно константних параметрів: $Data$, $Size_{CL}$, Num_{FL} . Візьмемо найпоширеніший за розміром вид флеш-накопичувача, нехай це буде накопичувач розміром 8Гб. Тобто $Data = 7780302848$ байт, кількість

покриваючих файлів нехай дорівнює двом – $Num_{Fl} = 2$. Існують різні за розміром кластери у структурі файлової системи, візьмемо мінімально можливий розмір – $Size_{Cl} = 2048$ байт. Знаючи при $Num_{Fl} = 2$ розмір одного стеганоблоку дорівнює один біт, то можна стверджувати що максимально можливий розмір повідомлення дорівнює кількості кластерів покриваючих файлів. Підставивши числові значення розміру файлової системи та розміру одного кластеру до формули 3.3 можна знайти кількість кластерів: $Num_{Cl} = 7780302848 / 2048 = 3799952$. Підставивши отримане значення до формули 3.4 та 3.5 отримаємо максимально можливий розмір стеганограми у байтах: $STG_{SIZE} = 3799952 / 8 = 474994$ байт для базового методу та $STG_{SIZE} = (3799952 * 2) / 8 = 949744$ для модифікованого методу приховування даних [49, 50].

Далі зафіксуємо кількість покриваючих файлів $Num_{Fl} = 2$, та будемо змінювати розмір кластеру із множини можливих розмірів кластерів $Size_{Cl} \in [2048, 4096, 8192, \dots, 65536]$. Отримані результати занесемо до таблиці 3.1.

Таблиця 3.1

Залежність розміру стеганограми STG_{SIZE} від розміру кластеру $Size_{Cl}$

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

За результатами таблиці 3.1 можна стверджувати, що розмір стеганограми залежить від розміру кластеру у зворотній пропорційності. Для наочності побудовано графіки описаної залежності для базового та для модифікованих методів, рисунок 3.1.

Що дає можливість впевнитись у логарифмічній залежності. Та як висновок рекомендовано використовувати файлову систему із мінімально можливим розміром кластеру, що дозволяє приховати повідомлення більшого розміру. Також можна зазначити що модифікований метод має більший вплив при меншому розмірі кластеру, тобто від певного розміру кластеру, використання модифікованого методу може витрачати більше часу ніж надавати пропорційну за часом пропускну здатність, більш детально аналіз часових властивостей надано далі.

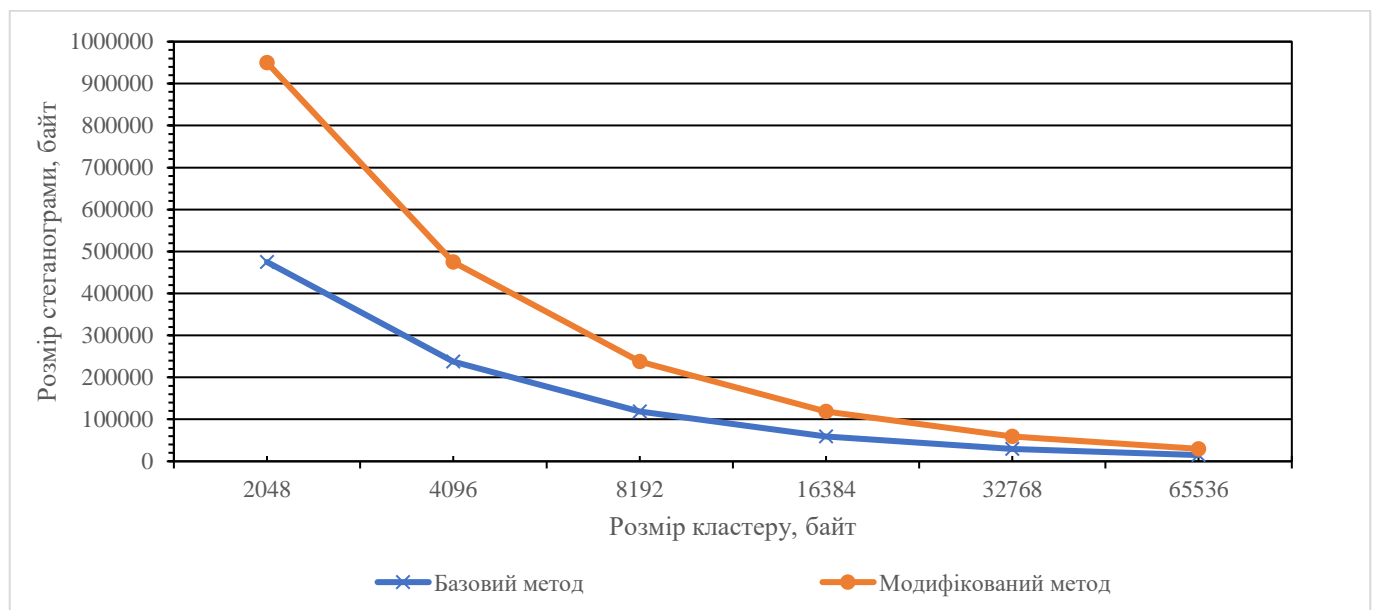


Рис. 3.1 Графік залежності розміру стеганограми від розміру кластеру

Наступним кроком виявимо аналогічно залежність і від кількості покриваючих файлів, для цього зафіксуємо загальний розмір накопичувача – $Data = 7780302848$ байт, та розмір одного кластеру – $Size_{Cl} = 2048$ байт, кількість покриваючих файлів прийматиме такі значення – $Num_{Fl} \in [2, 4, 8, 16, 32, 64]$. Хоча максимальна кількість файлів близька до загальної кількості кластерів, а теоретично майже необмежена, 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. Числові значення підставлятимемо до формул 3.4 та

3.5 при цьому необхідно зазначити що так як змінюється кількість файлів то й розмір кожного файлу також буде змінюватись відповідно. Занесемо отримані результати до таблиці 3.2.

Таблиця 3.2

Залежність розміру стеганограми від кількості покриваючих файлів

Кількість покриваючих файлів	Розмір одного файлу, байт	Базовий метод, байт	Модифікований метод, байт
2	3 890 151 424	474872	949744
4	1945075712	949744	1424616
8	972537856	1424616	1899488
16	486268928	1899488	2374360
32	243134464	2374360	2849232
64	121567232	2849232	5698464

Оцінивши результати таблиці 3.2 можна стверджувати що розмір стеганограми збільшується із кількістю покриваючих файлів, при чому для модифікованого методу необхідно у два рази менше покриваючих файлів ніж для базового методу для приховування одного й того за розміром повідомлення. Для наочності за результатами таблиці побудовано графіки, рисунок 3.2.

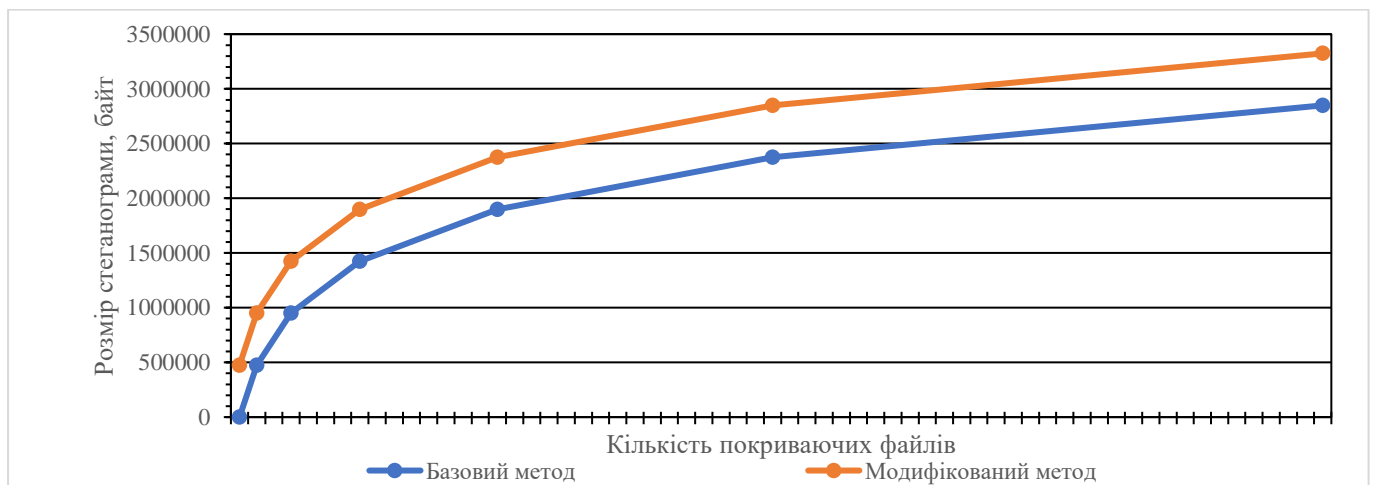


Рис. 3.2 Залежності розміру стеганограми від кількості покриваючих файлів

Наступним кроком є отримання значень можливого розміру стеганограми при $Size_{cl} \in [2048, 4096, 8192, \dots, 65536]$ та $Num_{Fl} \in [2, 4, 8, 16, 32, 64]$, також зафіксувавши розмір файлової системи. Результати занесено до зведеної таблиці 3.3 для базового методу та 3.4 для модифікованого методу

Таблиця 3.3

Розмір стеганограми у залежності від кількості покриваючих файлів та розміру одного кластеру для базового методу

КПФ PK \	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

Таблиця 3.4

Розмір стеганограми у залежності від кількості покриваючих файлів та розміру одного кластеру для модифікованого методу

КПФ PK \	2	4	8	16	32	64
2048	949744	1424616	1899488	2374360	2849232	3324104
4096	474872	712308	949744	1187180	1424616	1662052
8192	237436	356154	474872	593590	712308	831026
16384	118718	178077	237436	296795	356154	415513
32768	59359	89038,5	118718	148397,5	178077	207756,5
65536	29679,5	44519,25	59359	74198,75	89038,5	103878,3

У таблиці 3.3, 3.4 аббревіатури мають такі тлумачення:

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

Аналізуючи отримані дані із таблиць 3.3 та 3.4, можна стверджувати, що при збільшенні розміру кластеру у двічі, задля утримання розміру стеганограми необхідно збільшити кількість покриваючих файлів:

- у другу ступінь (x^2), для базового методу;
- у третій ступінь (x^3), для модифікованого методу.

Що також наштовхує на висновок, що модифікований метод має більше переваг при меншому розмірі кластеру. Схематично, для наочності, однакові розмірі стеганограми виділені одним кольором.

Роблячи 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 – 3.3 Мб, при максимальній кількості файлів – 64, та при мінімальному розмірі кластеру – 2048 байт. Аналізуючи розмір стеганограми до загального розміру накопичувача зроблено висновок, що базовий метод дозволяє приховати повідомлення розміром 0.002-0.036% від загального розміру файлової системи, а модифікований метод дозволяє приховати 0.003-0.04%. Звісно можна збільшити кількість покриваючих файлів, але при зовеликій кількості файлів збільшиться рівень фрагментованості та час необхідний на приховання повідомлення. Таким чином дані методи рекомендовано використовувати для прихованої передачі чи збереження повідомлення відносно невеликого розміру, для прикладу це можуть бути ключи, паролі, геши, контрольні суми, але для передачі чи збереження таких повідомлень важливим є ступінь захисту прихованого повідомлення від детектування, що проаналізовано у наступному підрозділі [49].

3.2 Аналіз захищеності методів приховування даних шляхом перемішування кластерів до детектування

Для того щоб детектувати приховану інформацію у структурі файлової системи FAT необхідно проаналізувати та виявити певні відхилення від стандартної роботи файлової системи. Оскільки методи приховування повідомлення змінюють структурні поля файлової системи, їх детектування буде полягати у пошуку аномальних значень відповідної структури [49, 51]. Для прикладу:

- детектування прихованої інформації у зарезервовані сектори полягає у пошуку будь яких інформаційних значень відмінних від нульових;
- детектування прихованої інформації у старших квартетах покажчиків вільних кластерів у FAT таблицях полягає у пошуку будь яких інформаційних значень відмінних від заданих специфікацією;
- і тому подібне.

Таким чином детектування полягає у порівнянні структури файлової системи з прихованою інформацією із структурою файлової системи, заданою у специфікації. Тобто необхідно зробити зліпки структури файлової системи до приховування та після приховування повідомлення. Будемо вважати, що стеганографічні методи, для яких вдалося виявити певні відмінності, є *нестійкими до детектування* [51].

Методи, що описані у розділі 2 (базовий та модифікований метод приховування інформації у структуру файлової системи FAT шляхом перемішування кластерів певних покриваючих файлів), використовують спеціальну послідовність кластерів файлів, що є цілком нормальною роботою файлової системи. Отже детектувати приховування інформаційного повідомлення шляхом порівняння зліпків не є надійним, та потребує детальнішого аналізу. Для цього необхідно проаналізувати рівень фрагментації даних та, враховуючи отримані результати, порівняти їх з деякими «стандартними» властивостями, притаманними для файлових систем без прихованої інформації.

У інформаційних технологіях, *фрагментація* це поділ чого-небудь на безліч дрібних розрізнених фрагментів. Значний рівень фрагментації негативно впливає на можливості файлової системи та апаратного носія інформації [49, 52]:

- збільшує кількість переміщень головки зчитування даних, що, в свою чергу, зменшує пропускну здатність. Для накопичувачів із SSD технологією збереження інформації, дана особливість майже не несе негативних наслідків на роботу файлової системи;

- зменшує час придатності носія інформації;

- у певних файлових системах (на відміну від FAT), неможливо записати дані у фрагментовані області, хоч і загальне вільне місце задовольняє необхідним умовам [53].

Для зменшення рівня фрагментації використовують зворотній процес до фрагментації, а саме– *дефрагментацію*. Це штучно створений процес, сутність якого полягає у розміщенні даних файлів (кластерів) один за одним. Тобто кластери файлу F^i , розміщені таким чином щоб індекси послідовних кластерів, у межах даного файлу, були послідовні й у межах індексів усієї файлової системи. Тобто необхідно щоб виконувалась умова зазначена у формулі 3.8:

$$F^i = [c_{i,j}^{Fi}, c_{i+1,j+1}^{Fi}, \dots, c_{i+n,j+n}^{Fi}] \quad (3.8)$$

де:

- F^i – певний покриваючий файл;
- $c_{i,j}^{Fi}$ – кластер покриваючого файлу F^i ;
- i – індекс кластеру у межах файлу F^i ;
- j – індекс кластеру у межах файлової системи;
- n – довжина файлу F^i у кластерах.

Загалом рівень фрагментації збільшується у наслідок багатопоточного доступу до файлової системи таким чином, що одночасно виконувалися операції запису та видалення декількох файлів. Тобто рівень фрагментації може збільшуватися при:

- роботі операційної системи. Для прикладу під час роботи операційної системи Windows, Linux увесь час виконується зчитування, запис та видалення конфігураційних файлів, файлів логування (збереження інформації для подальшого аудиту та моніторингу) і тому подібних;

- роботі процесів, що змінюють або видаляють дані із файлової системи. Більшість процесів мають доступ до файлової системи для коректного виконання процесів, вони можуть зберігати проміжну інформацію у файли, у тому числі й у тимчасові файли;

- копіювання файлів із одного носія інформації на інший. Даний механізм створює нові файли на носії, що може призвести до фрагментованості файлів, особливо якщо цільовий носій інформації не має достатньо неперервного вільного місця для збереження файлів;

- завантаження даних через інтернет. Даний процес подібний до попереднього, через те що так само на носії інформації створюються нові файли, але також завантаження через інтернет має більший вплив на рівень фрагментації так як зазвичай швидкість завантаження файлів менша ніж копіювання між носіями, і найголовніше, дані завантаження через інтернет можна призупинити, що по замовчуванню створить неповний файл у одній області файлової системи, а потім після продовження завантаження залишок файлу буде завантажено у наступну вільну область файлової системи.

Окремо виділяють *фрагментацію файлової системи* – це неспроможність файлової системи розмістити пов'язані дані послідовно (неперервно). Це явище притаманне файловим системам для зберігання даних, що дозволяють пряму модифікацію даних. Тобто, для прикладу, файлова система має необхідну кількість вільних кластерів, але дані кластери розміщені непослідовно, як показано на рисунку

3.3. Таке явище має наслідок, а саме при створенні нового файлу данні такого файлу будуть вже із певним рівнем фрагментації [49].

1	2	3	4	5	6	7	1	2	
	8	9	10	1	2	3	1	2	3
4	5	6		4	5	1	2	3	4
5	6	7							

Рис. 3.3 Фрагментація файлової системи (файлова система має непослідовні вільні кластери)

Також окремо виділяють *фрагментацію даних* – рівень роздробленості послідовності кластерів одного файлу, тобто, чим більше рівень фрагментації тим більша кількість несусідніх кластерів одного й того ж файлу. Для прикладу на рисунку 3.4 наведено фрагментацію різних файлів, на якому позначено різні випадки чергування окремих кластерів файлів А та В.

А	А	В	А	А	В	Фрагментація (А) = 2; Фрагментація (В) = 2
А	А	А	А	В	В	Фрагментація (А) = 1; Фрагментація (В) = 1
В	А	А	А	А	В	Фрагментація (А) = 1; Фрагментація (В) = 2

Рис. 3.4 – Приклад фрагментації даних файлів А та В

Для формальної оцінки рівня фрагментації окремого файлу введемо кількісний показник $\varphi(F^i)$ – рівень фрагментації даних файлу F^i . Рівень фрагментації відповідає формулі 3.9:

$$\varphi(F^i) = \sum_{k=0}^n \begin{cases} j_2 - j_1 = 1 \rightarrow 0 \\ j_2 - j_1 \neq 1 \rightarrow 1 \end{cases}; c_{i+k,j_1}^{Fi}, c_{i+k+1,j_2}^{Fi} \quad (3.9)$$

Тобто формула 3.9 тлумачить, що для отримання рівня фрагментації файлу F^i необхідно послідовно “пройтися” по усім кластерам файлу, та порахувати різницю між індексами j – індексами у межах усієї файлової системи. Якщо різниця між індексами дорівнює одиниці, то ці два кластери розміщені поруч у вірній зростаючій послідовності. Але, якщо різниця між індексами більше одиниці то між кластерами є кластери іншого файлу, тому необхідно додати одиницю до рівня фрагментації – $\varphi(F^i)$. У разі якщо різниця між індексами j менша за нульове значення, то це що кластери розміщені у зворотному напрямку, що також призводить до збільшення рівня фрагментації. Файли які мають кластери розташовані у зворотному напрямку називаються *переплетеними*. Рівень переплетеності є частковим випадком рівня фрагментації даних [54].

Для однозначного тлумачення рівня фрагментації даних файлу, також описано алгоритм із використанням псевдокоду, алгоритм зображено на рисунку 3.5. На рисунку позначено:

- A – позначення файлу певної довжини із відповідним ланцюгом кластерів;
- *кластер* – вказує на індекс кластеру у FAT таблиці;
- *фрагментація* – значення показника фрагментації $\varphi(A)$ (початкове значення дорівнює 1, так як кожен файл містить хоча б один фрагмент).

Таким чином, показник фрагментації $\varphi(A)$ розраховується наступним чином: якщо номер поточного кластеру відрізняється від номеру попереднього кластеру більш ніж на 1 тоді збільшуємо рівень фрагментації на одиницю (оскільки кластери файлу можуть розміщуватись у зворотному напрямку, необхідно оцінювати абсолютну різницю номерів). Отже, фактично, показник $\varphi(A)$ визначає кількість окремих фрагментів для файлу A у послідовному записі ланцюга кластерів файлової системи.

```

фрагментація = 1;
Файл A =[кластер, кластер, ..., кластер];
for(int i = 0; i < A.довжина; i++){
    якщо (A[i] - A[i-1]) > 1
    {
        ТО фрагментація++;
    }
}
ПОВЕРНУТИ фрагментація;

```

Рис. 3.5 Псевдокод знаходження рівня фрагментації файлу A

Введений показник фрагментації $\varphi(A)$ дозволяє кількісно оцінити рівень фрагментації окремих файлів. Виходячи із припущення про рівномірність фрагментації значення цього показника для різних файлів повинно бути приблизно однаковим. Отже дослідження фрагментованості файлової системи та оцінка рівня фрагментації $\varphi(A)$ для кожного файлу може дати критерій (правило) детектування стеганографічного приховування: якщо значення $\varphi(A)$ для певних файлів значно відрізняється від усередненого рівня фрагментації $\bar{\varphi}$ по всім файлам тоді файл A може містити інформацію приховану шляхом перемішування кластерів, що й означає що приховану інформацію детектовано [49, 54].

Слід відмітити, що показник фрагментованості $\varphi(A)$ дає абсолютне значення фрагментованості, яке безпосередньо залежить від загальної кількості кластерів у даному покриваючому файлі A . Отже, для надійного детектування приховування інформаційного повідомлення доцільним є застосування нормованого показника, яке нівелює різницю між файлами за розмірами у кластерах, що надає більш об'єктивну оцінку захищеності прихованої інформації від детектування. Формульне вираження нормованого рівня фрагментації вказано у формулі 3.10.

$$\|\varphi(A)\| = \frac{\varphi(A)}{n_A} \quad (3.10)$$

де n_A – розмір файлу A у кластерах. Тоді для виявлення можливого факту приховування інформації у структуру файлової системи FAT шляхом перемішування кластерів необхідно порівнювати значення фрагментованості кожного покриваючого файлу із усередненим значення фрагментованості по усім файлам у файловій системі, яке розраховується за формулою 3.11.

$$\bar{\varphi} = \frac{1}{N} \sum_{i=0}^N \varphi(A_i) \quad (3.11)$$

де N – загальна кількість файлів у файловій системі. Для визначення усереднених значень рівня фрагментованості файлів проведено експериментальні дослідження файлових систем лабораторних комп'ютерів.

Для аналізу рівня фрагментації файлів було досліджено файлові системи 30 робочих комп'ютерів із навчальних лабораторій Харківського Національного Університету ім. В.Н. Каразіна, на яких міститься системний том "С" із встановленою операційною системою Windows 7, том для зберігання іншої інформації "D". Також аналізувалися флеш-накопичувачі, умовно позначені, як том – "F". Попередня дефрагментація виконувалась 7–14 діб до отримання результатів. Результати отримано за допомогою вільного у використанні програмного забезпечення Auslogics Disk Defrag (www.auslogics.com) [49].

Результати роботи програми Auslogics Disk Defrag подано у вигляді файлів xml стилізованих у таблиці із вказаними даними для кожного фрагментованого файлу: кількість фрагментів – *Fragments*, індекси кластерів у яких розміщений файл – *Clusters*, розмір файлу – *Size*, результат дефрагментації – *Result*, ім'я та тип файлу – *File Name*. Приклад файлу xml як результат роботи програми зображено на рисунку 3.6.

Disk Defragmentation Details					
Fragments	Clusters	Size	Result	File Name	
2	4953 / 4960	26,44 KB	OK	C:\Program Files (x86)\360\Total Sec	
3	4960 / 4965	20,00 KB	OK	C:\ProgramData\USOShared\Logs\No	
4	44724 / 44767	169,35 KB	OK	C:\Users\students\AppData\Local\Par	
3	4965 / 4971	22,28 KB	OK	C:\Users\students\AppData\Local\Par	
2	4971 / 4979	29,55 KB	OK	C:\Users\students\AppData\Local\Par	
2	22 / 24	5,11 KB	OK	C:\Windows\System32\SleepStudy\w	
3	4979 / 4987	32,00 KB	OK	C:\Windows\Logs\WindowsUpdate\W	
4	82513 / 82556	169,35 KB	OK	C:\Users\students\AppData\Local\Par	
3	4987 / 4993	22,28 KB	OK	C:\Users\students\AppData\Local\Par	

Рис. 3.6 Приклад подання даних програмою Auslogics Disk Defrag

Для аналізу результатів роботи програми Auslogics Disk Defrag було створено програмну реалізацію автоматичного підрахунку необхідних даних, а саме наступних:

- залежність кількості файлів від рівня фрагментації кожного файлу;
- залежність кількості файлів від питомого (нормованого) рівня фрагментації;
- залежність кількості фрагментів від типу файлів (.log, .doc, .exe і т.п.).

Репозиторій із кодом програми для парсингу даних та результатами аналізу містяться за web-адресом “<https://github.com/ShekhaninKyryl/ausdiskdefrag-parser>”

Залежність кількості файлів від рівня фрагментації кожного файлу вказує на те скільки файлів у файловій системі містить певну кількість фрагментів. Цей результат надає можливість визначити оптимальні параметри для методів приховування інформації: кількість покриваючих файлів, максимально допустимий розмір стеганограми, щоб вихідний рівень фрагментованості покриваючих файлів не перевищував допустимий рівень фрагментованості.

Залежність кількості файлів від питомого (нормованого) рівня фрагментації вказує на те скільки файлів із зазначеним рівнем питомої фрагментації. Цей результат визначить на скільки залежить рівень фрагментації від розміру файлу. Що надає змогу оцінити оптимальний розмір покриваючих файлів для використання методу приховування інформації.

Залежність кількості фрагментів від типу файлу надає можливість аналізувати на скільки рівень фрагментації залежить від типу файлу, тобто те як даний файл використовується файловою системою. Що дає змогу зробити висновок щодо бажаних типів покриваючих файлів у залежності від виду використання файлової системи.

Далі надаються результати аналізу файлових систем, томів "C" та "D", за трьома показниками зазначеними вище.

Таблиця 3.5

Сумарна кількість файлів із відповідним рівнем фрагментації на "C"

Фрагментів	Кількість файлів	Фрагментів	Кількість файлів
2	6223	19	8
3	2196	20	8
4	985	21	6
5	525	22	10
6	278	23	3
7	152	24	4
8	102	25	4
9	79	26	7
10	54	27	7
11	49	28	2
12	51	29	4
13	38	30	6
14	26	31	7
15	24	32	5
16	21	33	2
17	25	34	3
18	9

Аналізуючи системний том "C" (таблиця 3.5), можна стверджувати, що фрагментація є неминучим результатом роботи операційної системи. Операційна система у поточному часі звертається до файлової системи задля запису та зчитування даних, що призводить до значної кількості фрагментованих файлів.

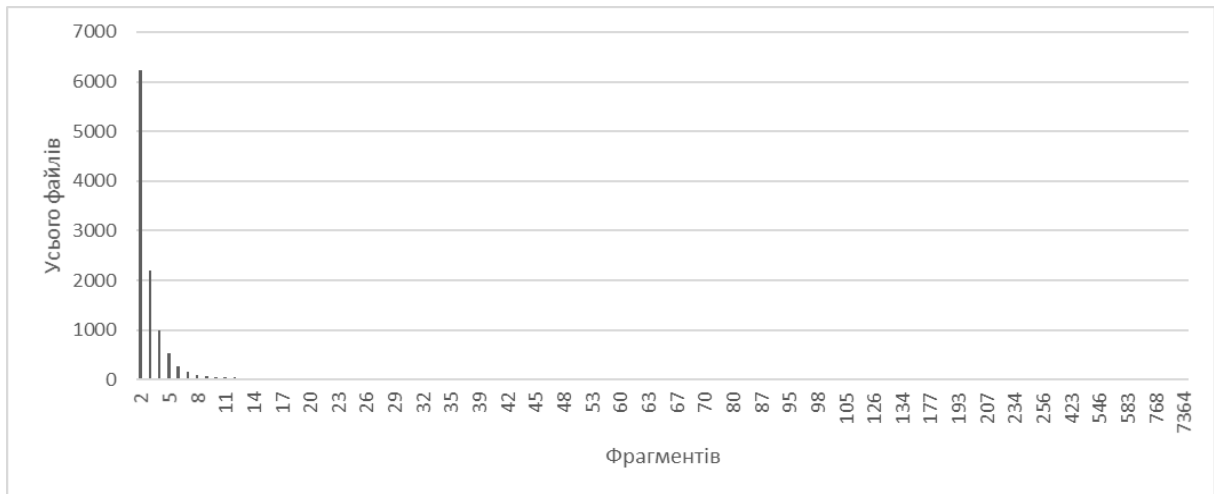


Рис. 3.7 Сумарна кількість файлів із відповідним рівнем фрагментації на системному томі "C"

Аналізуючи рисунок 3.7, на якому зображено відповідну гістограму, можна стверджувати, що фрагментація має зворотню залежність від кількості файлів. Тобто у файловій системі знаходиться значно більше файлів із низьким ніж із значним рівнем фрагментації.

Далі проаналізуємо середню кількість файлів із відповідним рівнем фрагментації для цього нормуємо кількість фрагментованих файлів із кожним рівнем фрагментації до загальної кількості фрагментованих файлів. Результати занесено до таблиці 3.6 та відповідній гістограмі зображеній на рисунку 3.8.

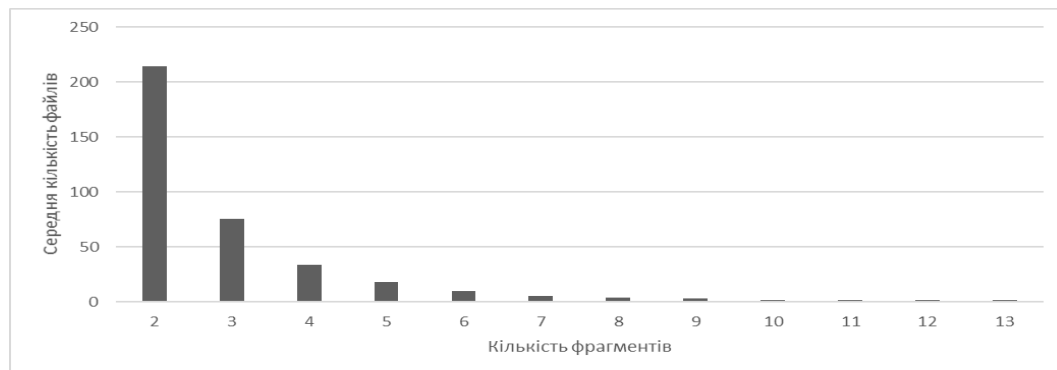


Рис. 3.8 Середня кількість файлів із відповідним рівнем фрагментації на системному диску "C"

Таблиця 3.6

**Середня кількість файлів із відповідним рівнем фрагментації на
системному диску "С"**

Фрагментів	Кількість файлів	Фрагментів	Кількість файлів
2	214,5862069	8	3,517241379
3	75,72413793	9	2,724137931
4	33,96551724	10	1,862068966
5	18,10344828	11	1,689655172
6	9,586206897	12	1,75862069
7	5,24137931	13	1,310344828

Аналізуючи середню кількість файлів із відповідним рівнем фрагментації можна стверджувати, що рівень фрагментації системного тому "С", такий що, є хоча б один файл із рівнем фрагментації 13. Що дозволяє використовувати модифікований метод приховування інформації із одним покриваючим файлом для приховування інформації розміром не більше 13 біт, без ризику детектування [49].

Питомий (нормований) рівень фрагментації– це рівень фрагментації на 1 кластер файлу, яку визначає формула 3.10. Розмір файлу розраховано за кількістю зайнятого місця у кластерах, при чому з округленням до більшого цілого (так як за специфікацією файлової системи FAT, частково заповнений кластер усе одно вважається таким, що зайнятий файлом, і даний кластер виключається із множини вільних кластерів [49].

Далі проаналізовано залежність кількості файлів від питомого рівня фрагментації кожного файлу, результати занесені до таблиці 3.7.

Аналіз загальної кількості файлів від питомого рівня фрагментації вказує на те що, у файловій системі найбільше зустрічаються файли із незначною фрагментацією ($\|\varphi(A)\| < 0.1$), або із 100% питомою фрагментацією. Але найчастіше такі файли розміром у 2 кластери та з рівнем фрагментації 2. Залежність рівня фрагментації від розміру файлу прослідковується, та має зворотню залежність. Але зустрічаються й аномальні значення, як наприклад 852 файли із нормованим рівнем фрагментації 0.66.

Таблиця 3.7

**Загальна кількість файлів із відповідним рівнем питомої (нормованої)
фрагментації на системному томі "С"**

Питома фрагментація	Кількість файлів	Питома фрагментація	Кількість файлів
0	1801	0,44	13
0,02	977	0,46	1
0,04	891	0,5	609
0,06	1041	0,52	2
0,08	401	0,54	8
0,1	339	0,56	20
0,12	318	0,58	1
0,14	191	0,6	69
0,16	176	0,62	12
0,18	159	0,66	852
0,2	177	0,68	4
0,22	111	0,7	9
0,24	208	0,74	47
0,26	50	0,78	2
0,28	162	0,8	31
0,3	37	0,82	10
0,32	265	0,84	10
0,34	7	0,86	3
0,36	92	0,9	3
0,38	2	0,92	1
0,4	360	0,94	1
0,42	87	1	1391

Та на основі таблиці 3.7 побудовано гістограму що зазначена на рисунку 3.9 – Загальна кількість файлів із відповідним рівнем питомої фрагментації на системному томі "С".

Наступним кроком є аналіз середньої кількості файлів із відповідними нормованими рівнями фрагментованості. Результати зображені на рисунку 3.10 та таблиці 3.8.

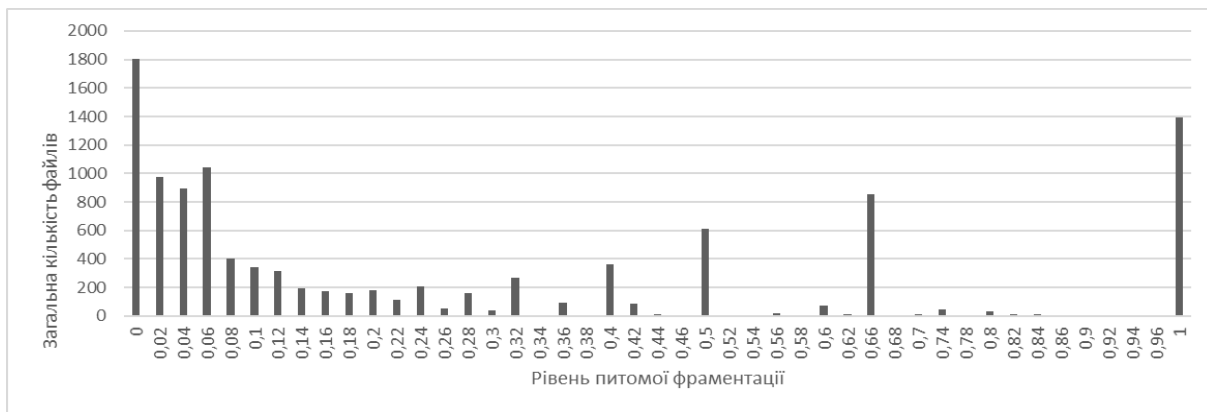


Рис. 3.9 Загальна кількість файлів із відповідним рівнем питомої фрагментації на системному томі "С"

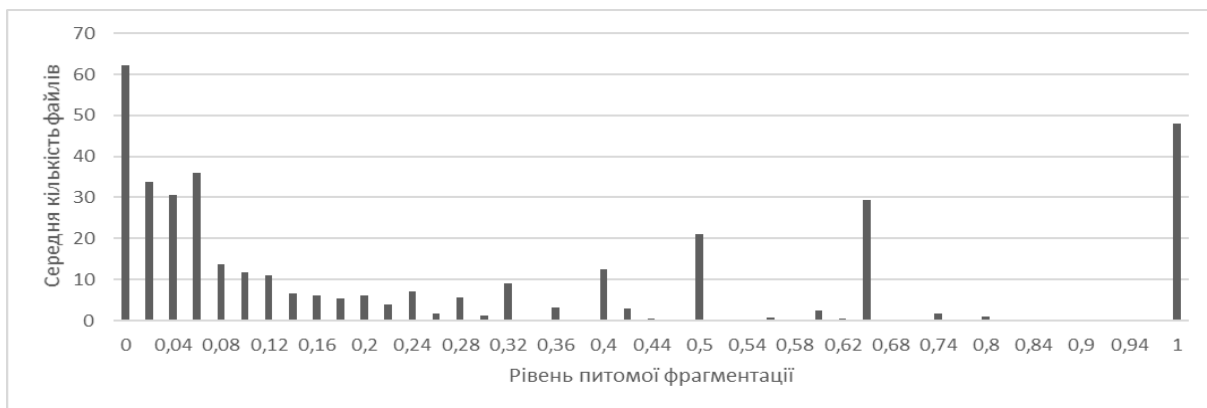


Рис. 3.10 Середня кількість файлів із нормованим рівнем фрагментації на "С"

Роблячи висновок із середніх значень кількості файлів із відповідним нормованим рівнем фрагментації, можна стверджувати що у системному томі "С" більша частка фрагментованих файлів має рівень фрагментації до 40%. Але зустрічаються й файли із 100% рівнем фрагментації. Що наводить на висновок, що для більшої захищеності прихованого повідомлення не можна приховувати максимальне допустиме за розміром повідомлення, а треба приховувати таке повідомлення щоб фрагментованість покриваючих файлів була до 40%.

Таблиця 3.8

Середня кількість файлів із відповідним рівнем питомої фрагментації на системному томі "С"

Питома фрагментація	Кількість файлів	Питома фрагментація	Кількість файлів
0	62,10344828	0,44	0,448275862
0,02	33,68965517	0,46	0,034482759
0,04	30,72413793	0,5	21
0,06	35,89655172	0,52	0,068965517
0,08	13,82758621	0,54	0,275862069
0,1	11,68965517	0,56	0,689655172
0,12	10,96551724	0,58	0,034482759
0,14	6,586206897	0,6	2,379310345
0,16	6,068965517	0,62	0,413793103
0,18	5,482758621	0,66	29,37931034
0,2	6,103448276	0,68	0,137931034
0,22	3,827586207	0,7	0,310344828
0,24	7,172413793	0,74	1,620689655
0,26	1,724137931	0,78	0,068965517
0,28	5,586206897	0,8	1,068965517
0,3	1,275862069	0,82	0,344827586
0,32	9,137931034	0,84	0,344827586
0,34	0,24137931	0,86	0,103448276
0,36	3,172413793	0,9	0,103448276
0,38	0,068965517	0,92	0,034482759
0,4	12,4137931	0,94	0,034482759
0,42	3	1	47,96551724

Наступним етапом є аналіз залежності рівня фрагментованості від типу фрагментованого файлу. Таким чином це дозволить виявити файли та розширення файлів із більшим рівнем фрагментованості, що надає відповідь, у яких файлах більш можливе приховання повідомлення. Результати вилучені з логів, що стосуються системного тому "С", які зібрані програмою Auslogics Disk Defrag. Результати занесені до таблиці 3.9 [49].

Таблиця 3.9

**Перші двадцять типів файлів із найбільшим рівнем фрагментації у системному
томі "С"**

Тип файлу	Рівень фрагментація		
	Мінімальний	Середній	Максимальний
79	11	255,7143	768
ZIP	2	94,07143	670
XML	2	37,84375	7364
VDM	2	36,46154	578
7Z	2	25,6	103
LOG	2	20,57759	3240
CACHE	2	19,05882	177
BIN	2	13,625	539
83	2	10,5	25
EDB	2	9,136364	59
INDEX	2	8,822581	196
CAB	2	8,6875	39
EXE	2	8,052632	130
DAT	2	7,923469	423
INI	2	7,342105	31
DLL	2	6,716216	281
PDB	2	6,571429	26
1	2	6,096774	19
7	2	6	8
TTF	2	5,933333	32

Данні із таблиці надають можливість оцінити двадцять найфрагментованіших фалів за мінімальним, середнім та максимальними рівнями фрагментації. Загалом у таблиці представлені такі розширення файлів:

– .79 – даний тип файлу є системним файлом, різні програмні забезпечення можуть по різному використовувати даний файл, найчастіше файл із розширення “.79” використовується операційною системою Windows10 та веб-браузером Google Chrome. Так як досліджувані комп’ютери мали інстальовану операційну систему як Windows7 то можна стверджувати, що файли “.79” створювалися та використовувалися веб-браузером, що відповідає закономірності – рівень фрагментації файлів збільшується при завантаженні даних через мережу інтернет;

– *.zip* – це широко розповсюджений тип файлу для збереження архівованих даних, у межах одного зжатого файлу. Це дозволяє зменшити розмір архівованих даних та спрощує передачу. ZIP-файли працюють так само як і директорії з точки зору інтерфейсу користувача. Через свою простоту у використанні та зменшені розміру даних цей тип файлу широко використовується при передачі даних через мережу інтернет. Тобто даний тип файлу має значний рівень фрагментації через те що дані файли були завантажені через мережу інтернет;

– *.xml* – дане розширення мають файли які містять спеціальну розмітку для збереження та представлення даних, які найчастіше використовуються у веб ресурсах. Коли користувач відкриває сторінку у браузері то браузер завантажує файли для стилізації даних (*.css*, *.js*, *.html*) та самі дані (*.xml*) Для оптимізації запитів до веб ресурсу, веб-браузери здебільшого зберігають такі *.xml* файли локально у файловій системі. Тобто рівень фрагментації файлів *.xml* значний через те що виконується завантаження файлів через мережу інтернет;

– *.vdm* – це системний тип файлу який використовується різними операційними системами: Mac, Windows, Linux, Android та iOS. Даний тип файлу був розроблений компанією Greenview Data та слугує як макрофайл для редагування великих за розміром (до 100Гб) файлів. Отже рівень фрагментації файлів *.vdm* значний через те що ці файли у поточному часі використовуються операційною системою;

– *.7z* – дане розширення мають файли які містять у собі заархівовані файли, подібно як і *.zip* файли, але *.7z* дозволяє досягти на 4-25% кращого ступеня стискання, а також *.7z* файли можуть бути багатопотоковими, що у свою чергу дозволяє одночасно додавати, змінювати чи видаляти файли із архіву через що такі файли мають значний рівень фрагментації;

– *.log* – це загальне розширення типу файлу, яке може бути відкрито різними програмами. Лог-файли створюються для подальшого моніторингу отже такі файли змінюються та дописуються увесь час поки працює програма, у тому числі й

процес операційної системи. Саме через такий постійний доступ до лог-файлів, їх рівень фрагментації збільшується.

- *.cache* – даний тип файлу зберігає тимчасово використану інформацію у кеши диска для програм, найчастіше даний тип файлу використовується веб-браузерами, програмами редагування текстових документів та зображень, а також іншими програмами для збільшення швидкодії роботи програм. Файли *.cache* не повинні відкриватися вручну користувачем, що свідчить що рівень фрагментації таких файлів здебільшого досягається через завантаження даних через інтернет та багатопотокового використання різними програмами.

- *.bin* – файли *.bin* зберігають інформацію у двійковому форматі. Такий формат даних відрізняється від текстових файлів, які можуть легко редагувати. Двійкові файли можуть бути створені різними програмами та як правило, користувач не може вірно редагувати такі файли. Як приклад файли образу диску містять двійкові файли, також такі файли містять різні інсталювані програми, що свідчить що рівень фрагментації виникає при встановленні програми, що означає що інсталяція програм, виконується у багатопотоковому режимі.

- *.83 – Pro/ENGINEER* є основною програмою яка використовує *Creo Elements/Pro Versioned Data* файли, найчастіше такі файли зустрічаються у операційних системах Windows10 або у системах яка використовує веб-браузер Google Chrome. Отже рівень фракції у таких файлах збільшений через використання їх у роботі інтернету.

- *.edb* – файли із розширення *.edb* містять бази даних вхідних повідомлень електронної пошти, які створені сервером Microsoft Exchange. Основні дані зберігаються у структурованому форматі b-дерева та містять як основні так і дочірні вузли. Файли *.edb* дозволяють зберігати як оброблену, так й збережену інформацію поза SMTP. Основними файлами, що використовуються сервером для цієї цілі, є файли *Priv1.Edb* та *Pub1.Edb*, тоді як усі SMTP-повідомлення мають окремі файли *STM*.

– *.index* – файли *.index* використовуються для зберігання індексів ключових слів для функції Eclipse Help Index. Eclipse – це багатоплатформене інтегроване середовище розробки (IDE), яка підтримує багато мов програмування. Файли *.index* можна знайти на персональних комп'ютерах із операційними системами Mac, Windows або Linux. Eclipse використовується здебільшого для програмування на таких мовах програмування: JAVA, C, C ++, Python, PHP та COBOL, але також може бути використано для розробки програмного забезпечення й на інших мовах програмування.

– *.cab* – тип файлів *.cab* відноситься до файлів архіву й часто використовуються в рамках різних програм-інсталяторів від компанії Microsoft. Часто зустрічаються у дистрибутивах Microsoft Windows, можуть бути використані різними програмними комплексами, для прикладу MS Office та іншими. В основному звичайний користувач комп'ютера не виконує архівування із використанням подібного формату. Стискання даних використовує алгоритми ZIP, а також Quantum та LZX. Для того щоб відкрити файл *.cab* можна використати сучасні архіватори як приклад WinRar та WinZip. У більш рідких випадках файл *.cab* може бути використано інсталяторами, побудованими на базі InstallShield.

– *.exe* – розширення виконуючих файлів, яке використовується у операційних системах DOS, Windows, Symbian OS, OS/2 та в деяких інших, які відповідають ряду форматів. Окрім об'єктного коду, файли *.exe* можуть містити різні метадані, такі як допоміжні ресурси, цифровий підпис, інформацію про розробника і так далі.

– *.dat* – це загальний формат зберігання даних різного програмного забезпечення. Як правило, файл *.dat* можна відкрити лише програмою, яка створила даний файл. Інформація може зберігатися як у текстовому форматі так і в двійковому вигляді. Текстові файли *.dat* можна продивитись будь-яким текстовим редактором. Приклад програм, які створюють та використовують файли *.dat*: Microsoft Visual

Studio, Corel WordPerfect, Nero ShowTime, Nullsoft Winamp, SoftVelocity Clarion, Ontrack EasyRecovery, Runtime GetDataBack, програмне забезпечення MapInfo.

– *.ini* – це файл конфігурації, який містить дані для налаштування для Microsoft Windows, Windows NT та деяких інших програм. Файли *.ini* з'явилися із перших версій Windows. У версії Windows 1.01 цей був тільки файл WIN.INI. У версії Windows 3.0 додали файл SYSTEM.INI. Потім кількість таких файлів збільшувалася швидко та безконтрольно. Файли *.ini* – це звичайні текстові файли, які можна редагувати та продивлятися використовуючи будь який текстовий редактор.

– *.dll* – ці файли *.dll* також відомі як файли Dynamic Link Library, та зміст файлів, які задіяні із файлами *.dll*, є набором компільованих ресурсів, таких як директив, процедур та бібліотек драйверів, які потребують вбудовані програми Windows, та сторонні програмні забезпечення для операційної системи Microsoft Windows. Ці файли *.dll* дозволяють програмному забезпеченню для Windows виконуватися із використанням загальних ресурсів, що у цілому звільняє від необхідності множинного створювання одних й тих файлів окремо під кожне програмне забезпечення. Більшість цих файлів поставляється із операційною системою Windows, деякі файли надаються із драйверами і лише специфічні файли *.dll* інколи встановлюються разом із певним програмним забезпеченням. Деякі із цих файлів *.dll* полегшують зв'язки між операційною системою Windows із відповідними вбудованими чи стороннім програмними забезпеченнями та ресурсами драйверів, які необхідні для зовнішніх чи внутрішніх пристроїв введення/виведення інформації.

– *.pdb* – формат файлів *.pdb* це стандартний формат баз даних, який використовується на певних операційних системах. Даний формат даних дозволяє зберігати різні типи даних – електронні книги, документи, зображення, карти, таблиці й тому подібне, але у чітко структурованому вигляді. Дуже важливо знати якою програмою були створені файли *.pdb* адже тільки цією програмою й можна відкрити такі файли.

– *.l* – даний формат файлу має пропрієтарну структур, тобто таку структуру що не має публічної специфікації. Відомо що файл *.l* створено компанією Microsoft та використовується у операційних системах Windows.

– *.7* – повне ім'я файлів, які використовують розширення *.7*: The 7th Guest. Специфікація The 7th Guest була створена Trilobyte, Inc.. Файли із розширенням *.7* можуть бути використані програмами, які використовують операційні системи iOS та Windows. Формат файлу *.7* наряду із іншими форматами файлів, які відносяться до категорії розваг. Користувачам рекомендується використовувати The 7th Guest програмне забезпечення для керування *.7* файлами, хоч й інші програми також можуть використовувати ці типи файлів.

– *.tff* – файл *.tff* використовується для комп'ютерних шрифтів та є векторним форматом, який було розроблено компанією Apple у 1980 році. Подібні файли шрифтів, використовують у різних операційних системах. Інформацію, яку містить формат TTF, представлено в векторному вигляді, що дозволяє легко масштабувати шрифт, при цьому, якість залишається незмінною. Даний формат файлу володіє декількома версіями, у залежності від яких, визначається максимальне число символів що знаходяться у файлі (значення 2^8 , 2^{16} або 2^{32}). В основному, *.tff* файли використовуються для шрифтів "TrueType". Як відомо, шрифт TrueType є прямим конкурентом шрифту Type-1 від компанії Adobe, опис формату надає розробникам переваги високого ступеня контролю над відображенням тексту. В то час, як був створений файл *.tff*, інші формати шрифтів, фактично не змогли надати можливості подібного рівня. Розширення файлу *.tff*, включало в себе можливості настройки пікселів при різному розмірі самого шрифту. На теперішній день, беручи до уваги технології широко розповсюджених програв для відображення, керування пікселями не потребується. Системи Mac OS, а також Windows, мають вбудовані можливості для перегляду та використання даного типу файлу. Створити розширення *.tff*, можна використовуючи таке програмне забезпечення, як CorelDRAW Graphics Suite або OpenOffice Draw. Щоб відкрити файл, що має розширення *.tff*, використовується

утилітою FontCreator або Font Viewer від компанії Microsoft. В рамках операційної системи MacOS, переважає використання комплексного програмного забезпечення Font Book від Apple. Слід взяти до уваги, що хорошої альтернативи для Apple Font, вважається програмне забезпечення FontForge, яке може використовувати файли із розширенням *.ttf* навіть й на операційній системі Linux [40, 55, 56].

Проаналізувавши результати з таблиці 3.10 побудуємо для наочності гістограми, зображені на рисунку 3.11. Кожен тип файлу три рівні фрагментації: мінімальний рівень, середній рівень та максимальний рівень фрагментації. Гістограми упорядковані саме по середньому рівню фрагментації.

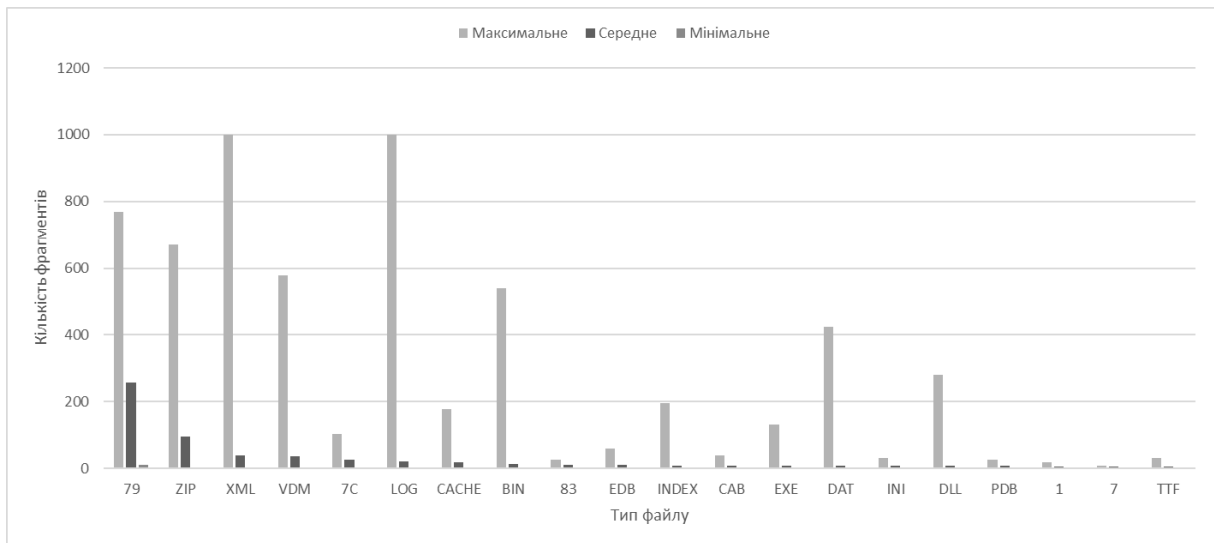


Рис. 3.11 Перші двадцять типів файлів із найбільшим рівнем фрагментації

Як видно із рисунку 3.11, можна стверджувати, що файли *.79* мають найбільший мінімальний рівень фрагментації, що означає, що такі файли мають завжди певний рівень фрагментації. Інші ж типи файлів мають лише рівень фрагментації як 2. Але аналізуючи середній рівень фрагментації, то усі типи файлів дозволяють приховати повідомлення, але файли із розширенням *.79*, *.zip*, *.xml*, *.vdm*, *.7z*, *.log*, *.cache*, *.bin*, *.83* мають середній рівень фрагментації більший за 10. Щодо максимального рівня

фрагментації, то необхідно виділити файли із розширенням *.xml*, *.log* які мають 7364 та 3240 рівні фрагментації відповідно.

Аналізуючи стандартні файлові розширення, можна визначити які операції із файлами приводять до більшої фрагментації:

- ZIP: файл містить архівовані дані у стислому вигляді. Такі данні у залежності від початкового розміру можуть мати досить значний час архівування. Таким чином із часом, наступний вільний кластер може стати зайнятим іншими даними, що призводить до збільшення рівня фрагментації. Також, зазвичай данні передаються через мережу у архівованому виді, а при завантажуванні даних через мережу інтернет фрагментація файлу є неминучим наслідком.

- XML: файл, що містить набір даних для програм, у тому числі браузерів (web-сторінок). Такі дані завантажуються разом із html-сторінкою, через мережу.

- LOG: файл, що містить згенеровану інформацію про результат роботи певних служб, сервісів програми або операційної системи. Такий файл генерується увесь час, що виконується програма. Тож, час запису у цей файл може бути великий, що збільшує ймовірність фрагментації [40, 55, 56].

Таким чином, до фрагментації неминуче призводить робота операційної системи. Файли без стандартизованого розширення (системні файли), є найбільш фрагментованими. Щодо стандартних розширень то можна стверджувати, що це є файли які:

1. записуються упродовж довгого часу, так що інші дані можуть зайняти наступний кластер (ZIP, LOG, CACHE);

2. завантажуються через мережу інтернет (XML, INI, DLL, ZIP).

Наступним етапом проаналізуємо наступний том “D”, даний том використовується для збереження файлів програм таких як допоміжні файли та основні інсталюючи файли. Будемо аналізувати загальну кількість файлів від рівня фрагментації та рівень фрагментацій від типу файлу, що дозволить зробити висновок які процеси призводять до збільшення рівня фрагментації на інформаційному диску

“D”. Результати по аналізу кількості файлів від рівня фрагментації заходяться у таблиці 3.10 та рисунку 3.12.

Таблиця 3.10

Загальна кількість файлів із відповідним рівнем фрагментації на інформаційному диску "D"

Фрагментів	Кількість файлів
2	193
3	89
4	56
5	40
6	26
7	20
8	9
9	5
10	6
11	7
12	2

Базуючись на таблиці 3.10 побудуємо гістограми, які зображені на рисунку 3.12 це дозволить більш наочно виявити оптимальні параметри для використання методу приховування даних.

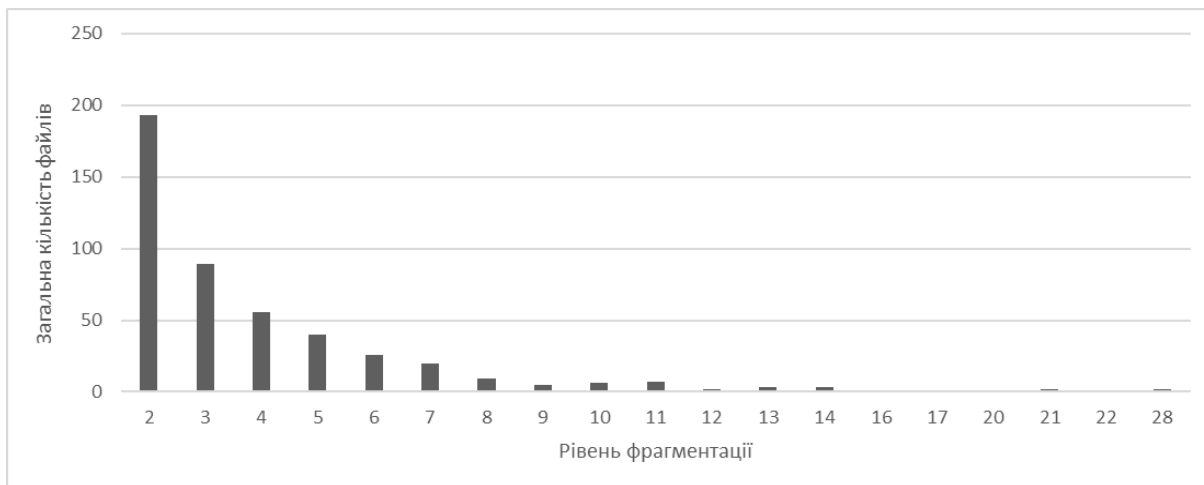


Рис. 3.12 Загальна кількість файлів із відповідним рівнем фрагментації на інформаційному диску "D"

Аналізуючи інформаційний том "D", можна стверджувати, що фрагментація є неминучим результатом роботи багатопотокового звернення до файлової системи. Здебільшого це досягається запитами користувача. Для прикладу, користувач одночасно може:

- запустити процес який створює лог-файли;
- завантажувати файли через мережу інтернет;
- виконувати редагування документів.

Одночасне виконання таких дій призводить до збільшення рівня фрагментації.

Загалом рівень фрагментації тому "D" має таку ж залежність як і системний том "C", але рівень фрагментації кожного файлу менший. Це пов'язано із тим що фрагментацію у том "D" вносить здебільшого користувач, та деякі програми, але не операційна система.

Наступним етапом є аналіз рівнів фрагментації у залежності від типу файлів, це дозволяє виявити особливості роботи файлової системи у інформаційному диску "D". Результати дослідження занесені до таблиці 3.11.

Аналізуючи результат можна стверджувати, що за відсутності впливу операційної системи на том "D", рівень фрагментації буде мінімальним. Але, є певний тип файлів що мають завищений рівень фрагментації:

- *.db* – це файл зі спеціальною структурою, для зберігання бази даних. Робота бази даних зазвичай виконується над одним таким файлом протягом довгого часу, що безпосередньо призводить до збільшення фрагментації.

- *.idb* – файл, створюваний IDA (The Interactive Disassembler) – програмою яка дізасемблює (розкладає виконуючи файли у файли коду) виконуючи файли (*.exe*) у програмний код нижнього рівня на мові програмування Assembler. Зберігає дізасембльовані дані у стислому форматі. Використовуються у процесі дослідження виконуючих файлів, часто для розкриття потенційно шкідливих для безпеки та системи програм. При відкритті файлів IDB в IDA дані разорхівовуються у набор файлів. Після закриття бази даних файли знов стискаються у файл IDB. Це дає змогу

використовувати більш швидку продуктивність у той час, коли база даних відкрита, та менше займати місця на дисковому просторі, коли база даних закрита.

- *.pdb* – дане розширення описано вище лише можна зробити висновок що даний файл розташовано як і на системному диску “C” так і на інформативному диску “D”.

- *.vdi* – файл *.vdi* – це образ віртуального жорсткого диску, який використовується програмним забезпеченням віртуалізації Oracle VM VirtualBox. Даний файл представляє собою жорсткий диск віртуальної машини – віртуального комп’ютера, створеного у VirtualBox та дозволяє встановлювати різні операційні системи та сумісні з ними програмні забезпечення. В налаштуваннях VirtualBox користувачі можуть обирати розмір віртуального диска та каталог розміщення файлу VDI. Файл являє собою образ CD-диска, створеного за допомогою Virtuo CD Manager, та містить точну копію диска. При монтуванні у віртуальний привод образ дозволяє працювати з даними, як би вони були записані на фізичний носій. Файли образів в форматі VDI можуть бути відкриті використовуючи сумісні програмні забезпечення для емуляції, наприклад, UltraISO.

- *.ipch* – файл розробки, використовує Microsoft Visual C++, інтегрована середовище розробки, що використовується для програмування програмного забезпечення мовою C++ у Windows. Містить попередньо скомпільовану головну інформацію, що використовується Intellisense – модулем підказок при кодуванні, необхідний для підказок коду, документації і для автоматичної комплектації коду під час друку програмістом програмного коду. Для проектів Visual C++ файл формату *.ipch* зберігаються у проектній піддиректорії (*\ipch*). Visual C++ є частиною середовища розробки програмного забезпечення Visual Studio від Microsoft.

- *.pch* – вихідний файл, створений MSC Nastran, програма структурного аналізу, яка дозволяє перевіряти цілісність різних структур, таких як космічні прилади або завдання; містять різні типи введення даних у залежності від виконуючий карти (аналогічно до команди). Прикладом карти є XYPUNCH, яка буде створювати вихідні

дані X и Y-осей, для прикладу, величину у порівнянні з частотою. Зміст РСН-файлу, створеного із цією картою, буде складатися із двох типів даних. Перший тип буде схожий на файл .F06 , с формами режимів, статистичними зміщеннями и даними стресової точки сітки, але в альтернативному форматі із файлів F06. Другий тип виводу буде відображати величину у порівнянні із частотою, що допомагає відобразити криву відгуку в електронній таблиці.

- *.index* – даний тип файлу описано вище.
- *.ilk* – файл що використовується Microsoft Visual Studio – інтегровано.

Середою розробки для програмного забезпечення, що розроблюються для операційної системи Windows. Містить зв'язані дані для компільованого виконуючого файлу (як приклад, файли форматів *.exe* або *.dll*). Дозволяє процесам компіляції й зв'язуванню проектів виконуватись швидше, особливо у випадках, якщо проекти мають великий розмір. Інтерактивні зв'язуючи файли містять дані таким чином, щоб локалізована розробка змінювала лише необхідні оновлення до частин компільованих та зв'язаних файлів, в яких були зафіксовані зміни. Це дозволяє розробникам змінювати вихідний код та тестувати результати швидше, без необхідності перебудови усієї запущеної програми. Використання файлів ILK може уповільнити продуктивність процесу роботи програм та збільшити їх розмір. Тем не менше, розробники повинні вивчати ці показники продуктивності при побудові й розміщенні остаточної версії програми. Маленькі за розміром проекти розробки можуть не отримати користь від використання інкрементного зв'язування.

- *.dat* – даний тип файлу описано вище.
- *.undefined* – усі випадки появи файлів із розширенням *.undefined* мають відношення до помилок програмного забезпечення або можливої неправильної конфігурації. Само по собі розширення *.undefined* означає "невизначений" та є показником неспроможності якогось програмного забезпечення, що присвоїло таке розширення файлу, розпізнати окремий формат файлу та/або присвоїти коректне розширення. Розширення *.undefined* може присвоюватись мультимедійним файлам

подібним MP3 (замість необхідного у даному випадку розширення .mp3), завантаженими із деяких некоректний онлайн-сервісів на базі FFMPEG по перекодуванню аудіо та відео файлів. Такий випадок виникає при неправильній конфігурації або помилок програмного забезпечення. Якщо відомий необхідний формат файлу, то можна спробувати змінити розширення файлу вручну (у даному випадку, з *.undefined* на *.mp3*). Це також може бути результатом помилкової обробки MIME-типів веб-серверів, іншими словами, коли сервер по якимось причинам "не знає", який MIME-тип повинен використовуватися по відношенню до конкретного змісту файлу, представленому сервером. І як висновок файлу присвоюється розширення *.undefined*. Повідомлення о помилка щодо невизначеного розширення файлу може також виникати на окремих веб-сайтах на базі Joomla, якщо доступ до такого сайту виконується за допомогою однієї із ранніх версій Mozilla Firefox та при цьому робиться спроба завантажити на сервер файл (зображення або інше). Така поведінка пов'язана із помилкою у програмному забезпеченні Firefox, яка усунена у більш пізніх версіях браузера.

— *.suo* – файли SUO створюються Microsoft Visual Studio задля зберігання інформації о користувачах. Зібрані дані зберігаються у двійковому структурованому форматі. У Visual Studio SUO-файл створюється за допомогою WriteUserOptions. Для зберігання змін у файлі використовується опція SaveUserOptions. Завантаження вже створеного SUO-файлу може бути виконана із використанням LoadUserOptions. Технічні відомості о файлах SUO. Як вже зазначено вище, файли SUO зберігають користувацькі дані. Ця інформація зберігається у потоках. Ім'я потоку також є ключом, який використовується для ідентифікації інформації, що зберігається в SUO-файлі. Файли SUO залежать від встановлення на певному персональному комп'ютері та версії Visual Studio, в які вони були створені. Тому розповсюджувати такі файли між іншими розробниками немає сенсу, навіть у тому випадку, якщо вони використовують Microsoft Visual Studio. Окрім того, файли SUO змінюються кожен раз, коли користувач використовує відповідну версію Visual Studio.

– *.obj* – даний формат файлів відповідає за опис геометрії, даний файл розроблено у компанії Wavefront Technologies для анімаційного пакета Advanced Visualizer. Формат файлу є відкритим та був прийнятий іншими розробниками програмного забезпечення 3D-графіки. Він може бути експортований та/або імпортований у інші програмні забезпечення по 3D-моделюванню: e-Frontier's Poser, Maya, XSI, Blender, MeshLab, Misfit Model 3D, 3D Studio Max та Rhinoceros 3D, Hexagon, CATIA, Newtek Lightwave, Art of Illusion, milkshape 3d, Modo, Cinema 4D, Zanoza Modeller, ПК ЛІРА, Mineways та тому подібне. У більшій мірі *.obj* загальноприйнятий формат. Формат файлів OBJ — це простий формат даних, який містить тільки 3D геометрію, а саме, позицію кожної вершини, та зв'язок координат текстури із вершинами, нормаль для кожної вершини, а також параметри, які створюють полігони.

– *.tlog* – тип файлу TLOG у першу чергу пов'язаний із ArduCopter від компанії ArduPilot. Файли даних TLOG містять зміст даних журналу телеметрії ArduCopter. TLOG конвертується у журнал телеметрії. ArduCopter – це багатокomпонентна платформа для розробки безпілотних летальних апаратів. Arducopter керується пілотом ArduPilot Auto із відкритим вихідним кодом. Цей програмний комплекс був розроблений командою та спільнотою ArduPilot. Для того щоб відкрити та редагувати файл із розширенням TLOG, користувачу знадобиться сумісне програмне забезпечення, таке як ArduCopter від компанії ArduPilot, щоб відкрити файл TLOG.

– *.resx* – файл ресурсів, що використовується програмним забезпеченням, розробленим із використанням .NET Framework Microsoft. Зберігає об'єкти та строки для програмного забезпечення у форматі XML. Може містити як тестову інформацію, так і двійкові дані, які кодуються як текст у тегах XML. Файли RESX зазвичай використовують для зберігання налаштування програмного забезпечення й інших програмних ресурсів, таких як зображень. Якщо файли будуть реалізовані вірно, вони можуть бути замінені на інші пакети ресурсів, без повторної компіляції програмного

забезпечення. Даний формат файлів має значні переваги при використанні у локалізації програм у різних середовищах. Навіть якщо файли RESX можуть містити двійкові дані, вони усе одно можуть бути відкриті із допомогою будь якого текстового редактора та проаналізовані із допомогою стандартного аналізатора XML файлів. Двійкові об'єкти кодуються у середині тегів “lt” та “gt”, теги, які містять властивості для програмного забезпечення та типи MIME. Деякі програмні забезпечення може декодувати двійкові значення файлів у текстовий вид.

- .exe – даний формат файлу описано вище.
- .cache – даний формат файлу описано вище.

Таблиця 3.11

Типи файлів із відповідним рівнем фрагментації (мінімальний, середній, максимальний) у інформаційному диску "D"

Тип файлу	Рівень фрагментація		
	Мінімальний	Середній	Максимальний
DB	2	10,71875	28
IDB	2	7,565217	16
PDB	2	5,272727	13
VDI	2	4,5	7
IPCH	2	3,966667	7
PCH	2	3,909091	6
INDEX	2	3,842105	22
ILK	2	3,7	10
DAT	2	3,315789	5
UNDEFINED	2	2,970588	14
SUO	2	2,833333	5
OBJ	2	2,611111	5
TLOG	2	2,388889	3
RESX	2	2,1	3
EXE	2	2	2
CACHE	2	2	2

Також необхідно проаналізувати флеш-накопичувачі, так як саме приховання інформації у структуру файлових систем на флеш-накопичувачах дозволяє робити

приховану передачу інформації із використанням методів приховування інформації шляхом перемішування кластерів покриваючих файлів [18, 40, 55, 56].

Аналізуючи флеш-накопичувач – "F", можна стверджувати, що фрагментація подібних носіїв інформації є мінімальною, це пов'язано із режимом використання флеш-накопичувачів. Здебільшого вони використовуються лише для зберігання на них даних, Таким чином не рекомендується використовувати флеш-накопичувача для прихованої передачі інформації. Але вони можуть бути використані, коли захищеність прихованого повідомлення від детектування не є таким значущим, наприклад для приховання аналогу цифрового водяного знаку для підтвердження валідності файлів на збережених на флеш-накопичувачі [28, 33, 57].

Для подальшого аналізу файлової системи, проведемо статистичний аналіз рівня фрагментації. А саме, визначимо закон розподілу рівня фрагментації. Проаналізувавши дані з таблиць та рисунків щодо залежності кількості файлів від рівня фрагментації отримаємо розподіл ймовірності для перших значень. Тобто з якою ймовірністю файл буде містити саме таку кількість фрагментів [36, 49, 54, 58,]. Проаналізувавши гістограми було висунуто гіпотезу, що розподіл рівня фрагментації підпорядковується закону розподілу Парето, формула 3.12 та рисунок 3.13.

$$f_X(x) = \begin{cases} \frac{kx_m^k}{x^{k+1}}, & x \geq x_m \\ 0, & x < x_m \end{cases} \quad (3.12)$$

де k та x_m – параметри розподілу Парето. Параметр k впливає на щільність розподілу зображену на рисунку 3.13.

Проаналізувавши результати залежності рівня фрагментації та кількості файлів, отримано данні фактичної та очікуваної ймовірності, при параметрах $k = 1$ та $x_m = 2$, та занесемо ці данні до таблиці 3.12.

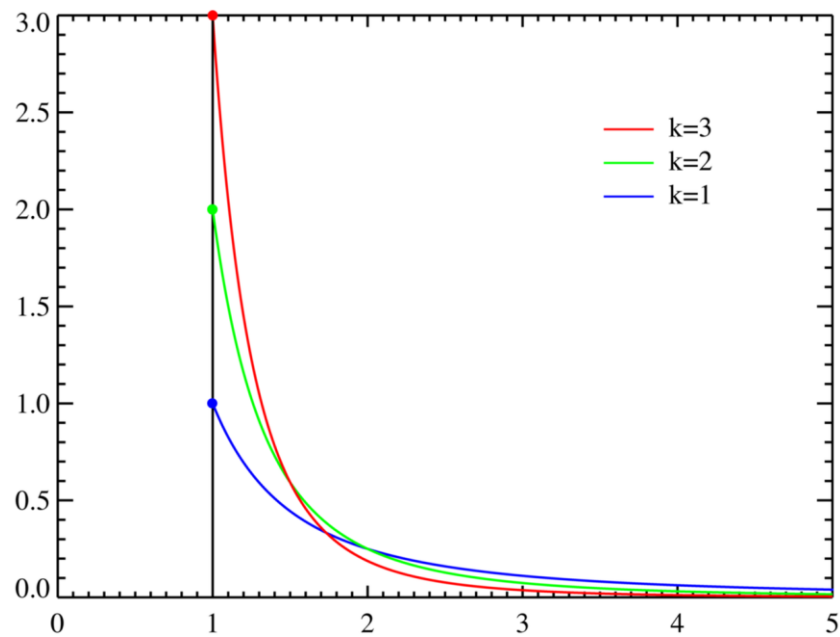


Рис. 3.13 Щільність ймовірності розподілу Парето

Таблиця 3.12

Фактичні та очікувані значення рівнів фрагментації

Рівень фрагментації	Фактичні значення	Очікувані значення	χ^2 Пірсона (0,115357)
2	0,584703561	0,5	0,014349
3	0,206332801	0,222222222	0,001136
4	0,092549093	0,125	0,008424
5	0,049328197	0,08	0,011759
6	0,026120455	0,055555556	0,015596
7	0,014281687	0,040816327	0,017250
8	0,009583764	0,03125	0,015022
9	0,007422719	0,024691358	0,012077
10	0,005073757	0,02	0,011140
11	0,004603965	0,016528926	0,008603

Перевіримо гіпотезу за допомогою критеріїв згоди:

1. критерія згоди Пірсона;
2. критерій згоди Колмогорова.

При використанні критерія згоди Пірсона збіжність деякого теоретичного розподілу статистичним (емпіричним) даним перевіряють шляхом оцінки міри розбіжності, у якості якої застосовують суму квадратів відхилень. Пірсоном доведено,

що при збільшенні вибірки цей закон наближається до закону «розподілу χ^2 ». За даними таблиці 3.5, кількість розглянутих файлів, тобто вибірка становить 367, що є більше ніж мінімальне рекомендоване значення. Таким чином маємо міру розбіжності теоретичного та емпіричного розподілів зазначену у формулі 3.13:

$$\chi_n^2 = \sum_{i=1}^n \frac{(O_i - E_i)^2}{E_i} \quad (3.13)$$

де:

- n – загальна кількість значень;
- i – порядковий номер значення;
- O_i – отримане практичне значення;
- E_i – отримане теоретичне значення;

Розподіл χ^2 залежить від числа ступенів свободи, яке дорівнює $d = n - 1$. Для розподілу χ^2 складено спеціальні таблиці, значення яких залежить від числа ступенів свободи та ймовірності того, що величина яка розподілена по закону χ^2 , буде вищою за це значення. Тобто, ймовірність p визначає те, що міра розбіжності теоретичного та статистичного розподілів буде не менше, ніж фактично розраховане за даними спостережень значення χ^2 . Якщо табличне значення χ^2 менше за розраховане, тоді статистичні дані слід вважати такими, що протирічать теоретичним та слід відкинути гіпотезу. Якщо табличне значення χ^2 більше за розраховане, тоді статистичні дані слід вважати такими, що відповідають теоретичним та слід не відкидати гіпотезу.

Вибірка складається з 10 інтервалів, отже ступінь свободи дорівнює 9. Знайдено критичне значення з таблиці значень χ^2 для $d = 9$, та $p = 0,95$, це значення дорівнює $\chi^2 = 3,32511$. Далі розрахуємо значення χ^2 за даними з таблиці 3.13, це значення дорівнює $\chi^2 = 0,115357$. Порівнявши ці значення, можна стверджувати, що гіпотеза, стосовно розподілу фрагментації за законом розподілу Парето не відкидається [39, 49].

Оцінюючи ступені згоди теоретичного та статистичного розподілів за критерієм Колмогорова необхідно визначити максимальне значення різниці між статистичною функцією розподілу та відповідною теоретичною функцією розподілу, формула 3.14.

$$D = \max |F^*(a) - F(a)| \quad (3.14)$$

Далі необхідно визначити межу критерію Колмогорова λ , для цього використаємо формулу 3.15 та таблицю 3.13 із результатами максимального значення різниці між фактичною функцією розподілу та теоретичною [59, 60].

$$D\sqrt{N} \leq \lambda \quad (3.15)$$

де:

- D – максимальне значення різниці між функцією розподілу та статистичними даними;
- N – число спостережень, у нашому випадку – загальна кількість файлів (367 спостережень);
- λ – значення критерію Колмогорова.

Таким чином $\lambda \leq 0,084 * \sqrt{376} = 1,62$. Далі за спеціальною таблицею критерію Колмогорова визначимо ймовірність $p(\lambda)$ того, що максимальна розбіжність між статистичним розподілом і теоретичним буде не менша ніж та, що фактично спостерігається. Якщо ймовірність $p(\lambda)$ мала, тоді гіпотезу щодо розподілу випадкової величини відкидають як неправдоподібну. Якщо значення $p(\lambda)$ велике, тоді гіпотезу вважають сумісною з результатами моделювання. У нашому випадку $p(1,62) = 0,012$, тобто ймовірність значно менша за 0,95. Як висновок, гіпотеза щодо розподілу рівнів фрагментації не відповідає розподілу Парето за критерієм згоди Колмогорова.

Таблиця 3.13

Максимальне значення різниці між фактичною функцією розподілу та теоретичною

Рівень фрагментації	Фактичні значення	Очікувані значення	Різниця значень
2	0,584703561	0,5	0,084704
3	0,206332801	0,222222222	0,015889
4	0,092549093	0,125	0,032451
5	0,049328197	0,08	0,030672
6	0,026120455	0,055555556	0,029435
7	0,014281687	0,040816327	0,026535
8	0,009583764	0,03125	0,021666
9	0,007422719	0,024691358	0,017269
10	0,005073757	0,02	0,014926
11	0,004603965	0,016528926	0,011925

Отже гіпотеза підтвердилась за критерієм Пірсона, але не за критерієм Колмогорова. Далі було проведено аналіз із меншим числом спостережень – 100 спостережень і за результатами оцінки критерію обидва критерію згоди виправдалися. Критерій згоди Пірсона прийняв гіпотезу із $p \geq 0.9$, а критерій згоди Колмогорова прийняв гіпотезу із $p \geq 0.54$. Таким чином, можна стверджувати, що:

- при незначній кількості спостережень (100 фрагментованих файлів), їх розподіл відповідає розподілу Парето.
- при великій кількості спостережень (>300), розподіл набуває експоненційного характеру [39, 49].

Далі визначимо, можливий розмір прихованого повідомлення, без збільшення фрагментації файлової системи, тобто можливий розмір повідомлення без ризику детектувати дане повідомлення. Для цього проаналізуємо можливості базового та модифікованого методів, щодо необхідного набору параметрів для приховування необхідного повідомлення, та порівняємо із отриманими статистичними даними файлової системи.

Сформуємо набір повідомлень розмірами 8, 16, 32 байт, та побудуємо таблиці із вхідними параметрами базового, модифікованого методів необхідних для приховування цих повідомлень, таблиця 3.14.

Таблиця 3.14

Необхідні вхідні параметри для надійного приховування повідомлення

Кількість файлів		Біт / кластер	Необхідна кількість кластерів / файл	
Базовий метод	Модифікований метод		Базовий метод	Модифікований метод
8 байт повідомлення (64 біт)				
2	1	1	32	64
4	2	2	8	16
8	4	3	3 (2.67)	6 (5.33)
16 байт повідомлення (128 біт)				
2	1	1	64	128
4	2	2	16	32
8	4	3	6 (5.33)	11 (10.67)
16	8	4	2	4
32 байт повідомлення (256 біт)				
2	1	1	128	256
4	2	2	32	64
8	4	3	11 (10.67)	22 (21.33)
16	8	4	4	8
32	16	5	2 (1.6)	4 (3.2)

Порівнюючи отримані дані із таблиці 3.14 із емпіричними даними із попередніх таблиць можна стверджувати, що:

— для того, щоб приховати повідомлення розміром 8 байт необхідно використовувати принаймні 8 файлів розміром 3 кластери кожен, для базового методу, або 4 файли розміром 6 кластерів, для модифікованого методу;

— для того, щоб приховати повідомлення розміром 16 байт необхідно використовувати принаймні 8 файлів розміром 6 кластерів кожен, для базового методу, або 8 файлів розміром 4 кластерів, для модифікованого методу;

– для того, щоб приховати повідомлення розміром 32 байти необхідно використовувати принаймні 16 файлів розміром 4 кластери кожен, для базового методу, або 16 файлів розміром 4 кластери, для модифікованого методу;

Загалом, можна зробити висновок, що при приховуванні більшого повідомлення необхідно збільшувати кількість покриваючих файлів, адже у файловій системі, в середньому, майже немає файлів із значним рівнем фрагментації, але є велика кількість файлів із малим рівнем фрагментації. Можна вивести закономірність щодо розміру повідомлення у залежності від кількості покриваючих файлів та допустимого рівня фрагментації, формула 3.16:

$$\begin{cases} M = F \times \lfloor \log_2 F \rfloor \times \varphi \text{ (Базовий метод)} \\ M = F \times (\lfloor \log_2 F \rfloor + 1) \times \varphi \text{ (Модифікований метод)} \end{cases} \quad (3.16)$$

де: M – розмір повідомлення у бітах; F – кількість файлів із рівнем фрагментації φ .

Оцінюючи максимальний розмір прихованого повідомлення за даними таблиці із інформацією про кількість файлів із відповідним рівнем фрагментації та формулою 3.16, можна розрахувати що:

– Базовим методом можна приховати 374 байт повідомлення використовуючи 214 файлів із розміром 2 кластери;

– Модифікованим методом можна приховати 428 байт повідомлення використовуючи 214 файлів із розміром 2 кластери;

Таким чином надана оцінка необхідних параметрів для приховування повідомлень, та надана оцінка максимально можливого повідомлення для безпечного приховування. І як висновок, використовуючи методи приховування інформації у структуру файлової системи FAT шляхом переміщування кластерів, можна приховати незначну кількість інформації (близько 32 байт) без ризику детектування прихованої інформації. Що означає що рекомендується використовувати методи для приховування такої інформації як паролі, геши чи ключи. Або необхідно

використовувати дані методу у системах де рівень захищеності інформації від детектування не є критичним параметром [39, 49].

3.3 Аналіз часових параметрів та обчислювальної складності методу приховування даних у структуру файлової системи FAT

Для аналізу часових параметрів даного методу була використана програма «SteganoFAT» (<https://drive.google.com/drive/folders/10kLAIZ-v7of7SeSIBjI3KMumFAVjjBtw?usp=sharing>), опис якої надано дипломній роботі, файлова система FAT на флеш накопичувачу JetFlash 350 Transcend ємністю 8 Гб, інтерфейс підключення USB2.0 та ноутбук Lenovo Y510P.

Програмна реалізація методів «SteganoFAT» не є ідеальною та оптимізованою, але дозволяє виявити закономірності затраченого часу методами приховування інформації.

Аналізуючи час виконання методу, будемо змінювати такі параметри:

- розмір кластеру;
- розмір файлу повідомлення;
- кількість покриваючих файлів;
- загальний розмір покриваючих файлів;
- режим роботи алгоритмів.

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 3.15.

Як видно із результатів таблиці 3.15, при збільшені розміру кластеру час на виконання приховування та вилучення повідомлення зменшується, але й пропускна

спроможність зменшується у двічі. Також необхідно зазначити, що модифікований режим роботи алгоритму потребує більше часу на приховування інформації ніж базовий, при тій самій конфігурації. На час вилучення інформації обраний режим не впливає.

Таблиця 3.15

Залежність часових параметрів методу від розміру кластеру (Час базового методу/час модифікованого методу)

Розмір одного кластеру, байт	2048	4096	8192
Час приховування повідомлення, секунд	3.341/4.276	2.87/3.41	2.37/2.96
Час вилучення повідомлення, секунд	0.022/0.021	0.012/0.012	0.008/0.009

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 3.16.

Таблиця 3.16

Залежність часових параметрів методів приховування від розміру повідомлення (Час базового методу/час модифікованого методу)

Розмір повідомлення, байт	100	200	400
Час приховування повідомлення, секунд	3.82/4.341	5.839/7.102	8.36/9.57
Час вилучення повідомлення, секунд	0.019/0.02	0.024/0.022	0.032/0.032

Як видно із таблиці 3.16, при збільшені розміру повідомлення, час на приховування та вилучення повідомлення збільшується. Це пов'язано із тим, що при збільшені розміру повідомлення, необхідно виконати більше перестановок кластерів тобто необхідно збільшити кількість переміщень зчитуваючої головки фізичного

носія інформації, також збільшується кількість зчитувань та запису даних у кластери файлової системи (що є найбільш затратною операцією).

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. Результати аналізу зазначені у таблиці 3.17.

Таблиця 3.17

Залежність часових параметрів методів приховування інформації від кількості покриваючих файлів (Час базового методу/час модифікованого методу)

Кількість покриваючих файлів	2	4	8
Час приховування повідомлення, секунд	4.693/5.601	2.76/3.31	2.704/3.01
Час вилучення повідомлення, секунд	0.022/0.017	0.025/0.026	0.032/0.031

Як видно із результатів таблиці 3.17, 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, так як загальна кількість кластерів буде збільшена, а для надійності що усе приховане повідомлення вилучено правильно, необхідно зчитати усі кластери покриваючих файлів.

Для оцінки 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. Результати аналізу вказані у таблиці 3.18.

Як видно із таблиці 3.18, при збільшені загального розміру покриваючих файлів час на приховування та вилучення повідомлення збільшується. Це пов'язано із тим, що при збільшені загального розміру покриваючих файлів, необхідно виконати більше перестановок кластерів тобто необхідно збільшити кількість переміщень зчитуваючої головки фізичного носія інформації, також збільшується кількість зчитувань та запису даних у кластери файлової системи (що є найбільш затратною операцією) [17, 39, 55, 49].

Таблиця 3.18

Залежність часових параметрів методів приховування інформації від загального розміру покриваючих файлів (Час базового методу/час модифікованого методу)

Загальний розмір покриваючих файлів, Мб	1.7	3.5	7
Час приховування повідомлення, секунд	2.083/2.201	2.417/3.01	3.293/3.71
Час вилучення повідомлення, секунд	0.007/0.006	0.012/0.013	0.021/0.02

Роблячи висновок із аналізу часових параметрів методів приховування та вилучення інформації шляхом перемішування кластерів покриваючих файлів із структури файлової системи FAT 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, окрім того не завжди є можливість встановити необхідний розмір кластеру для файлової системи;

– 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 час на вилучення є не таким значимим як час на приховування повідомлення;

– 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;

– при використанні модифікованого методу час приховування збільшується, це пов'язано із тим що, при базовому методі кластери одного файлу розміщуються один за одним, а при модифікованому методі – перемішуються, що збільшує рівень фрагментації файлів і час приховування, відповідно.

Щодо оцінки пропускну́ї здатності, необхідно впевнитись, що практичні результати будуть співпадати із теоретичними, отриманими у розділі 3.1. 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, від розміру одного кластеру та від обрання режиму роботи алгоритму. 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 формули 3.4, 3.5 у залежності від режиму роботи алгоритму, модифікованого чи базового, відповідно. Отримані результати дослідів занесені у таблицю 3.19 [39, 44, 49]. Скорочення у таблиці 3.19 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 3.4 та 3.5 по розрахунку пропускну́ї 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 пропускну здатності при інших розрахунках.

Таблиця 3.19

Розмір стеганограми у залежності від кількості покриваючих файлів та розміру одного кластеру (практичне\теоретичне значення, базовий, (модифікований) методи роботи алгоритму)

ПК \ КПФ	2	4	8
2048	1758\1757 (3514\3514)	3516\3516 (5270\5269)	5274\5273 (7021\7020)
4096	879\878 (1756\1756)	1758\1758 (2633\2633)	2637\2637 (5266\5265)
8192	439\439 (877\877)	879\879 (1316\1314)	1320\1318 (2629\2629)

Наступним кроком необхідно визначити обчислювальну складність методів приховування та вилучення інформації шляхом перестановки кластерів покриваючих файлів структури файлової системи FAT для рекомендацій щодо використання методів та розробки програмної реалізації

Так як більшість часу на приховування повідомлення у структуру файлової системи займає саме робота із фізичним носієм то виділимо такі операції:

- обчислювальна складність на переміщення зчитуваючої головки фізичного носія (для HDD накопичувачів), чи зміна позиції робочого сектору (для SSD накопичувачів) – $O(f(n))$, де n – кількість переміщень зчитуваючої головки фізичного носія у кількості пройдених секторів;
- обчислювальна складність на зчитування даних із сектору – $O(g(n))$, де n – кількість кластерів що зчитані;
- обчислювальна складність на запис даних у сектор – $O(h(n))$, де n – кількість кластерів що записано;

У деяких випадках обчислювальна складність залежить від кількості стеганоблоків – k . Далі необхідно визначити загальну обчислювальну складність для базового методу та для модифікованого. Обчислювальну складність f, g, h вважатимемо складністю за часовими ознаками, у той час як необхідну кількість оперативної пам'яті m – вважатимемо складністю за об'ємними параметрами. Обчислювальною складністю на генерацію перестановки будемо знехтувати, так як час на виконання даної операції, у порівнянні на час роботи із фізичним носієм, є мінімальним [61, 62, 63, 64].

Загальна обчислювальна складність для базового методу складається із суми перелічених вище обчислювальних складностей, формула 3.17:

$$O(B) = O(f(n)) + O(g(n)) + O(h(n)) \quad (3.17)$$

Також необхідно зазначити, що загальна обчислювальна складність залежить від кількості секторів що були перезаписані, та від кількості переміщень зчитуваючої головки пристрою, а отже необхідно виявити залежність між розміром повідомлення (кількістю стеганоблоків) та кількістю перезаписаних кластерів. Для цього спиратимемося на описаний вище (розділ 2) приклад приховування даних у структуру файлової системи, а саме на приклад у рисунку 2.6.

3.4 Математична модель оцінки основних параметрів кластерних стеганосистем

Для кластерних стеганосистем можна виділити такі основні параметри:

- пропускна здатність методу приховування;
- стійкість до детектування;
- обчислювальна складність приховування повідомлення;

Аналізуючи ці параметри, можна стверджувати щодо ефективності стеганосистеми. Також необхідно зазначити, що для різних стеганосистем можуть бути різні критичні параметри. Наприклад для системи верифікації даних пропускна здатність стеганоканалу не матиме важливого значення, а стійкість прихованого повідомлення до детектування – навпаки, матиме критичне значення. Таким чином, у даній роботі запропоновано наступну модель оцінки основних параметрів кластерних стеганосистем, що дозволяє більш повно оцінити ефективність стеганосистеми.

Нехай пропускна здатність методу буде – C (біт/кластер), стійкість до детектування – D (рівень фрагментарності), обчислювальна складність – B (кількість операцій). Необхідно провести оцінку по кожному параметру, при чому пропускна здатність та стійкість до детектування напряму залежать від можливості власника інформації впливати на стеганоконтейнер. З цього випливає, що якщо власник інформації має доступ до стеганоконтейнеру – файлової системи, то він може створити надлишковий рівень фрагментованості усієї системи, таким чином збільшити стійкість до детектування прихованого повідомлення, а також маючи доступ до стеганоконтейнеру власник інформації може згенерувати/обрати таку кількість покриваючих файлів та обрати покриваючі файли необхідного формату, способу використання так щоб збільшити пропускну здатність та стійкість до детектування (чим більша кількість покриваючих файлів тим більша кількість біт на один кластер може бути прихована). Отримавши оцінку по кожному параметру можна ввести коефіцієнт значимості даного параметра для стеганосистеми в цілому, та вивести загальне значення оцінки параметра, схематично модель оцінки основних параметрів кластерних стеганосистем зображено на рисунку 3.14.

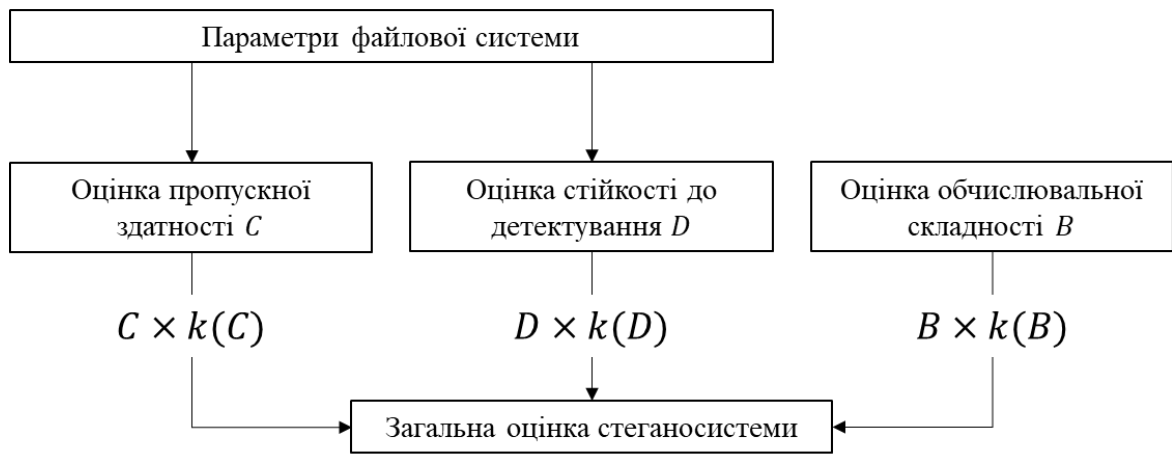


Рис. 3.14 Модель оцінки основних параметрів кластерних стеганосистем

3.5 Удосконалення методів приховування інформації у структуру кластерних файлових систем

Приховування інформації стеганографічними методами виконується за рахунок перезапису даних із кластерів. Та для того щоб покриваючи файли були цілісними інформацію необхідно копіювати повністю, це робиться із використання оперативної пам'яті. За способом роботи із оперативною пам'яттю можна виділити такі варіації [65, 66]:

- повний запис даних з кластерів до оперативної пам'яті, у такому випадку необхідна кількість оперативної пам'яті залежить від кількості кластерів покриваючих файлів;
- почерговий запис даних з кластерів до оперативної пам'яті, у такому випадку у оперативну пам'ять зчитуються дані з кластеру B , після чого записуються дані з кластеру A , далі каретка зчитуваючої головки переміщуються до кластеру B і процес повторюється (зчитуються дані із B , записуються дані із B і так далі). Таким чином при виконанні приховування інформації необхідно мати розмір оперативної пам'яті як подвійний розмір до одного кластеру.

За шляхом генерації перестановок, можна виділити такі варіації виконання методів приховування:

— виконання перестановки із подальшим послідовним перезаписом кластерів у необхідній послідовності. Це означає, що спочатку приховується кластери які складають стеганограму, а вже після цього необхідно розмістити усі вільні (ті які не несуть інформаційного навантаження на приховування повідомлення) кластери покриваючих файлів у впорядкованій послідовності. Спочатку впорядковані кластери першого покриваючого файлу, потім другого і так далі. Така варіація методу приховування дозволяє знизити надлишковий рівень фрагментації покриваючих файлів;

— виконання перестановки із переміщенням лише інформативних кластерів у нормальну послідовність. У такому випадку спочатку переміщуємо інформаційні кластери у відповідну до повідомлення послідовність, після чого переміщуємо кластери які були витиснені інформаційними кластерами. Витиснені кластери впорядковано розміщуємо лише на позиції де знаходилися інформаційні кластери. Перевагою такого методу є те що необхідно буде перемістити лише обмежену розміром повідомлення кількість кластерів;

— виконання перестановки із оптимальним переміщенням лише інформативних кластерів. У такому випадку переміщуємо інформаційні кластери у відповідну до повідомлення послідовність, але із можливістю збереження позиції кластером, якщо він відповідає значенню стеганоблока. Усе інше аналогічно до попереднього методу. Такий метод дозволяє ще значніше зменшити кількість перезаписів кластерів, але можуть виникати випадку коли покриваючи файли матимуть переплетеність (деякі кластери файлу розміщені у зворотній послідовності, що є аномальною поведінкою файлової системи що може детектувати приховане повідомлення).

Комбінуючи методи оптимізації за оперативною пам'яттю та методи зменшення обчислювальної складності було запропоновано 4 варіації приховування інформації у структуру кластерних файлових систем (чотири для базового методу та чотири для модифікованого) із прикладами [65, 67].

Розглянемо такі варіанти перестановки повідомлення:

1. Виконання перестановки із повним завантаженням кластерів до оперативної пам'яті та подальшим послідовним перезаписом кластерів у необхідній послідовності, рисунок 3.15. Далі даний метод називатимемо повне завантаження до оперативної пам'яті (ПЗОП).

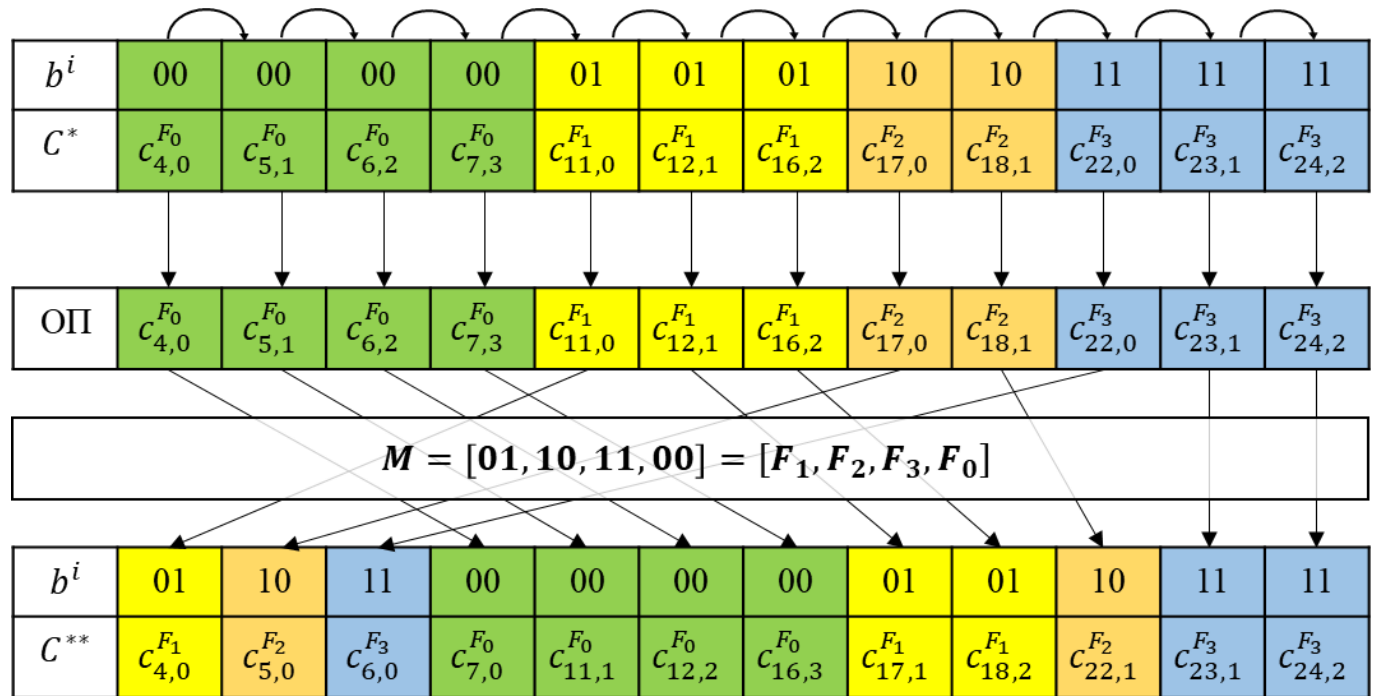


Рис. 3.15 Приклад перестановки кластерів із повним завантаженням даних у операційну пам'ять

Даний метод реалізацій перестановки дозволяє нівелювати обчислювальну складність на переміщення зчитуваючої головки фізичного носія, так як по перше виконується послідовне зчитування кластерів до операційної пам'яті. Потім перестановка виконується у операційній пам'яті як робота із масивом, а далі так само послідовно виконуємо перезапис кластерів. Необхідно зазначити, що спосіб перемішування кластерів потребує значного розміру оперативної пам'яті, щоб завантажити усі кластери усіх покриваючих файлів до неї. А отже $O(m(n)) = n \times Cluster_{size}$, що означає що може виникнути ситуація коли операція

перестановки не може бути виконана взагалі. Але як висновок необхідно буде перезаписати усі кластери, а отже загальна обчислювальна складність матиме вигляд, формула 3.18.

$$O(B) = O(f(2n)) + O(g(n)) + O(h(n)); n = C_{len} \quad (3.18)$$

де C_{len} – довжина матриці стану у кластерах, тобто загальний розмір покриваючих файлів у кластерах. Подвійне переміщення зчитуваючої головки як раз і пов’язана із тим що необхідно спочатку зчитати усі кластери, а потім записати усі кластери.

Для даного прикладу матимемо такі показники, що зазначені у таблиці 3.20, де $\varphi(F_i)$ означає кількість фрагментів та рівень фрагментації (відстань між фрагментами), відповідно, а π – правило перестановки [65]:

Таблиця 3.20

Показники результату приховування прикладу шляхом ПЗОП

π	(4,7,16,18,22,6,12,17,5,11)(23)(24)
$O(f(n))$	20
$O(g(n))$	10
$O(h(n))$	10
$O(m(n))$	10
$\varphi(F_0)$	1
$\varphi(F_1)$	2 (6)
$\varphi(F_2)$	2 (7)
$\varphi(F_3)$	2 (7)

2. Виконання перестановки із почерговим завантаженням кластерів до оперативної пам’яті із впорядкованим розміщенням залишку кластерів, рисунок 3.16. Далі даний метод називатимемо почерговим завантаженням до оперативної пам’яті І (ПчЗОП-І).

Особливість даного методу полягає у тому що перестановка виконується почергово по ланцюгу кластерів, тобто на першій ітерації зчитується перший кластер

ланцюгу, у наступній ітерації зчитується кластер на який слідом перезаписуємо попередньо зчитаний кластер. Таким чином зростає кількість переміщень зчитуваючої головки пристрою, так як доводиться “стрибати” назад та вперед по кластерам. Що означає що кількість переміщень може бути максимум сумою чисел до числа n , де n – кількість кластерів, а отже $O\left(f\left(n * \frac{n+1}{2}\right)\right) = O(f(n^2))$, тобто складність має квадратичний характер. І так як усе одно виконується впорядкований перезапис подальших кластерів, то обчислювальна складність буде дорівнювати формулі 3.19.

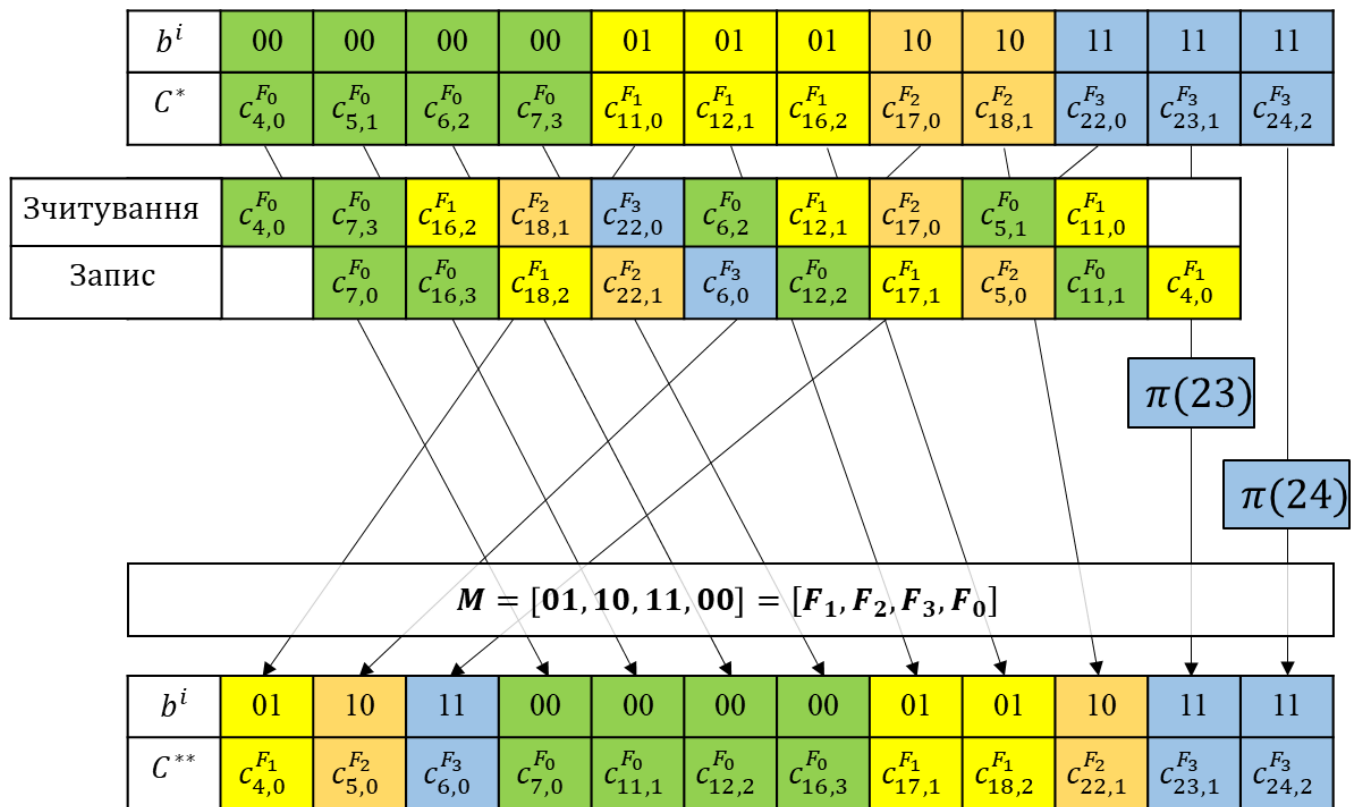


Рис. 3.16 – Приклад перестановки кластерів із почерговим завантаженням кластерів (ПчЗОП-I)

$$O(B) = O(f(n^2)), + O(g(n)) + O(h(n)); n = C_{len} \quad (3.19)$$

де C_{len} – довжина матриці стану у кластерах, тобто загальний розмір покриваючих файлів у кластерах. При чому можуть виникати умови коли кластер перезаписується сам у себе у такому випадку виконувати перезапис кластеру не є необхідним. Також необхідно зазначити що при виконанні перестановки методом ПчЗОП-I обчислювальна система потребує лише подвійного розміру кластеру у якості оперативної пам'яті, тобто $O(m(const)) = 2$ що дозволяє виконувати приховування інформації теоретично у необмежені за розміром покриваючи файли. Для даного прикладу матимемо такі показники, що зазначені у таблиці 3.21 [65].

Таблиця 3.21

Показники результату приховування прикладу шляхом ПчЗОП-I

π	(4,7,16,18,22,6,12,17,5,11)(23)(24)
$O(f(n^2))$	34
$O(g(n))$	10
$O(h(n))$	10
$O(m(const))$	2
$\varphi(F_0)$	1
$\varphi(F_1)$	2 (6)
$\varphi(F_2)$	2 (7)
$\varphi(F_3)$	2 (7)

3. Виконання перестановки із почерговим завантаженням кластерів до оперативної пам'яті із переміщенням лише інформативних кластерів у нормальну послідовність, рисунок 3.17. Далі даний метод називатимемо почерговим завантаженням до оперативної пам'яті (ПчЗОП-II).

Особливість даного способу приховування інформацій полягає у тому, що при розрахунку остаточної матриці стану кластерів перемішування виконується лише над кластерами які безпосередньо беруть участь у приховуванні інформаційного повідомленні. Можуть переміщуватися не інформаційні кластери лише у випадку коли їх заміщають інформаційні кластери. Наступним кроком є розміщення переміщених кластерів у нормальній послідовності, тобто кластери одного файлу повинні міти

індекси що зростають зліва на право, тобто не мати переплетеності між кластерами. Це дозволяє зменшити рівень фрагментації для можливо переплєтених покриваючих файлів.

У даному випадку кількість переміщень зчитуваючої головки та кількість зчитувань та записів кластерів вже залежить від кількості стеганоблоків – k , а не лише від кількості кластерів покриваючих файлів – n . У той час оперативної пам'яті необхідно так само лише подвійний розмір кластеру.

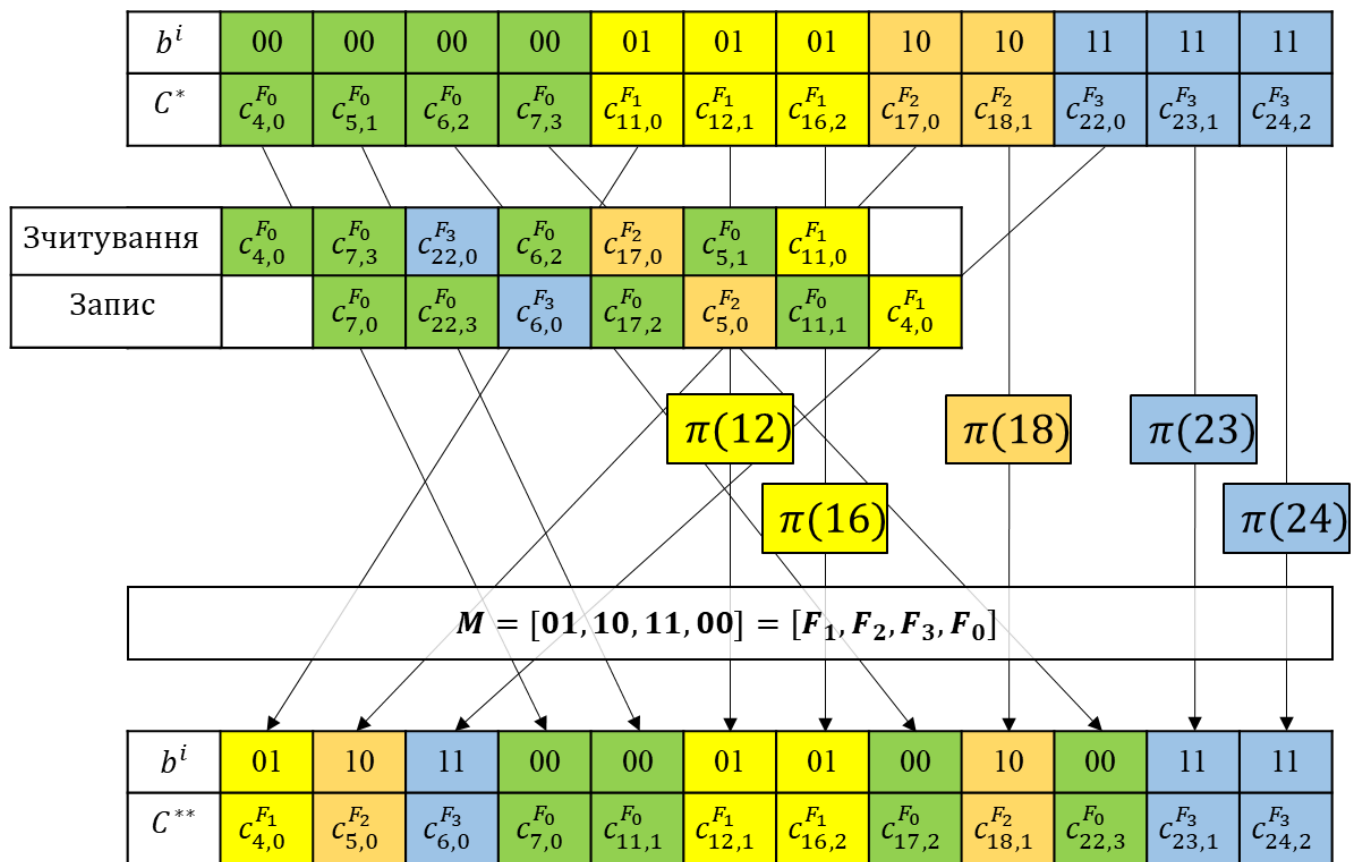


Рис. 3.17 Приклад перестановки кластерів із почерговим завантаженням кластерів (ПчЗОП-II)

Для того щоб виконати перестановку необхідно перемістити лише k кластерів які можуть замістити собою іще k кластерів. Дані кластери можуть розміщуватися по усій довжині матриці стану, тобто переміщення зчитуваючої головки для

переміщення одного кластеру може бути із крайньої лівої позиції до крайньої правої тобто необхідно пройти по усім кластерам – n . А отже кількість переміщень зчитуваючої головки дорівнює – $O(f(2kn))$. При чому якщо розмір повідомлення у стеганоблоках наближається до кількості кластерів то обчислювальна складність переміщень зчитуваючої головки наближається до квадратичної залежності – $O(f(n^2)); k \rightarrow n$.

Кількість зчитувань та записів кластерів фіксована та залежить лише від кількості стеганоблоків, але таких записів може бути по два коли інформаційний кластер заміщує не інформаційний. А отже $O(g(2k))$ та $O(h(2k))$. Результуюча обчислювальна складність матиме вигляд формули 3.20 [65].

$$O(B) = O(f(2kn)) + O(g(2k)) + O(h(2k)); n = C_{len}; k = M_{len} \quad (3.20)$$

Так як не виконується подальше перерозміщення неінформативних кластерів у суцільні послідовності, то зазвичай рівень фрагментації покриваючих файлів, при перерозміщенні у такий спосіб, буде вищий а ніж при виконанні методу приховування інформації шляхом ПчЗОП-I. Для даного прикладу матимемо такі показники, що зазначені у таблиці 3.22.

Таблиця 3.22

Показники результату приховування прикладу шляхом ПчЗОП-II

π	(4,7,22,6,17,5,11)(12)(16)(18)(23)(24)
$O(f(n^2))$	34
$O(g(2k))$	8
$O(h(2k))$	8
$O(m(const))$	2
$\varphi(F_0)$	3 (3)
$\varphi(F_1)$	2 (4)
$\varphi(F_2)$	2 (6)
$\varphi(F_3)$	2 (7)

4. Виконання перестановки із почерговим завантаженням кластерів до оперативної пам'яті із оптимальним переміщенням лише інформативних кластерів, рисунок 3.18. Далі даний метод називатимемо почерговим завантаженням до оперативної пам'яті ПчЗОП-III.

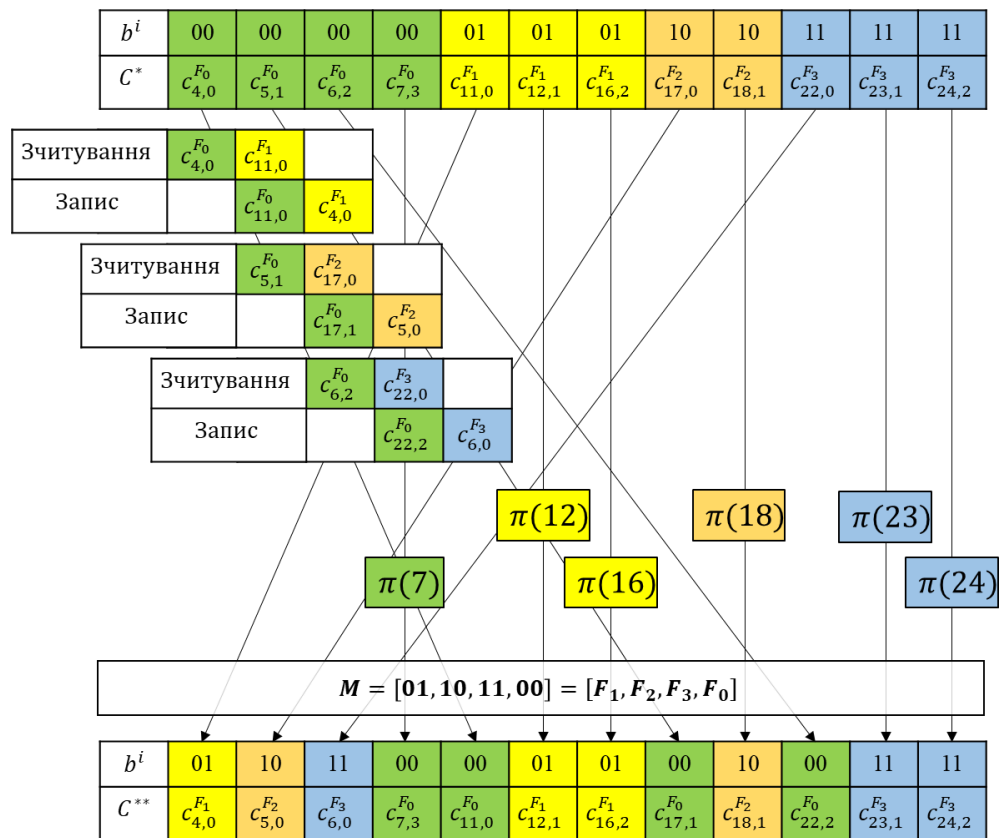


Рис. 3.18 Приклад перестановки кластерів із почерговим завантаженням кластерів (ПчЗОП-III)

Особливість даного способу приховування інформації полягає у тому, що при розрахунку остаточної матриці стану кластерів перемішування виконується лише над кластерами які безпосередньо беруть участь у приховуванні інформаційного повідомлення, але кластер файлу у початковому стані співпадає із кластером також файлу у кінцевому стані, то такий кластер не переміщується. Така особливість може призвести до зростання переплетеності покриваючих файлів, що збільшить рівень

фрагментації. Але така особливість надає перевагу так як зменшиться кількість переміщень та перезаписів у порівнянні із способом ПчЗОП-II. Можуть переміщуватися не інформаційні кластери лише у випадку коли їх заміщають інформаційні кластери. Також така особливість (ПчЗОП-III) має сенс якщо використовується модифікований метод приховування інформації, так як він по визначенню призведе до зростання переплетеності.

У даному випадку кількість переміщень зчитуваючої головки та кількість зчитувань та записів кластерів вже залежить від кількості стеганоблоків – k , а не лише від кількості кластерів покриваючих файлів – n . У той час оперативної пам'яті необхідно так само лише подвійний розмір кластеру.

Для того щоб виконати перестановку необхідно перемістити лише k кластерів які можуть замінити собою іще k кластерів. Дані кластери можуть розміщуватися по усій довжині матриці стану, тобто переміщення зчитуваючої головки для переміщення одного кластеру може бути із крайньої лівої позиції до крайньої правої тобто необхідно пройти по усім кластерам – n . А отже кількість переміщень зчитуваючої головки дорівнює – $O(f(2kn))$. При чому якщо розмір повідомлення у стеганоблоках наближається до кількості кластерів то обчислювальна складність переміщень зчитуваючої головки наближається до квадратичної залежності – $O(f(n^2)); k \rightarrow n$.

Кількість зчитувань та записів кластерів фіксована та залежить лише від кількості стеганоблоків, але таких записів може бути по два коли інформаційний кластер заміщує не інформаційний. Але можливі випадки коли деякі кластери можна не перезаписувати, як описано вище. А отже $O(g(2k))^-$ та $O(h(2k))^-$. Результуюча обчислювальна складність матиме вигляд формули 3.21 [65].

$$O(B) = O(f(2kn)) + O(g(2k))^- + O(h(2k))^-; n = C_{len}; k = M_{len} \quad (3.21)$$

Так як не виконується подальше перерозміщення неінформативних кластерів у суцільні послідовності, то зазвичай рівень фрагментації покриваючих файлів, при перерозміщенні у такий спосіб, буде вищий а ніж при виконанні методу приховування інформації шляхом ПчЗОП-I, та через можливу переплетеність деяких файлів рівень фрагментації буде вищий за рівень фрагментації при виконанні приховування повідомлення шляхом ПчЗОП-II. Але для систем де рівень фрагментації покриваючих файлів не є значним показником то приховування повідомлення шляхом перемішування кластерів покриваючих файлів спосіб виконання перестановки із почерговим завантаженням кластерів до оперативної пам'яті із оптимальним переміщенням лише інформативних кластерів є оптимальним. Для даного прикладу матимемо такі показники, що зазначені у таблиці 3.23.

Таблиця 3.23

Показники результату приховування прикладу шляхом ПчЗОП-III

π	(4,11)(5,17)(6,22)(7)(12)(16)(18)(23)(24)
$O(f(n^2))$	36
$O(g(2k))^-$	6
$O(h(2k))^-$	6
$O(m(const))$	2
$\varphi(F_0)$	4 (8)
$\varphi(F_1)$	2 (4)
$\varphi(F_2)$	2 (6)
$\varphi(F_3)$	2 (7)

Порівняємо дані способи реалізації базового методу приховування інформації шляхом перемішування кластерів покриваючих файлів за такими показниками:

- Обчислювальна складність за необхідним об'ємом вільного місця у оперативній пам'яті – $O(m)$;
- Обчислювальна складність за часовими показниками – $O(f)$, $O(g)$, $O(h)$;
- Захищеність від детектування, тобто за впливом на рівень фрагментації.

Порівнюючи способи реалізації по обчислювальній складності за часовими показниками, необхідно зазначити, що час на переміщення зчитуваючої головки пристрою займає найменше часу у абсолютних величинах. А час на запис кластеру займає найбільше часу. Тобто, затрачений час підпорядковується такій закономірності, формула 3.22

$$O(f) < O(g) < O(h) \quad (3.22)$$

При чому, для SSD технології час затрачений на переміщення зчитуваючої головки наближається до нуля, так як у SSD пристроях реалізована паралельна обробка секторів, та таке визначення як зчитуваюча головка пристрою не має сенсу. Отже результат порівняльного аналізу способів реалізації методу зазначено у таблиці 3.24.

Таблиця 3.24

**Результат порівняльного аналізу способів реалізації базового методу
приховування інформації**

Спосіб реалізації	Необхідний об'єм ОП	Необхідний час	Захищеність від детектування
ПЗОП	--	-	++
ПчЗОП-I	++	--	++
ПчЗОП-II	++	+	+
ПчЗОП-III	++	++	-

Роблячи висновок за результатами порівняльного аналізу можна стверджувати що:

— для систем де розмір оперативної пам'яті є достатнім (розмір покриваючих файлів цілком уміщається у оперативну пам'ять) переважним способом буде ПЗОП. Даний метод дозволяє досягти мінімального впливу на рівень фрагментації покриваючих файлів, але ПЗОП потребує значного часу на виконання приховування повідомлення;

– для систем де затрачений час є критичним параметром, то у такому випадку рекомендується використовувати способи ПчЗОП-II та ПчЗОП-III. ПчЗОП-III потребує менше часу на приховування повідомлення, але вплив на рівень фрагментації у такий спосіб найбільший;

– для систем у яких захищеність від детектування є критичним параметром рекомендовано використовувати способи ПЗОП та ПчЗОП-I (у залежності від доступного об'єму ОП).

Окремо можна виділити ПчЗОП-II як найоптимальніший метод, через те що даний метод дозволяє досягти задовільних показників при приховуванні повідомлення без збитку у інших показниках [39, 49, 65].

Також необхідно зазначити, якщо для приховування повідомлення буде задіяно значну кількість кластерів покриваючих файлів (тобто кількість стеганоблоків наближена ко кількості кластерів), то часові переваги способів ПчЗОП-II та ПчЗОП-III нівелюються. У такому випадку бажано використовувати ПЗОП та ПчЗОП-I, у залежності від допустимого об'єму ОП.

Після того як розглянули способи реалізації приховування даних у структуру файлової системи за базовими методом, необхідно оцінити обчислювальну складність й для модифікованого методу, за різними способами реалізації.

Способи перестановки матимуть такі само особливості як і при базовому методі, отже розглянемо такі варіанти реалізації перестановки [39, 65, 68]:

1. Виконання перестановки із повним завантаженням кластерів до оперативної пам'яті та подальшим послідовним перезаписом кластерів у необхідній послідовності, рисунок 3.19. Далі даний метод називатимемо повне завантаження до оперативної пам'яті (ПЗОПм).

Даний метод реалізацій перестановки дозволяє нівелювати обчислювальну складність на переміщення зчитуваючої головки фізичного носія, так як по перше виконується послідовне зчитування кластерів до операційної пам'яті. Потім перестановка виконується у операційній пам'яті як робота із масивом, а далі так само

послідовно виконуємо перезапис кластерів. Необхідно зазначити, що спосіб перемішування кластерів потребує значного розміру оперативної пам'яті, щоб завантажити усі кластери усіх покриваючих файлів до неї. А отже $O(m(n)) = n \times Cluster_{size}$, що означає що може виникнути ситуація коли операція перестановки не може бути виконана взагалі. Але як висновок необхідно буде перезаписати усі кластери, а отже загальна обчислювальна складність матиме вигляд, формула 3.23.

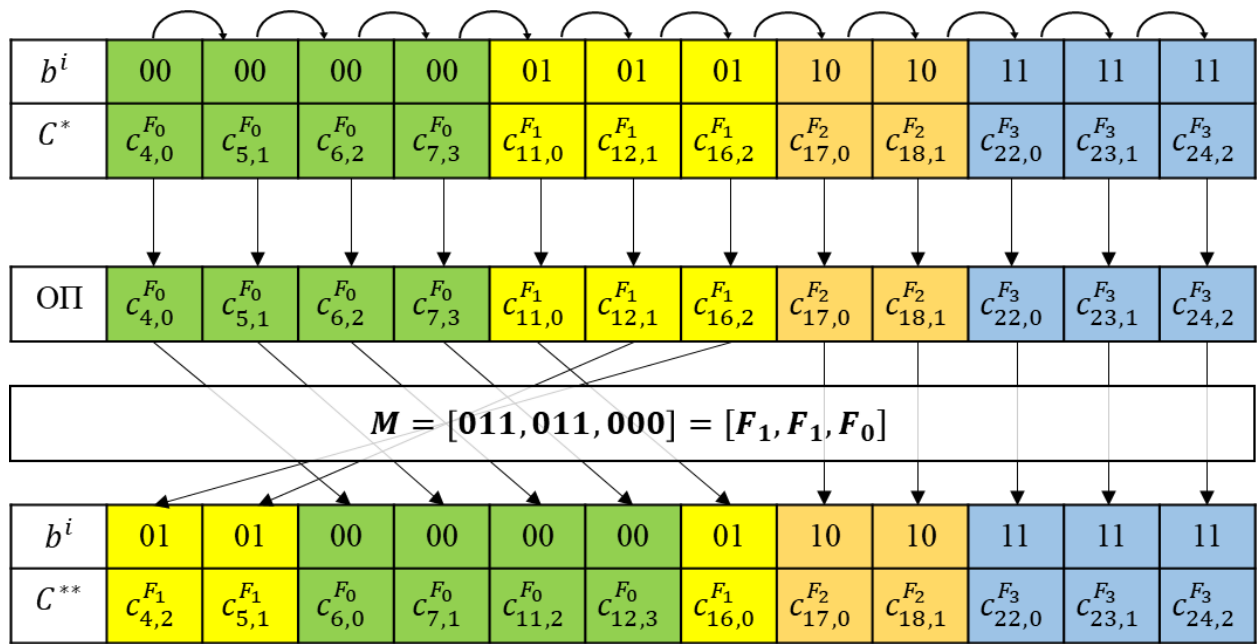


Рис. 3.19 Приклад перестановки кластерів із повним завантаженням даних у операційну пам'ять (модифікований метод)

$$O(B) = O(f(2n)) + O(g(n)) + O(h(n)); n = C_{len} \quad (3.23)$$

Для даного прикладу матимемо такі показники, що зазначені у таблиці 3.25, де $\varphi(F_i)$ означає кількість фрагментів та рівень фрагментації (відстань між фрагментами), відповідно, а π – правило перестановки:

Таблиця 3.25

Показники результату приховування прикладу шляхом ПЗОПм

π	(4,6,11,16)(5,7,12)(17)(18)(22)(23)(24)
$O(f(n))$	20
$O(g(n))$	7
$O(h(n))$	7
$O(m(n))$	7
$\varphi(F_0)$	1
$\varphi(F_1)$	3 (6)
$\varphi(F_2)$	1
$\varphi(F_3)$	1

2. Виконання перестановки із почерговим завантаженням кластерів до оперативної пам'яті із впорядкованим розміщенням залишку кластерів модифікованим методом, рисунок 3.20. Далі даний метод називатимемо почерговим завантаженням до оперативної пам'яті І (ПчЗОП-Ім).

Особливість даного методу полягає у тому що перестановка виконується почергово по ланцюгу кластерів, тобто на першій ітерації зчитується перший кластер ланцюгу, у наступній ітерації зчитується кластер на який слідом перезаписуємо попередньо зчитаний кластер. Таким чином зростає кількість переміщень зчитуваючої головки пристрою, так як доводиться “стрибати” назад та вперед по кластерам. Що означає що кількість переміщень може бути максимум сумою чисел до числа n , де n – кількість кластерів, а отже $O\left(f\left(n * \frac{n+1}{2}\right)\right) = O(f(n^2))$, тобто складність має квадратичний характер. І так як усе одно виконується впорядкований перезапис подальших кластерів, то обчислювальна складність буде дорівнювати формулі 3.24.

$$O(B) = O(f(n^2)), + O(g(n)) + O(h(n)); n = C_{len} \quad (3.24)$$

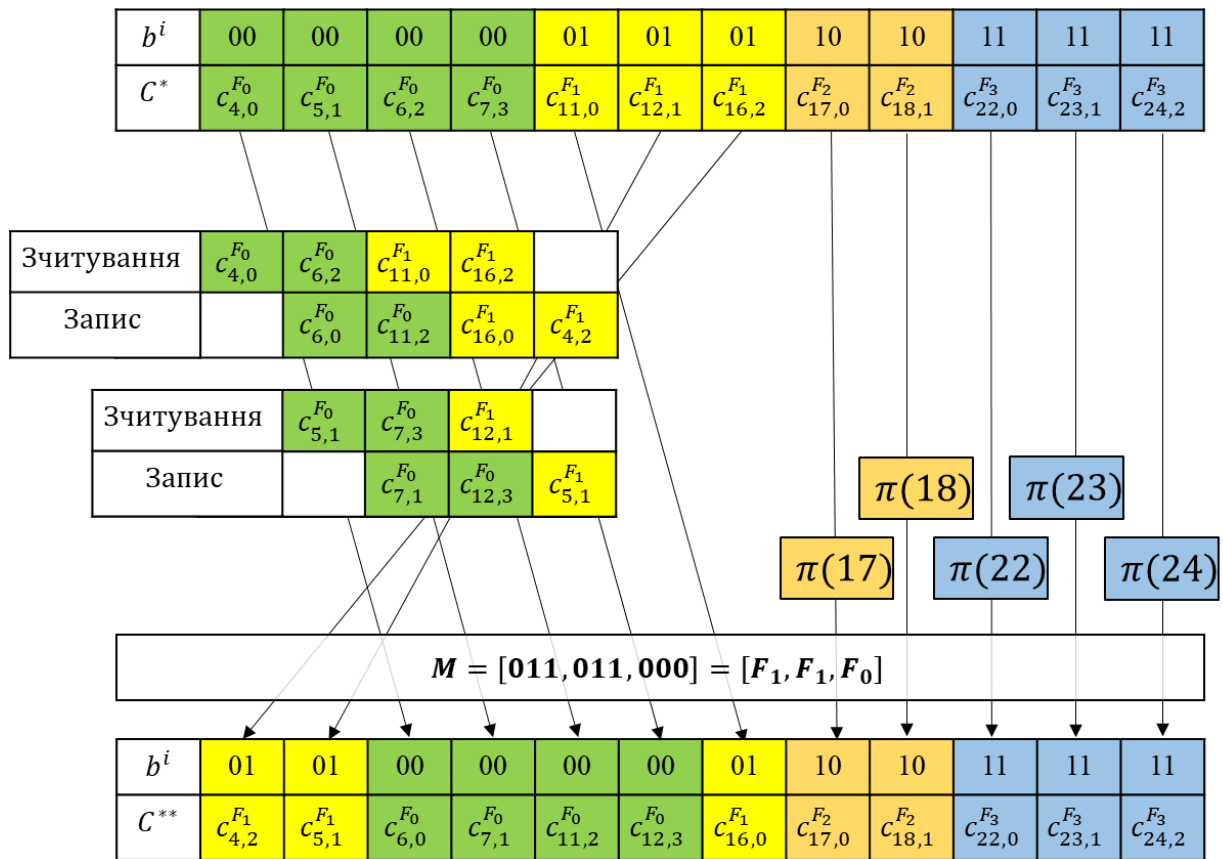


Рис. 3.20 Приклад перестановки кластерів із почерговим завантаженням кластерів (ПчЗОП-Ім)

При чому можуть виникати умови коли кластер перезаписується сам у себе у такому випадку виконувати перезапис кластеру не є необхідним. Також необхідно зазначити що при виконанні перестановки методом ПчЗОП-Ім обчислювальна система потребує лише подвійного розміру кластеру у якості оперативної пам'яті, тобто $O(m(const)) = 2$ що дозволяє виконувати приховування інформації теоретично у необмежені за розміром покриваючи файли. Для даного прикладу матимемо такі показники, що зазначені у таблиці 3.26

Таблиця 3.26

Показники результату приховування прикладу шляхом ПчЗОП-I

π	(4,6,11,16)(5,7,12)(17)(18)(22)(23)(24)
$O(f(n^2))$	20
$O(g(n))$	7
$O(h(n))$	7
$O(m(const))$	2
$\varphi(F_0)$	1
$\varphi(F_1)$	3 (6)
$\varphi(F_2)$	1
$\varphi(F_3)$	1

3. Виконання перестановки із почерговим завантаженням кластерів до оперативної пам'яті із переміщенням лише інформативних кластерів у нормальну послідовність для модифікованого методу, рисунок 3.21. Далі даний метод називатимемо почерговим завантаженням до оперативної пам'яті (ПчЗОП-ІІм).

Особливість даного способу приховування інформацій полягає у тому, що при розрахунку остаточної матриці стану кластерів перемішування виконується лише над кластерами які безпосередньо беруть участь у приховуванні інформаційного повідомлення. Можуть переміщуватися не інформаційні кластери лише у випадку коли їх заміщають інформаційні кластери. Наступним кроком є розміщення переміщених кластерів у нормальній послідовності, тобто кластери одного файлу повинні мати індекси що зростають зліва на право, тобто не мати переплетеності між кластерами (за можливістю, якщо це не порушить приховане повідомлення). Це дозволяє зменшити рівень фрагментації для можливо переплетених покриваючих файлів.

У даному випадку кількість переміщень зчитуваючої головки та кількість зчитувань та записів кластерів вже залежить від кількості стеганоблоків – k , а не лише від кількості кластерів покриваючих файлів – n . У той час оперативної пам'яті необхідно так само лише подвійний розмір кластеру.

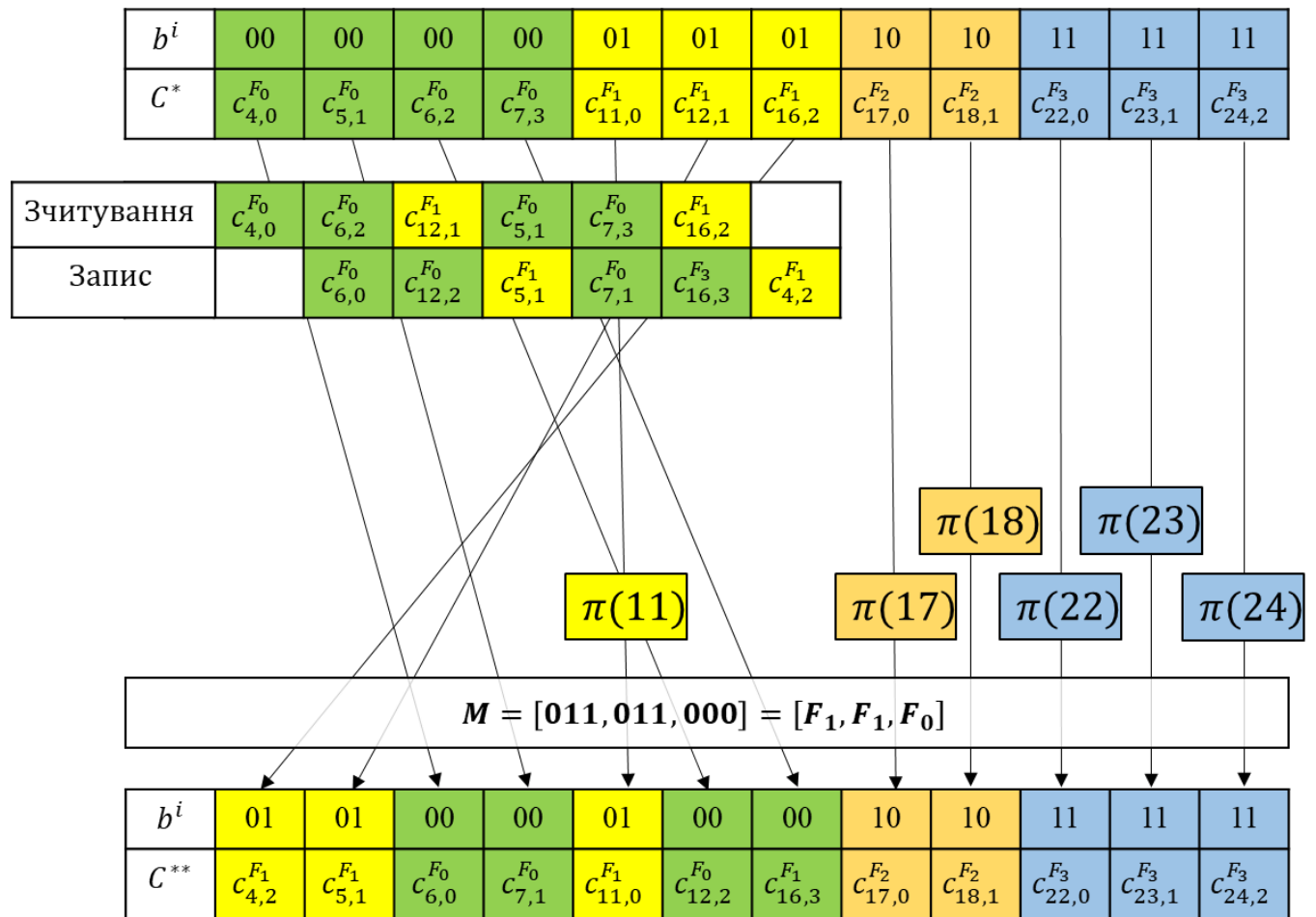


Рис. 3.21 Приклад перестановки кластерів із почерговим завантаженням кластерів (ПчЗОП-ІІм)

Для того щоб виконати перестановку необхідно перемістити лише k кластерів які можуть замінити собою іще k кластерів. Дані кластери можуть розміщуватися по усій довжині матриці стану, тобто переміщення зчитуваючої головки для переміщення одного кластеру може бути із крайньої лівої позиції до крайньої правої тобто необхідно пройти по усім кластерам – n . А отже кількість переміщень зчитуваючої головки дорівнює – $O(f(2kn))$. При чому якщо розмір повідомлення у стеганоблоках наближається до кількості кластерів то обчислювальна складність переміщень зчитуваючої головки наближається до квадратичної залежності – $O(f(n^2)); k \rightarrow n$.

Кількість зчитувань та записів кластерів фіксована та залежить лише від кількості стеганоблоків, але таких записів може бути по два коли інформаційний кластер заміщує не інформаційний. А отже $O(g(2k))$ та $O(h(2k))$. Результуюча обчислювальна складність матиме вигляд формули 3.25.

$$O(B) = O(f(2kn)) + O(g(2k)) + O(h(2k)); n = C_{len}; k = M_{len} \quad (3.25)$$

Так як не виконується подальше перерозміщення неінформативних кластерів у суцільні послідовності, то зазвичай рівень фрагментації покриваючих файлів, при перерозміщенні у такий спосіб, буде вищий а ніж при виконанні методу приховування інформації шляхом ПчЗОП-Ім. Для даного прикладу матимемо такі показники, що зазначені у таблиці 3.27.

Таблиця 3.27

Показники результату приховування прикладу шляхом ПчЗОП-Ім

π	(4,6,12,5,7,16)(11)(17)(18)(22)(23)(24)
$O(f(n^2))$	34
$O(g(2k))$	6
$O(h(2k))$	6
$O(m(const))$	2
$\varphi(F_0)$	2 (1)
$\varphi(F_1)$	3 (3)
$\varphi(F_2)$	1
$\varphi(F_3)$	1

4. Виконання перестановки із почерговим завантаженням кластерів до оперативної пам'яті із оптимальним переміщенням лише інформативних кластерів для модифікованого методу, рисунок 3.22. Далі даний метод називатимемо почерговим завантаженням до оперативної пам'яті (ПчЗОП-ІІІм).

Особливість даного способу приховування інформації полягає у тому, що при розрахунку остаточної матриці стану кластерів перемішування виконується лише над кластерами які безпосередньо беруть участь у приховуванні інформаційного

повідомленні, але кластер файлу у початковому стані співпадає із кластером також файлу у кінцевому стані, то такий кластер не переміщується. Така особливість може призвести до зростання переплетеності покриваючих файлів, що збільшить рівень фрагментації. Але така особливість надає перевагу так як зменшиться кількість переміщень та перезаписів у порівнянні із способом ПчЗОП-ІІм. Можуть переміщуватися не інформаційні кластери лише у випадку коли їх заміщають інформаційні кластери. Також така особливість (ПчЗОП-ІІІм) має сенс якщо використовується модифікований метод приховування інформації, так як він по визначенню призведе до зростання переплетеності.

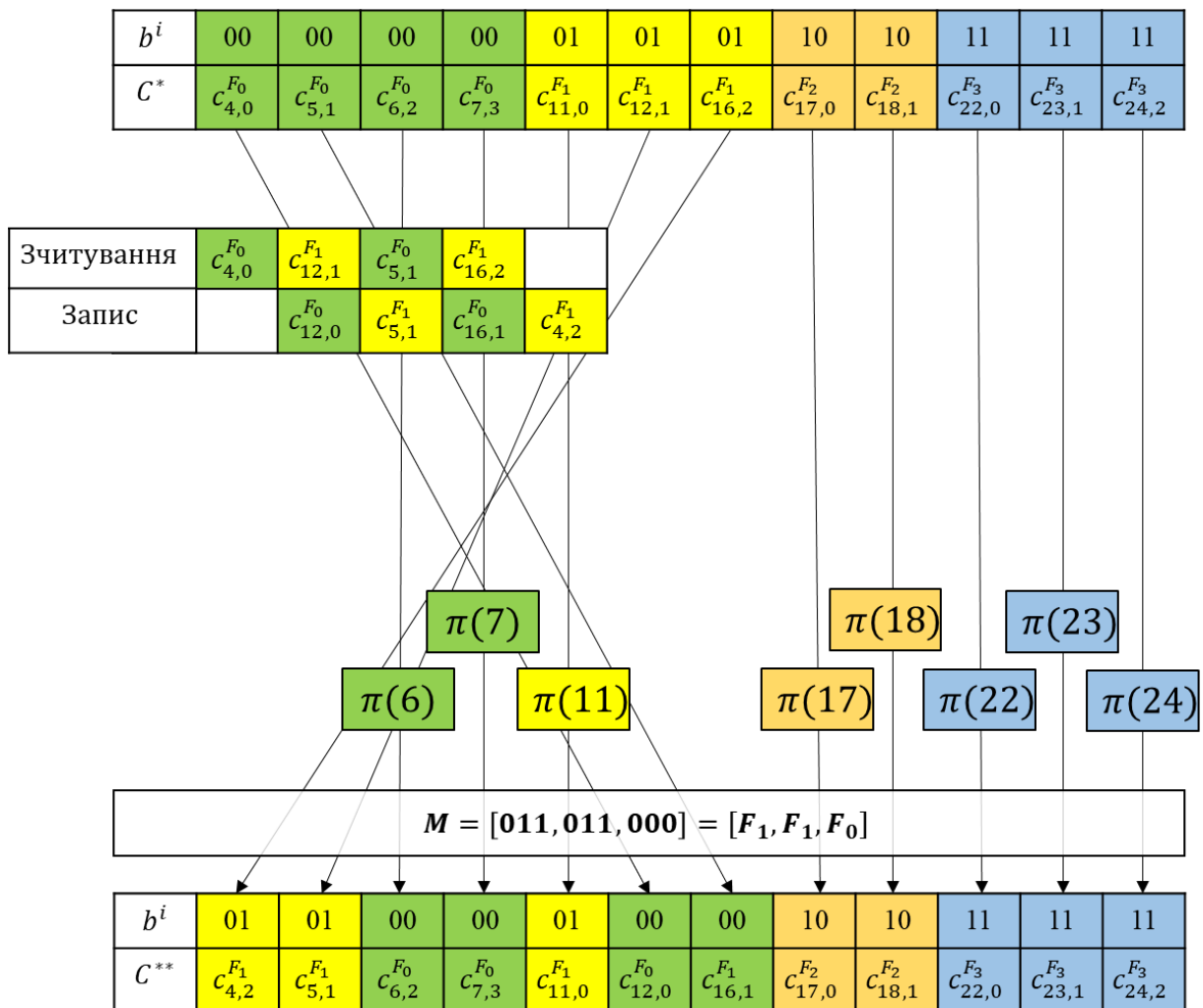


Рис. 3.22 Приклад перестановки кластерів із почерговим завантаженням кластерів (ПчЗОП-ІІІм)

У даному випадку кількість переміщень зчитуваючої головки та кількість зчитувань та записів кластерів вже залежить від кількості стеганоблоків – k , а не лише від кількості кластерів покриваючих файлів – n . У той час оперативної пам'яті необхідно так само лише подвійний розмір кластеру.

Для того щоб виконати перестановку необхідно перемістити лише k кластерів які можуть замінити собою іще k кластерів. Дані кластери можуть розміщуватися по усій довжині матриці стану, тобто переміщення зчитуваючої головки для переміщення одного кластеру може бути із крайньої лівої позиції до крайньої правої тобто необхідно пройти по усім кластерам – n . А отже кількість переміщень зчитуваючої головки дорівнює – $O(f(2kn))$. При чому якщо розмір повідомлення у стеганоблоках наближається до кількості кластерів то обчислювальна складність переміщень зчитуваючої головки наближається до квадратичної залежності – $O(f(n^2)); k \rightarrow n$.

Кількість зчитувань та записів кластерів фіксована та залежить лише від кількості стеганоблоків, але таких записів може бути по два коли інформаційний кластер заміщує не інформаційний. Але можливі випадки коли деякі кластери можна не перезаписувати, як описано вище. А отже $O(g(2k))^-$ та $O(h(2k))^-$. Результуюча обчислювальна складність матиме вигляд формули 3.26.

$$O(B) = O(f(2kn)) + O(g(2k))^- + O(h(2k))^-; n = C_{len}; k = M_{len} \quad (3.26)$$

Так як не виконується подальше перерозміщення неінформативних кластерів у суцільні послідовності, то зазвичай рівень фрагментації покриваючих файлів, при перерозміщенні у такий спосіб, буде вищий а ніж при виконанні методу приховування інформації шляхом ПчЗОП-Ім, та через можливу переплетеність деяких файлів рівень фрагментації буде вищий за рівень фрагментації при виконанні приховування повідомлення шляхом ПчЗОП-ІІм. Але для систем де рівень фрагментації

покриваючих файлів не є значним показником то приховування повідомлення шляхом перемішування кластерів покриваючих файлів спосіб виконання перестановки із почерговим завантаженням кластерів до оперативної пам'яті із оптимальним переміщенням лише інформативних кластерів є оптимальним. Для даного прикладу матимемо такі показники, що зазначені у таблиці 3.28.

Таблиця 3.28

Показники результату приховування прикладу шляхом ПчЗОП-Шм

π	(4,12,5,16)(6)(7)(11)(17)(18)(22)(23)(24)
$O(f(n^2))$	20
$O(g(2k))^-$	4
$O(h(2k))^-$	4
$O(m(const))$	2
$\varphi(F_0)$	3 (4)
$\varphi(F_1)$	3 (3)
$\varphi(F_2)$	1
$\varphi(F_3)$	1

Порівняємо дані способи реалізації модифікованого методу приховування інформації шляхом перемішування кластерів покриваючих файлів за такими показниками:

- Обчислювальна складність за необхідним об'ємом вільного місця у оперативній пам'яті – $O(m)$;
- Обчислювальна складність за часовими показниками – $O(f)$, $O(g)$, $O(h)$;
- Захищеність від детектування, тобто за впливом на рівень фрагментації.

Порівнюючи способи реалізації по обчислювальній складності за часовими показниками, необхідно зазначити, що час на переміщення зчитуваючої головки пристрою займає найменше часу у абсолютних величинах. А час на запис кластеру займає найбільше часу. Тобто, затрачений час підпорядковується такій закономірності як і було зазначено у формул 3.22

При чому, для SSD технології час затрачений на переміщення зчитуваючої головки наближається до нуля, так як у SSD пристроях реалізована паралельна обробка секторів, та таке визначення як зчитуваюча головка пристрою не має сенсу. Отже результат порівняльного аналізу способів реалізації методу зазначено у таблиці 3.29 [65, 69, 70].

Таблиця 3.29

**Результат порівняльного аналізу способів реалізації модифікованого методу
приховування інформації**

Спосіб реалізації	Необхідний об'єм ОП	Необхідний час	Захищеність від детектування
ПЗОПм	--	-	+
ПчЗОП-Ім	++	--	+
ПчЗОП-ІІм	++	+	-
ПчЗОП-ІІІм	++	++	--

Роблячи висновок за результатами порівняльного аналізу можна стверджувати що:

- для систем де розмір оперативної пам'яті є достатнім (розмір покриваючих файлів цілком уміщається у оперативну пам'ять) переважним способом буде ПЗОПм. Даний спосіб дозволяє досягти мінімального впливу на рівень фрагментації покриваючих файлів, але ПЗОПм потребує значного часу на виконання приховування повідомлення;
- для систем де затрачений час є критичним параметром, то у такому випадку рекомендується використовувати способи ПчЗОП-ІІм та ПчЗОП-ІІІм. ПчЗОП-ІІІм потребує менше часу на приховування повідомлення, але вплив на рівень фрагментації у такий спосіб найбільший;
- для систем у яких захищеність від детектування є критичним параметром рекомендовано використовувати способи ПЗОПм та ПчЗОП-Ім (у залежності від доступного об'єму ОП).

Порівнюючи способи реалізації за базовим методом та за модифікованим методом, можна стверджувати, що:

- необхідний об'єм оперативної пам'яті не впливає на обрання методу приховування, а залежить лише від способу реалізації;
- якщо час на приховування інформації є критичним параметром, то способи модифікованого методу є переважними, так як модифікований метод дозволяє зменшити кількість стеганоблоків та як наслідок у цілому зменшити кількість необхідних перезаписів. Найкращим способом реалізації є ПчЗОП-Шм;
- якщо рівень захищеності повідомлення є важливим параметром то не рекомендується використовувати способи модифікованого методу, а необхідно використовувати ПЗОП, ПчЗОП-І із базового методу.

Висновки до розділу 3

У даному розділі було проаналізовано теоретично та практично пропускну здатність методів у залежності від розміру одного кластера файлу (формула 3.7), від кількості покриваючих файлів (формула 3.5). Також було проаналізовано можливий рівень захищеності методів приховування шляхом перемішування кластерів шляхом аналізу комп'ютерних систем у лабораторіях університету. Як результат було надано зведені дані за рівнем фрагментації у залежності від типу файлів. За даними результатами було зроблено статистичні аналізи які надали змогу оцінити можливий розмір прихованого повідомлення у комп'ютерній системі. Тобто розмір повідомлення яке можна приховати без значного ризику детектувати дане повідомлення. Також для надання часових параметрів роботи методів була використана програмна реалізація «SteganoFAT», дана програма є демонстраційною та не використовує усіх можливих способів і не є оптимальною з алгоритмічної точки зору, але аналізуючи результати роботи програми можна стверджувати що час необхідний на запис повідомлення значно більший ніж на вилучення. Наступним

кроком було надано та теоретично проаналізовано 4 способи використання кожного з методів. Використання кожного із способів призведе до приховування одного й того ж повідомлення, але використовуючи при цьому різну обчислювальну складність та різний рівень фрагментації. Наступним етапом необхідно емпіричне обґрунтування теоретично обчислених результатів аналізу обчислювальної складності.

У рамках даного розділу виконано такі часткові задачі дослідження: удосконалено математичну модель оцінки основних параметрів кластерних стеганосистем та удосконалено методи приховування інформації у структуру кластерних стеганосистем. Також удосконалено математичну модель оцінки кластерних стеганосистем та удосконалено метод приховування інформації у структуру кластерних стеганосистем за рахунок генерації відповідного набору перестановок кластерів, що дозволяє зменшити час приховування інформації.

Отримано наступні практичні значення дисертаційної роботи:

- отримано емпіричні залежності різних показників ефективності стеганографічного перетворення (пропускної здатності, стійкості до несанкціонованого детектування, швидкодії (кількості циклів перезапису) та обчислювальних витрат пам'яті);
- розроблено практичні рекомендації щодо впровадження розроблених методів.

Результати досліджень даного розділу наведено в публікаціях здобувача: [39, 49, 65].

РОЗДІЛ 4

РОЗРОБКА ПРОГРАМНОЇ РЕАЛІЗАЦІЇ МЕТОДІВ ПРИХОВУВАННЯ ІНФОРМАЦІЇ

Для підтвердження розрахованих результатів із попереднього розділу необхідно розробити програмну реалізацію, яка зможе виконувати базовий та модифікований методи приховування даних у структуру файлової системи, виконувати це наглядно та як додатковий результат виводити статистику щодо обчислювальної складності. Так як вже надана демонстраційна програма «SteganoFAT», але вона не відповідає усім необхідним критеріям для даної дисертаційної роботи тож необхідно розробити нову більш інформативну та наглядну реалізацію [19, 49].

Для даної роботи було розроблено програмну симуляцію на мові програмування TypeScript – SteganoSimulation (<https://github.com/ShekhaninKyril/SteganoSimulation>). Від повноцінної роботи із файловою системою було вирішено відмовитись так як:

1. необхідні навички у роботі із мовами програмування для встановлюваних програмних засобів, у свою чергу TypeScript (JavaScript) проста у вивченні мова;
2. результатом роботи із TypeScript є html файл для відображення даних та надання інтерфейсу користувачеві, а також скрипт із усією логікою роботи програми, що у свою чергу не потребує додаткових навичок від користувача так як дана програма може бути запущена безпосередньо через браузер, без необхідної попередньої інсталяції;
3. браузерні програми є кросплатформені, що дозволяє одну й туж програмну реалізацію використовувати на різних операційних системах;
4. один із критеріїв це отримання емпіричних даних щодо обчислювальної складності при виконанні методів, що потребує значної кількості перезаписів даних

кластерів, що у свою чергу може негативно вплинути на фізичний носій інформації (так як кожен носій має амортизаційний параметр), тому краще використовувати симуляцію;

5. час на виконання методів під час симуляції значно швидший аніж при використанні фізичного носія;

6. програмна симуляція дозволяє із більшою вірогідністю виявити помилку у програмному коді та виправити її;

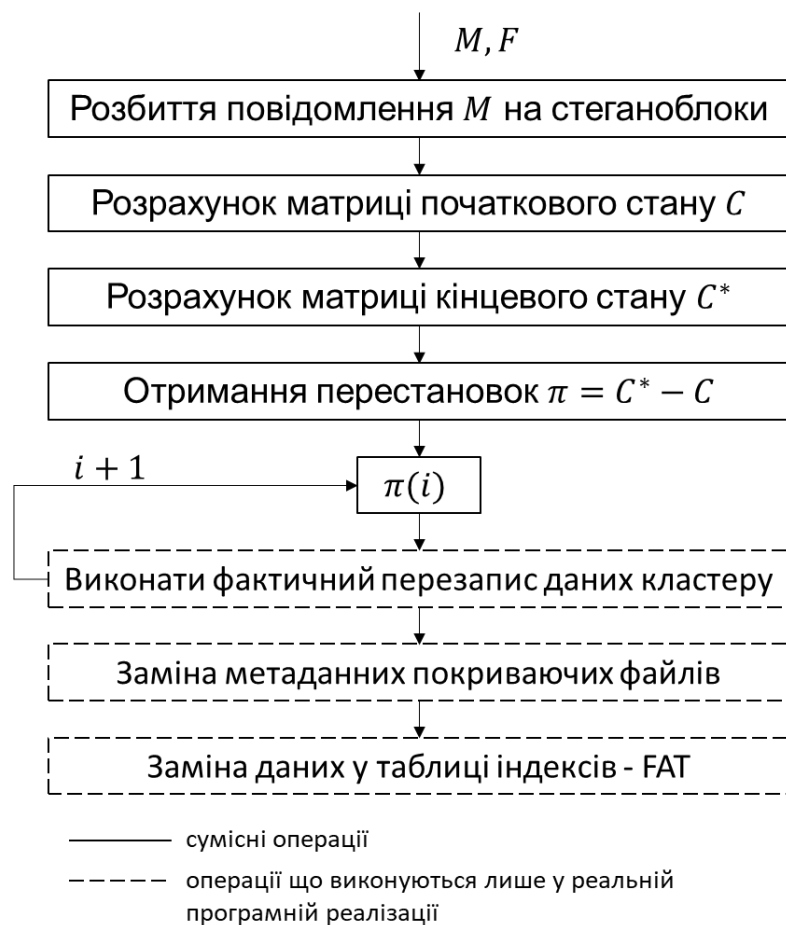


Рис. 4.1 Схематичне зображення відмінностей у роботі програми симуляції у порівнянні із реальною програмою

7. як зображено на рисунку 4.1, основною різницею між симуляцією та програмою що працює із реальним фізичним носієм, з точки зору програмної цінності,

є те що програма що працює із реальним фізичним носієм додатково потребує компоненти для зчитування, запису даних у кластер та обробці безпосередньо FAT таблиць із метаданими. Усі інші компоненти які реалізовані у симуляції можуть бути у подальшому використані для розробки повноцінного програмного забезпечення [25, 27].

4.1 Опис програмної реалізації

Для задоволення вищезазначених критеріїв було вирішено розробити програмну симуляцію. Дана симуляція розроблена мовою програмування JavaScript, а саме її типізованою версією TypeScript. Для відображення інтерфейсу користувача використано мову розмітки HTML, а саме набір бібліотек React який дозволяє конвертувати JavaScript код у HTML вид. Для стилізації використано набір компонентів із бібліотеки Material-UI (що базується на CSS). Таким чином для розробки програмної симуляції необхідні лише навички роботи із React бібліотекою, що не потребує значного часу [70, 71].

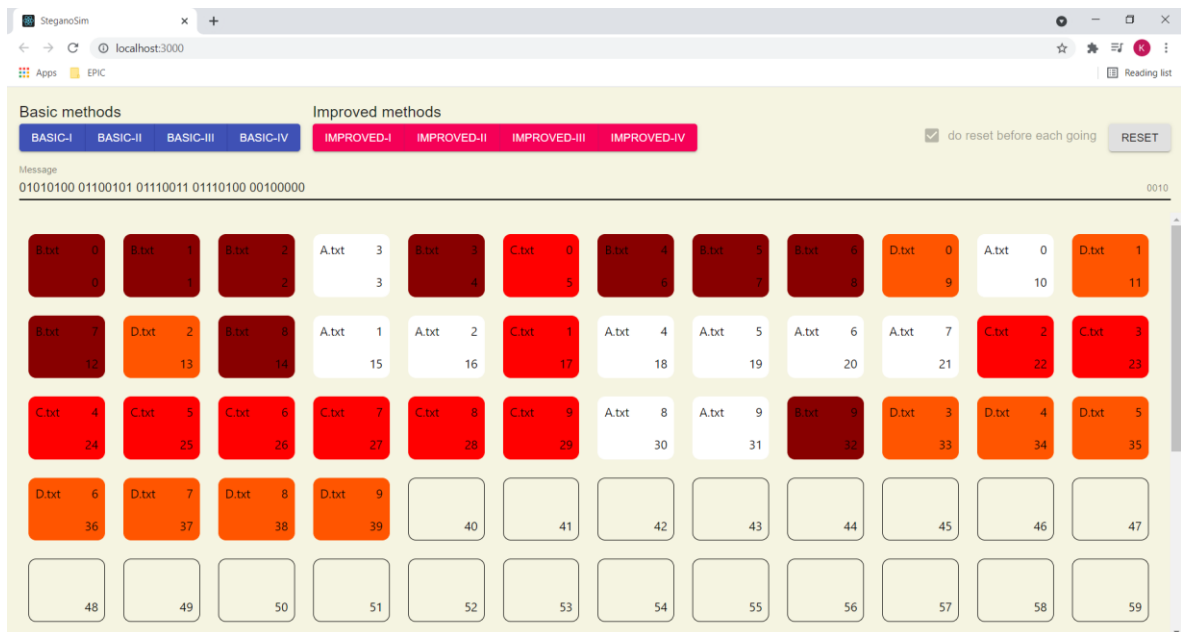


Рис. 4.2 Візуальний інтерфейс програми симуляції

Інтерфейс користувача має вигляд сайту який може бути запущений як і локально так і використовуючи веб-сервер, будь яким браузером. Даний інтерфейс зображено на рисунку 4.2 та має такі частини:

- заголовок (*header*) – частина що знаходиться у шапці сайту та має такі елементи керування: поле для вводу повідомлення (*Message*), відповідні кнопки для обрання одного із способів для виконання базового методу приховування інформації (*Basic-IV*), відповідні кнопки для виконання модифікованого методу (*Improved-IV*), кнопка для встановлення початкового стану (*Reset*);
- основна частина (*body*) – у даній частині представлено стан файлової системи, що виражений набором кластерів. Кожен кластер має відповідні поля, що вказано на рисунку 4.3, а саме: індекс кластеру у межах файлової системи, файл до якого даний кластер належить (якщо кластер належить до файлу то даний кластер фарбується у відповідний колір) та індекс кластеру у межах одного файлу.

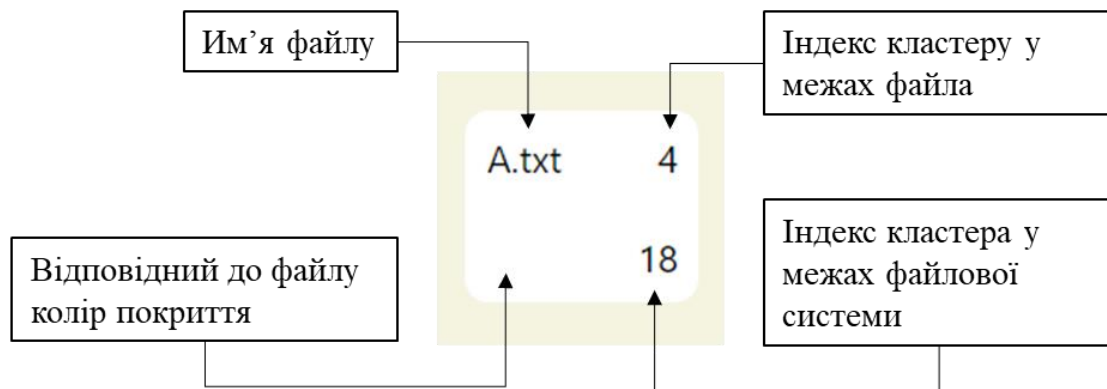


Рис. 4.3 Схематичне тлумачення елементів кластера

- модуль статистики (*Statistics*) – даний модуль збирає інформацію щодо кожного способу приховування інформації, а саме: стан файлової системи до приховування, стан файлової системи після приховування, повідомлення яке було приховано, матриці перестановки, кількість переміщень зчитуваючої головки у кластерах, кількість кластерів що зчитано, кількість записаних кластерів, розмір

оперативної пам'яті одночасно необхідної для виконання методів у кластерах, рівень фрагментації по кожному із покриваючих файлів – до приховування та після приховування. Дана інформація виводиться у логи сайту та може бути скопійована для подальшого аналізу, як вказано на рисунку 4.4.

```
clustersRead: 24
clustersWrite: 24
► fileSystemEnd: {type: 0, clusters: Array(100), clustersInMemory: Array(0), files: Array(4), generateClusters: f, ...}
► fileSystemStart: FileSystem {type: 0, clusters: Array(100), clustersInMemory: Array(0), files: Array(4), generateClusters: f, ...}
► fragmentations: (4) [{...}, {...}, {...}, {...}]
headerMoves: 289
memorySize: 2
message: "Test "
▼ permutation: Array(19)
► 0: (2) [0, 10]
► 1: (13) [1, 15, 7, 21, 17, 12, 2, 16, 8, 30, 9, 31, 11]
► 2: [3]
► 3: (9) [4, 18, 14, 6, 20, 5, 19, 32, 13]
► 4: [22]
► 5: [23]
► 6: [24]
► 7: [25]
► 8: [26]
► 9: [27]
► 10: [28]
► 11: [29]
► 12: [33]
► 13: [34]
► 14: [35]
► 15: [36]
► 16: [37]
► 17: [38]
► 18: [39]
```

Рис. 4.4 Приклад результату роботи модуля статистики

Для того щоб задати початковий стан файлової системи необхідно у файлі json кількість покриваючих файлів та їх параметри. Можна задавати такі параметри: ім'я файлу (*name*), розмір файлу у кластера (*sizeInClusters*) та відповідний колір (*color*). Покриваючи файли будуть створені у порядку задання із відповідним розміром у кластерах та розміщенні впорядковано один за одним (без надлишкової фрагментації).


Повідомлення може бути задано використовуючи відповідне поле у шапці сайту, дане поле може працювати у двох режимах: текстовий режим, бінарний режим. У текстовому режимі введені символи опрацьовуються як *string* значення. У разі якщо обрано бінарний режим то кожен символ опрацьовується як *boolean* значення. При перемиканні одного режиму у інший той введені значення конвертується у

відповідне. Також якщо обрано бінарний режим то використовується додаткова функція валідації перед додавання нового значення, яка забороняє введення усіх зайвих символів окрім «0» та «1», а також кожен восьмий символ це розділяючий символ – пробіл, що візуально допомагає розбити значення по байтам.

Використовуючи дані елементи керування користувач має можливість задавати набір покриваючих файлів, встановлювати повідомлення яке необхідно приховати, наглядно оцінити зміни станів файлової системи до приховування повідомлення та після, отримати статистичні дані щодо використаного способу методу приховування інформації у структуру файлової системи.

Алгоритм роботи із даним сайтом наступним:

1. користувач, через JSON файл, задає параметри файлової системи, кількість та конфігурацію покриваючих файлів;
2. користувач обирає режим вводу повідомлення – текстовий чи бінарний;
3. користувач вводить необхідне повідомлення;
4. користувач застосовує метод приховування повідомлення відповідним способом натискаючи на відповідну кнопку у шапці сайту;
5. якщо усі необхідні умови виконані, файлова система змінить свій стан таким чином щоб приховати повідомлення, що відобразиться у певній послідовності кластерів у основній частині сайту;
6. якщо кількість кластерів покриваючих файлів не відповідає кількості відповідних стеганоблоків, то користувач побачить повідомлення про помилку із додатковою таблицею на якій вказано кластерів якого файлу не вистачає, рисунок 4.5;
7. відкривши логи користувач побачить зібрану статистику для подальшого аналізу;



File names	Steganoblocks	Clusters
A.txt	16	10
B.txt	11	10
C.txt	3	10
D.txt	10	10

Рис. 4.5 Помилка що відображається у разі зовеликого повідомлення

8. користувач може обрати наступний метод натиснувши на відповідну кнопку (перейти до пункту 4) і так далі.

Схематично дана схема зображена на рисунку 4.6.



Рис. 4.6 Схематичне зображення алгоритму роботи із програмою симуляцією

4.2 Розробка програмної реалізації

Дана програма розроблена мовою програмування TypeScript у середі розробці Visual Studio Code, що є вільною до користування середою програмування. Програмна реалізація умовно складається із наступних модулів: модуль візуалізації, модуль збереження інформації, модуль обробки інформації. У рамках даної дисертаційної роботи наукове значення мають модуль збереження та модуль обробки інформації так як містять алгоритмічне представлення описаних вище методів [72, 73, 74].

4.2.1 Модуль збереження інформації

У рамках наступного модуля були реалізовані наступні класи: Файлової системи (*FileSystem*), Файлу (*File*), Кластеру (*Cluster*), Статистики (*Statistics*).

Клас файлової системи представляє собою елемент, що об'єднує у собі усі інші класи, та надано у програмі у одиничному екземплярі. Має такі обов'язкові поля:

- clusters: Cluster[] – масив усіх кластерів файлової системи;
- files: File[] – набір покриваючих файлів.

Повний опис класу файлової системи надано у додатку Б – Опис класу файлової системи.

Клас файлу містить у собі конфігураційні параметри покриваючих файлів, реалізовує низькорівневу логіку роботи із кластерами (задати-видалити кластери) та має такі поля:

- name: string – ім'я файлу, може бути будь-якою строкою, необхідно лише для відповідного відображення значення кластерах, що полегшує користувацький інтерфейс;
- fs: FileSystem – об'єкт що посилається на файлову систему до якої належить даний файл;

- `color: string` – текстове значення кольору, задається у шістнадцятиричному вигляді (`0xFFFFFFFF`), у даний колір буду розфарбовані усі відповідні кластери.

- `clusters: Cluster[]` – масив із кластерів які належать до даного файлу, дані кластери пов'язані із набором кластерів у файловій системі, тобто якщо буде змінено значення у кластері цього файлу, то це відобразиться й у масиві усіх кластерів.

Повний опис даного класу надано у додатку В – Опис класу файлу.

Клас кластеру імплементує найменшу логічну одиницю у роботі даної програми та являє собою сутність, що зберігає у собі наступні поля:

- `fs: FileSystem` – об'єкт, що містить посилання на усю файлову систему;
- `fsIndex: number` – число, що відповідає позиції кластера у наборі усіх кластерів файлової системи, іншими словами – це індекс кластеру у межах усієї файлової системи;

- `file?: File` – об'єкт що відповідає файлу до якого даний кластер належить, дане поле необов'язкове так як можуть бути кластери що не мають приналежності до покриваючого файлу;

- `fileIndex?: number` – число, що відповідає позиції кластера у наборі кластерів обраного файлу, поле може бути пустим, якщо даний кластер не має покриваючого файлу.

Повний опис даного класу надано у додатку Г – Опис класу кластеру.

Клас статистики найбільший клас так як поєднує у собі елементи із модуля логіки, тобто робота по збереженню інформації та по обробці інформації виконується у рамках одного класу. Даний клас умістить у собі поля необхідні для оцінки обчислювальної складності того чи іншого способу приховування інформації. Поля у даному класі мають наступний вигляд:

- `fileSystemStart?: FileSystem` – об'єкт що посилається на стан файлової системи до виконання методів приховування інформації;

- `fileSystemEnd?: FileSystem` – об'єкт що посилається на стан файлової системи після виконання методів приховування інформації;

- `message?: string` – значення повідомлення яке було приховано;
- `permutation?: number[][]` – масив масивів із чисел, що відповідають повному набору перестановок необхідних для отримання кінцевого стану файлової системи (як приклад `[[1,3,2], [4], [5,8], ...]`);
- `headerMoves?: number` – значення що відповідає кількості переміщень головки зчитуваючого пристрою фізичного носія інформації, розраховується шляхом підсумовування відстаней (індексами) між кластерами які необхідно перемістити у відповідності до набору перестановок;
- `clustersRead?: number` – відповідає значенню кількість кластерів які були зчитані до оперативної пам'яті, розраховується шляхом підрахунку кількості кластерів що приймають участь у перестановці;
- `clustersWrite?: number` – відповідне значення перезаписаних кластерів (дорівнює `clustersRead`), є найзатратнішою операцією у оцінці складності;
- `memorySize?: number` – умовний необхідний розмір оперативної пам'яті для успішного виконання методів приховування, для способів ПЗОП та ПЗОПм залежить від кількості кластерів які були переміщенні, у інших випадках значення є константою та дорівнює 2;
- `fragmentations?: { before: number; after: number; }[]`; – набір даних які містять значення рівня фрагментації до та після використання методу по кожному покриваючому файлу. За допомогою цього параметру можна оцінити, як фактично кожен метод впливає на кінцевий рівень фрагментації [73, 75].

Повний опис класу надано у додатку Г – Опис класу статистики.

4.2.2 Модуль обробки інформації

До даного модулю належать такі основні функції як: функції обробки повідомлення у стеганоблоки, робота із перестановкою (отримання перестановки,

застосування перестановки), стеганографічні способи базового методу, стеганографічні способи модифікованого методу [76, 77].

Частина, що стосується обробки повідомлення у стеганоблоки має значну кількість допоміжних функцій, але основними є:

- `getSteganoMessage` – на вхід приймає повідомлення та файлову систему, наступним кроком із кількості покриваючих файлів, що належать до файлової системи розраховується розмір стеганоблока. Після чого повідомлення конвертується у бінарний вигляд, яке вже слідом “нарізується” на стеганоблоки. Важливою приміткою є те що, якщо останній блок неповний то він доповнюється нульовими значеннями, щоб усі стеганоблоки мали рівну довжину. Результатом даної функції є стеганограма – масив із чисел які відповідають індексу файлу кластер якого повинен бути записано наступним.

- `getSteganoMessageImproved` – функція подібна до попередньої, але також додатково повертає масиви із нульовим або одиничними значеннями. Кожен такий масив відповідає файлу та значення такого масиву відповідають інформації яка повинна бути прихована саме модифікованим методом. Це досягається за рахунок того, що збільшено на 1 розмір кожного стеганоблоку, але потім кожен стеганоблок розбивається на базову частину та модифіковану ($[M^B, M^M] = M$).

Повний опис даних функцій надано у додатку Д – Функції обробки повідомлення у стеганоблоки.

Частина, що стосується роботі із перестановкою необхідна для отримання набору перестановок та застосування даного набору до файлової системи. Основними функціями є:

- `getPermutation` – дана функція приймає початковий стан та кінцевий стан і як результат повертає набір перестановок які необхідно виконати для отримання кінцевого стану (що безпосередньо використовується методами приховування інформації). Необхідно зазначити що у роботі використовуються стани не з повних

Частина що стосується роботі із функціями базового методу приховування реалізовує наступні способи приховування інформації до структури файлової системи: ПЗОП, ПчЗОП-I, ПчЗОП-II, ПчЗОП-III. Що відповідає наступним функціям:

- I_Basic – даний метод використовує повідомлення та файлову систему у якості параметрі, та як результат повертається мутована файлова система, що відповідає прихованому повідомленню. По-перше у даній функції, виконується заповнення кластерів нового стану файлової системи у відповідності до стеганоблоків, а вже після цього файли доповнюють до такої ж довжини як і у початковому стані файлової системи;

- II_Basic – з функціональної точки зору виконується такий же спосіб як і й I_Basic, але при цьому уся робота виконується через набори перестановок, що відповідає способу ПчЗОП-I (із почерговим завантаження кластерів до оперативної пам'яті);

- III_Basic – дана функція приймає також само параметри, та виконує таку ж логіку як і II_Basic, але додатково перевіряє чи було поточний кластер використано у приховуванні повідомлення. Якщо кластер не був використаний, тоді його позиція ніяк не зміниться, що відповідає умові ПчЗОП-I – приховування повідомлення із переміщенням лише кластерів які приймали участь у приховуванні.

- IV_Basic – дана функція доповнює попередню (III_Basic) ти, що при приховуванні повідомлення беруть участь не лише послідовні кластери, але й такі, що в цілому відповідають стеганоблоку. У порівнянні результатів роботи IV_Basic та III_Basic можна побачити, що кольори кластерів будуть ідентичні, але можуть виникати випадки, коли індекси кластерів у межах файлів будуть різними.

Повний опис даних методів надано у додатку Є – Функції базового методу приховування.

Частина що стосується стеганографічних способів приховування інформації модифікованим методом містить функції, що відповідають способам: ПЗОПм,

ПчЗОП-Ім, ПчЗОП-ІІм, ПчЗОП-ІІІм, а також додатково функцію що виконує перестановку на рівні кластерів одного файлу. Загалом реалізовані такі функції:

- `replaceClustersImproved` – дана функція приймає у якості параметрів на вхід кластери та модифіковану частину стеганоблоків відповідного файлу. Як результат кластери змінюють свої індекси у межах файлу у відповідності до стеганоблоків. Оцінюється масив усіх можливих індексів файлу – F^i також повідомлення розбивається на серії із «0» та «1», після чого опрацьовується кожна серія. У рамках однієї серії оцінюється знак даної серії, якщо знак «1» то поточний кластер приймає останній індекс файлу із множини F^i . А із масиву видаляється останній елемент. Якщо ж знак серії «0» – то кластер приймає перший індекс, а із масиву F^i видаляється перший елемент. І так поки не буде опрацьована кожна серія. Якщо ж серії закінчилися, а у масиві F^i ще лашилися елементи то вони переносяться почергово у індекси кластерів, так начебто опрацьовується серія із «0» значенням елементів;

- `I_Improved` – функція подібна до `I_Basic` за винятком того, що після виконання основної логіки, виконується функція `replaceClustersImproved`. Тобто, спочатку виконується перестановка за базовим методом, а вже після виконується перестановка модифікованої частини стеганоблоків окремо над кожним файлом;

- `II_Improved` – подібний до `I_Improved` із виконанням базової частини описаній у `II_Basic`;

- `III_Improved` – аналогічно до вищевказаних функцій;

- `IV_Improved` – аналогічно.

Повний опис заданих функцій надано у додатку Ж – Функції модифікованого методу приховування.

4.3 Емпірична оцінка обчислювальної складності методів приховування інформації

У рамках даного підрозділу буде проведено емпіричну оцінку методів приховування інформації у структуру файлової системи шляхом перемішування кластерів покриваючих файлів такими способами: ПЗОП, ПчЗОП-I, ПчЗОП-II, ПчЗОП-III та ПЗОПм, ПчЗОП-Ім, ПчЗОП-ІІм, ПчЗОП-ІІІм. Способи будуть оцінені за наступними критеріями [65, 78, 79, 80]:

- $f(n)$ – кількість переміщень зчитуваючої головки;
- $g(n)$ – кількість зчитаних кластерів;
- $h(n)$ – кількість записаних кластерів;
- $m(n)$ – необхідний розмір оперативної пам'яті.

Також буде оцінено вплив способів на рівень фрагментації кожного із покриваючих файлів. Для більш об'єктивної оцінки будемо досліджувати способи із різними вихідними параметрами. Нехай будемо змінювати кількість покриваючих файлів як 2, 4, 8. Також оцінимо вплив методів при різних розмірах покриваючих файлів – по 10, 20, 50 кластерів кожен файл. Оцінимо вплив у залежності від різного розміру повідомлення – 10, 20, 50 байт. Також необхідно оцінити формат повідомлення – перевіримо читаємий текст та псевдовипадкову послідовність.

Для оцінки впливу кількості файлів на обчислювальну складність, зафіксуємо розмір кожного файлу у 20 кластерів, розмір повідомлення нехай буде 5 байт (максимально можливий розмір при використанні 2 файлів) та будемо змінювати кількість покриваючих файлів у межах 2, 4, 8 файлів. Оцінку візьмемо як середнє між 5 спробами із різним повідомленнями для кожного із способів. Результати вказані у таблиці 4.1 для базового методу та 4.2 для модифікованого.

Таблиця 4.1

Емпіричні значення аналізу у залежності до різної кількості покриваючих файлів для базового методу

Параметр ОС	ПЗОП			ПчЗОП-I			ПчЗОП-II			ПчЗОП-III		
	2	4	8	2	4	8	2	4	8	2	4	8
$f(n)$	74	132	257	335	902	1676	335	896	1637	334	896	1630
$g(n)$	36	66	128	36	66	128	36	33	26	27	27	25
$h(n)$	36	66	128	36	66	128	36	33	26	27	27	25
$m(n)$	36	66	128	2	2	2	2	2	2	2	2	2

Оцінюючи результати таблиці можна стверджувати що емпіричні дані підтвердили теоретичні. Особливо важливо відмітити, що для способів ПчЗОП-II/III кількість кластерів, що була прочитана, записана дійсно залежить від кількості стеганоблоків, а не від кількості кластерів, як це відбувається у ПЗОП та ПчЗОП-I.

Оцінюючи рівень фрагментації можна стверджувати, що рівень збільшується при використанні ПчЗОП-II/III, що відповідає теоретичним очікуванням.

Таблиця 4.2

Емпіричні значення аналізу у залежності до різної кількості покриваючих файлів для модифікованого методу

Параметр ОС	ПЗОПм			ПчЗОП-Iм			ПчЗОП-IIм			ПчЗОП-IIIм		
	2	4	8	2	4	8	2	4	8	2	4	8
$f(n)$	77	160	186	330	1012	1180	330	1124	1286	330	1140	1271
$g(n)$	39	80	93	39	80	93	39	80	72	39	80	72
$h(n)$	39	80	93	39	80	93	39	80	72	39	80	72
$m(n)$	39	80	93	2	2	2	2	2	2	2	2	2

Оцінюючи отримані результати необхідно зазначити, що загалом теоретична оцінка обчислювальної складності не відкидається, але кожного разу кількість перезаписаних кластерів була близькою до повної кількості кластерів покриваючих файлів. Це пояснюється тим що після виконання приховування базової компоненти стеганоблока, усе одно виконується перемішування кластерів використовуючи

модифіковану компоненту що и призводить до збільшення кількості перезаписаних кластерів. Наразі необхідно вдосконалити механізм приховування модифікованої компоненти, щоб можна було би знехтувати зайвими переміщеннями. На теперішній час можна рекомендувати використовувати модифікований метод приховуванні інформації із максимальним навантаження, щоб збільшити коефіцієнт корисної дії.

Аналізуючи рівні фрагментації можна стверджувати, що рівень фрагментації майже не залежить від обраного способу приховування, а більше залежить від структури прихованого повідомлення.

Далі оцінено вплив розміру покриваючого файлу на обчислювальну складність, для цього зафіксуємо кількість покриваючих файлів як 4, а розмір повідомлення як 5 байт. Та будемо змінювати кількість кластерів на кожен покриваючий файл у межах 10,20,50. Результати занесено у таблицю 4.3 для базового методу та таблицю 4.4 для модифікованого методу приховування інформація.

Таблиця 4.3

Емпіричні значення аналізу у залежності до різного розміру покриваючих файлів для базового методу

Параметр ОС	ПЗОП			ПчЗОП-I			ПчЗОП-II			ПчЗОП-III		
	10	20	50	10	20	50	10	20	50	10	20	50
$f(n)$	66	128	308	321	872	2503	309	851	2501	309	851	2501
$g(n)$	33	64	154	33	64	154	28	36	36	26	32	32
$h(n)$	33	64	154	33	64	154	28	36	36	26	32	32
$m(n)$	33	64	154	2	2	2	2	2	2	2	2	2

Оцінюючи результати таблиці можна стверджувати що емпіричні дані підтвердили теоретичні. Знову для способів ПчЗОП-II/III кількість кластерів, що була прочитана, записана дійсно залежить від кількості стеганоблоків, а не від кількості кластерів, як це відбувається у ПЗОП та ПчЗОП-I, що у котрий раз доводить ефективність даних способів для базового методу приховування інформації. Також при збільшенні розміру покриваючого файлу, кількість перезаписаних кластерів

майже не змінилася. Лише збільшилась відстань яку треба пройти зчитуваючій головці від кластера до кластера. Оцінюючи рівень фрагментації можна стверджувати, що рівень збільшується при використанні ПчЗОП-ІІ/ІІІ, що відповідає теоретичним очікуванням. Оцінюючи вплив розміру покриваючого файлу на вихідний рівень фрагментації, можна стверджувати, що відслідковується пропорційна залежність щодо обраного способу, але вплив мінімальний.

Оцінюючи результати таблиці 4.4 можна стверджувати, що емпіричні результати задовольняють теоретично отриманим. Але також необхідно зазначити, що переваги у використанні ПчЗОП-ІІ/ІІІм пере ПчЗОП-Ім не спостерігається навпаки кількість переміщень зчитуваючої головки при використанні способу ПчЗОП-Ім навіть менша. Щодо оцінки рівня фрагментація то залежності від використовуваного способу не спостерігається. Є лише пропорційна залежність від розміру покриваючого файлу.

Таблиця 4.4

Емпіричні значення аналізу у залежності до різного розміру покриваючих файлів для модифікованого методу

Параметр ОС	ПЗОПм			ПчЗОП-Ім			ПчЗОП-ІІм			ПчЗОП-ІІІм		
	10	20	50	10	20	50	10	20	50	10	20	50
$f(n)$	80	160	380	358	1013	2743	349	1171	3653	349	1171	3307
$g(n)$	40	80	190	40	80	190	40	80	200	40	80	200
$h(n)$	40	80	190	40	80	190	40	80	200	40	80	200
$m(n)$	40	80	190	2	2	2	2	2	2	2	2	2

Далі необхідно оцінити вплив способів на обчислювальну складність у залежності від розміру приховуваного повідомлення. Для цього будемо змінювати розмір повідомлення у межах 10, 20, 50 байт. Та зафіксуємо кількість покриваючих файлів як 4 файли та розмір кожного покриваючого файлу у 60 кластерів (50 кластерів на файл мінімально можливий розмір файлу для того щоб приховати 50 байт у 4 покриваючих файли. Але треба враховувати неоднорідність повідомлення, тобто

деякий вид стеганоблоку може зустрічатися частіше ніж інші). Результати аналізу данної залежності вказані у таблиці 4.5 для базового методу та таблиці 4.6 для модифікованого методу [65].

Таблиця 4.5

Емпіричні значення аналізу у залежності до різного розміру приховуваного повідомлення для базового методу

Параметр ОС	ПЗОП			ПчЗОП-I			ПчЗОП-II			ПчЗОП-III		
	10	20	50	10	20	50	10	20	50	10	20	50
$f(n)$	376	388	434	6348	9824	12К	6227	9707	12К	6227	9837	12К
$g(n)$	188	194	217	188	194	217	73	123	201	66	108	175
$h(n)$	188	194	217	188	194	217	73	123	201	73	108	175
$m(n)$	188	194	217	2	2	2	2	2	2	2	2	2

Оцінюючи результати таблиці 4.5 можна стверджувати, теоретична оцінка обчислювальної складності для способів приховування інформації базовим методом підтверджена емпіричним шляхом. Також можна зазначити, що для ПЗОП та ПчЗОП-I кількість перезаписаних кластерів майже не відрізняється від розміру приховуваного повідомлення. Невідмінну від ПчЗОП-II/III, при виконанні яких значно помітніша різниця у кількості перезаписаних кластерів у залежності від розміру приховуваного повідомлення (73 кластерів для 10 байт, 123 – 20 байт, 201 – 50 байт). Таким чином ПчЗОП-II/III має значні переваги перед ПЗОП та ПчЗОП-I при приховуванні повідомлення значно меншого за максимально допустимий розмір. Щодо оцінки фрагментації то можна зазначити що при використанні ПчЗОП-II/III рівень фрагментації буде трохи більший аніж при використанні ПЗОП та ПчЗОП-I. Але ця різниця мінімальна у порівнянні залежності рівня фрагментації від розміру приховуваного повідомлення.

Таблиця 4.6

Емпіричні значення аналізу у залежності до різного розміру приховуваного повідомлення для модифікованого методу

Параметр ОС	ПЗОПм			ПчЗОП-Ім			ПчЗОП-ІІм			ПчЗОП-ІІІм		
	10	20	50	10	20	50	10	20	50	10	20	50
$f(n)$	480	480	480	5397	9300	15K	5900	9257	15K	5619	9368	15K
$g(n)$	240	240	240	240	240	240	220	230	240	220	230	240
$h(n)$	240	240	240	240	240	240	220	230	240	220	230	240
$m(n)$	240	240	240	2	2	2	2	2	2	2	2	2

Аналізуючи результат із таблиці 4.6 можна стверджувати використання модифікованого методу нівелює переваги ПчЗОП-ІІ/ІІІм (що ще раз вказує на недосконалість розробленого алгоритму для способів ПчЗОП-ІІ/ІІІм). Але в цілому розрахункова складність отримана емпіричним шляхом у цілому відповідає отриманій теоретично, але не по кожному пункту. З точки зору рівня фрагментації використання різних способів не майже не змінює рівня фрагментації (а іноді при використанні ПчЗОП-ІІІм отриманий рівень фрагментації буде меншим за рівень фрагментації при ПчЗОП-ІІм).


Також оцінимо вплив формату повідомлення на оцінку обчислювальної складності, для цього зафіксуємо кількість покриваючих файлів як 4, розмір повідомлення як 50 байт та розмір кожного покриваючого файлу як 60 кластерів. Та виконаємо приховування таких повідомлень [65]:

- читаємий текст (для прикладу уривок із «Lorem Ipsum»);
- псевдовипадкову згенеровану послідовність заданої довжини.

Результати аналізу занесено до таблиці 4.7 для способів базового методу та до таблиці 4.8 для способів модифікованого методу відповідно.

При спробі приховати повідомлення «Lorem ipsum dolor sit amet, consectetur adipiscing» (що відповідає 50 байт форматowanego тексту), виявилось що заданий розмір покриваючих файлів не здатен виконати приховування повідомлення. Так як кількість стеганоблоків із значення «01» значно перевищує інший тип стеганоблоків,

як зазначено на рисунку 4.8. Саме тому довелося змінити повідомлення на інше читаєме повідомлення.



File names	Steganoblocks	Clusters
1.txt	45	60
2.txt	72	60
3.txt	44	60
4.txt	39	60

Рис. 4.8 Попередження при приховуванні повідомлення «Lorem ipsum» у незадовільну за параметрами файлової систему

Таблиця 4.7

Емпіричні значення аналізу у залежності від типу приховуваного повідомлення для базового методу

Параметр ОС	ПЗОП		ПчЗОП-I		ПчЗОП-II		ПчЗОП-III	
	Текст	ПвП	Текст	ПвП	Текст	ПвП	Текст	ПвП
$f(n)$	466	448	15K	15K	15K	15K	15K	15K
$g(n)$	233	224	233	224	233	224	209	192
$h(n)$	233	224	233	224	233	224	209	192
$m(n)$	233	224	2	2	2	2	2	2

Аналізуючи результат із таблиці 4.7 можна стверджувати, що теоретична оцінка обчислювальної складності була підтверджена емпірично. Необхідно зауважити що у переваги при використанні способів приховування інформації ПчЗОП-II/III для форматowanego тексту майже такі самі як і при приховуванні псевдовипадкової послідовності. Але при використанні ПвП кількість кожного виду стеганоблоків буде розподілена рівномірно, що у цілому збільшує максимально допустимий розмір

приховуваного повідомлення. З точки зору оцінки фрагментації необхідно зазначити що при приховуванні форматowanego тексту може виникнути аномальний рівень фрагментації для певних покриваючих файлів. Відмінно від приховування псевдовипадкової послідовності – у результаті рівні фрагментації покриваючих файлів розподілені більш рівномірно, що позитивно впливає на захищеність повідомлення до детектування, як зазначено у розділі 3.

Таблиця 4.8

Емпіричні значення аналізу у залежності від типу приховуваного повідомлення для модифікованого методу

Параметр ОС	ПЗОПм		ПчЗОП-Ім		ПчЗОП-ІІм		ПчЗОП-ІІІм	
	Текст	ПвП	Текст	ПвП	Текст	ПвП	Текст	ПвП
$f(n)$	476	480	17К	16К	17К	16К	17К	16К
$g(n)$	238	240	238	240	238	240	238	240
$h(n)$	238	240	238	240	238	240	238	240
$m(n)$	238	240	2	2	2	2	2	2

Аналізуючи результати із таблиці 4.8 можна стверджувати, що обчислювальна складність майже не залежить від обраного способу приховування інформації. Також необхідно зазначити, що для способів приховування інформації модифікованим методом майже не має різниці щодо формату повідомлення – обчислювальна складність при приховуванні форматowanego текст майже така сама як і при приховуванні псевдовипадкової послідовності. Але з іншого боку рівень фрагментації так само однаково рівномірний як при приховуванні тексту так і при приховуванні ПвП. І в цілому рівень фрагментації нижчий чим при використанні базового методу.

Узагальнюючи результати отримані емпіричним шляхом (таблиці 4.1 – 4.8), можна зробити наступний висновок. Для способів приховування інформації базовим методом емпірично отримана обчислювальна складність відповідає теоретично розрахованій. Особливо необхідно виділити способи ПчЗОП-ІІ та ПчЗОП-ІІІ так як кількість перезаписаних кластерів значно нижча ніж при використанні ПЗОП та

ПчЗОП-I. Але з іншого боку, практично отримана обчислювальна складність способів приховування інформації модифікованим методом не повністю відповідає теоретично розрахованій. А саме при використанні ПчЗОП-II/III, кількість зчитаних та записаних кластерів залежить від кількості кластерів покриваючих файлів – n , а не від кількості стеганоблоків – k . Узагальнююча оцінка обчислювальної складності вказана у таблиці 4.9, де:

- $O(f)$ – обчислювальна складність на переміщення зчитуваючої головки фізичного носія (для HDD накопичувачів), чи зміна позиції робочого сектору (для SSD накопичувачів);
- $O(g)$ – обчислювальна складність на зчитування даних із сектору;
- $O(h)$ – обчислювальна складність на запис даних у сектор;
- $O(m)$ – кількість необхідного об'єму оперативної пам'яті обчислювальної системи.
- n – загальна кількість кластерів покриваючих файлів;
- k – розмір приховуваного повідомлення у стеганоблоках.

Отже як видно із таблиці 4.9 кількість перезаписаних кластерів для способів ПчЗОП-II та ПчЗОП-III у даній програмній реалізації не відповідає теоретично розрахованим. Частково це можна вирішити реалізувавши вдосконалену функцію приховування модифікованої компоненти стеганоблоку (по аналогії із функцією *replaceClustersImproved*). Для цього при розрахунку та перемішуванні кластерів у відповідності до базової компоненти треба помічати кластери які не були переміщені, та за можливістю зберігати їх позиції при виконанні перемішування кластерів за допомогою модифікованої компоненти. Також необхідно виконувати такий алгоритм рекурсивно – якщо на поточній ітерації приховати необхідне повідомлення неможливо із збереженням позицій усіх кластерів які необхідно зберегти, то необхідно знехтувати одним таким кластером. Та знову спробувати приховати повідомлення. І так далі до поки не вийде приховати усю необхідну інформацію.

Таблиця 4.9

Порівняння теоретично розрахованої обчислювальної складності із емпірично отриманою, при використанні різних способів

Параметр ОС (теоретичне / емпіричне)		$O(f)$	$O(g)$	$O(h)$	$O(m)$
Базовий метод	ПЗОП	$O(f(n))$ $/O(f(n))$	$O(g(n))$ $/O(g(n))$	$O(h(n))$ $/O(h(n))$	$O(m(n))$ $/O(m(n))$
	ПчЗОП-I	$O(f(n^2))$ $/O(f(n^2))$	$O(g(n))$ $/O(g(n))$	$O(h(n))$ $/O(h(n))$	$O(m(2))$ $/O(m(2))$
	ПчЗОП-II	$O(f(n^2))$ $/O(f(n^2))$	$O(g(2k))$ $/O(g(2k))$	$O(g(2k))$ $/O(g(2k))$	$O(m(2))$ $/O(m(2))$
	ПчЗОП-III	$O(f(n^2))$ $/O(f(n^2))$	$O(g(2k))^-$ $/O(g(2k))^-$	$O(h(2k))^-$ $/O(h(2k))^-$	$O(m(2))$ $/O(m(2))$
Модифікований метод	ПЗОПм	$O(f(n))$ $/O(f(n))$	$O(g(n))$ $/O(g(n))$	$O(h(n))$ $/O(h(n))$	$O(m(n))$ $/O(m(n))$
	ПчЗОП-Iм	$O(f(n^2))$ $/O(f(n^2))$	$O(g(n))$ $/O(g(n))$	$O(h(n))$ $/O(h(n))$	$O(m(2))$ $/O(m(2))$
	ПчЗОП-IIм	$O(f(n^2))$ $/O(f(n^2))$	$O(g(2k))$ $/O(g(n))$	$O(g(2k))$ $/O(g(n))$	$O(m(2))$ $/O(m(2))$
	ПчЗОП-IIIм	$O(f(n^2))$ $/O(f(n^2))$	$O(g(2k))^-$ $/O(g(n))$	$O(h(2k))^-$ $/O(g(n))$	$O(m(2))$ $/O(m(2))$

У най оптимістичному випадку це дозволить методам ПчЗОП-IIм та ПчЗОП-IIIм мати таку ж розрахункову складність як і теоретично розраховану. У найгіршому ж випадку обчислювальна складність буде такою як емпірично отримана вище, що зазначена у таблиці 4.9. А отже ми нічого не втрачаємо, так як розрахунок перестановки майже нічого не коштує у порівнянні із перезаписами кластерів. Приклад роботи такої вдосконаленої функції зображено на рисунку 4.9.

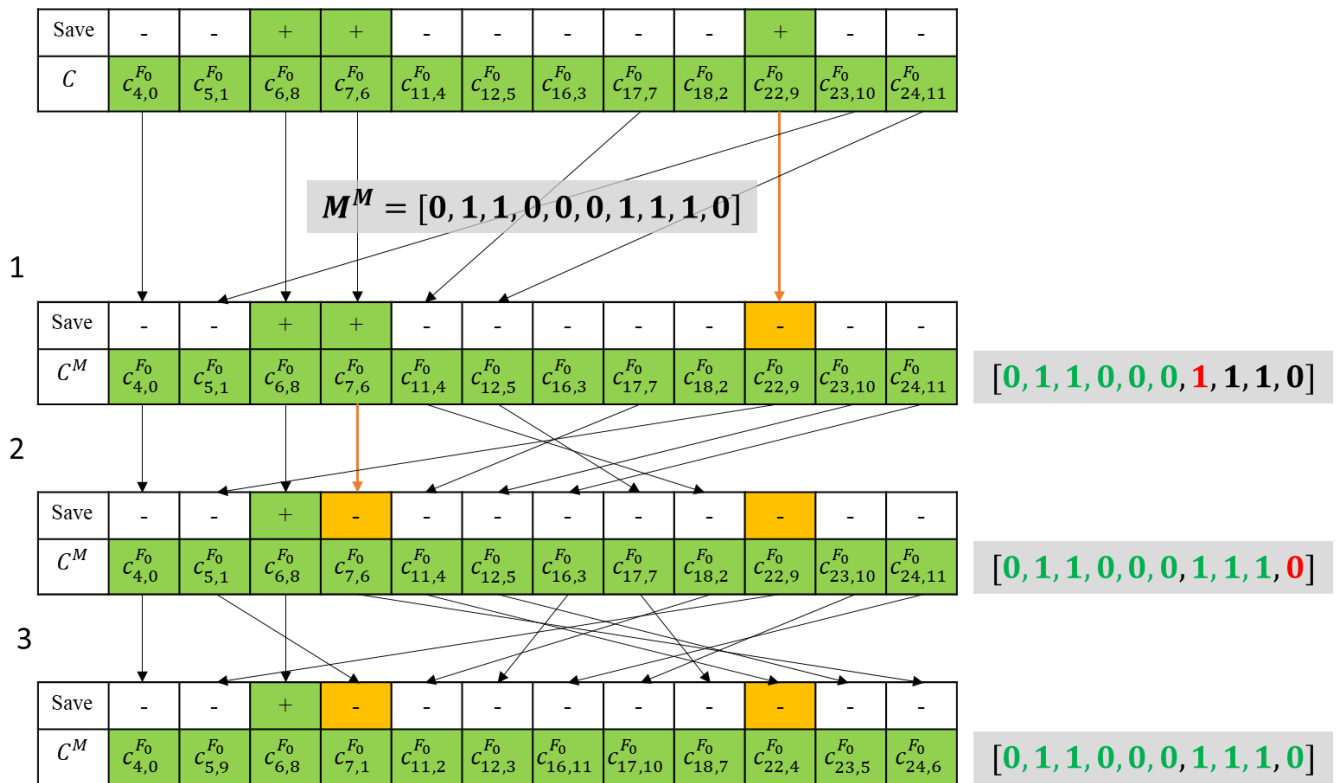


Рис. 4.9 Приклад роботи вдосконаленого алгоритму приховування модифікованої компоненти стеганоблоку

Також необхідно оцінити вплив способів приховування методів ПЗОП, ПчЗОП-I/II/III та ПЗОПм, ПчЗОП-I/II/IIIм на рівень фрагментації покриваючих файлів. Для цього проаналізуємо дані, що були зібрані та використані у таблицях 4.1 – 4.8. Необхідно опрацювати рівень фрагментації із кожного досліджуваного блоку. Для оцінки саме впливу способів на рівень фрагментації, рівень фрагментації по кожному з файлів було нормовано, та розраховано середній рівень по кожному з досліджуваних блоків (відповідних таблицям 4.1 – 4.8). Далі отримані середні рівні фрагментації були об'єднані у загальний рівень фрагментації. Це дозволяє виключити вплив розміру стеганограми, кількості та розміру покриваючих файлів на досліджуваній результат. Зведена гістограма впливу обраного способу приховування інформації на середній рівень фрагментації зображена на рисунку 4.10 [65].

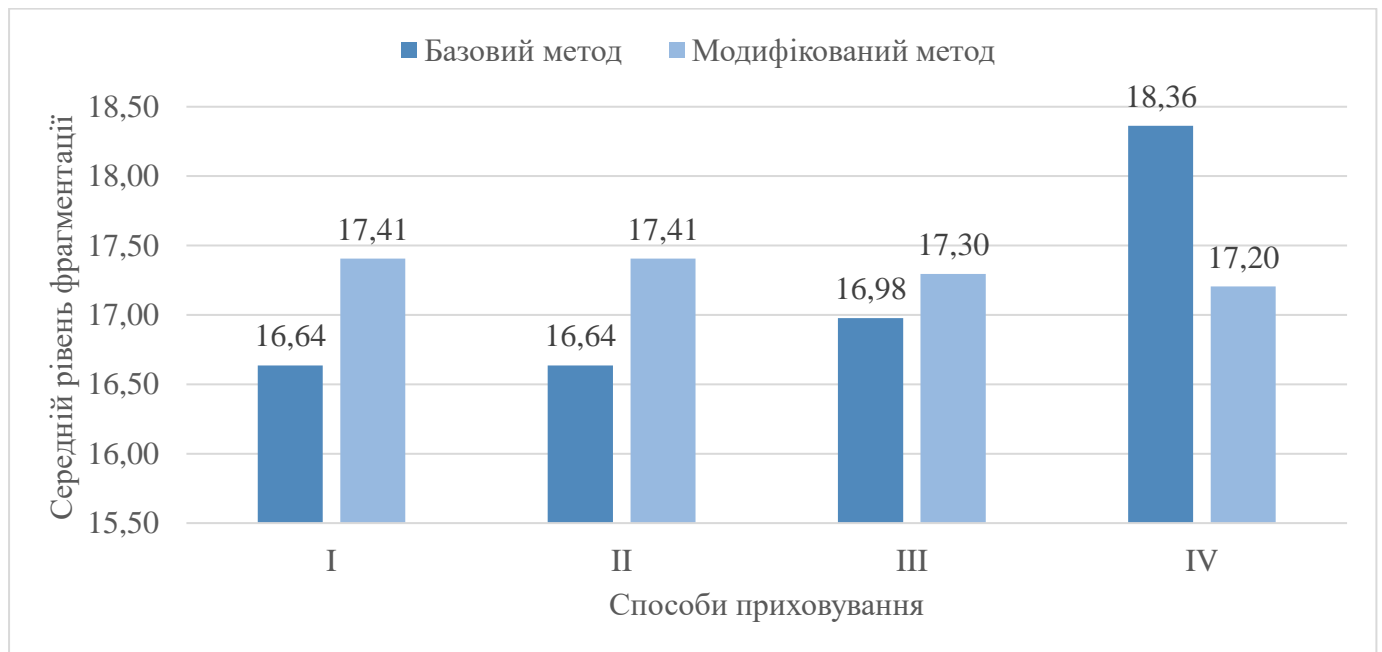


Рис. 4.10 Залежність способу приховування інформації на середній рівень фрагментації покриваючих файлів.

Як видно із рисунку 4.10 рівень фрагментації при використанні одного способу майже не змінюється якщо виконати приховування іншим способом (відмінність у межах 0.5-1%). Також необхідно зазначити, що при використанні способів базового методу приховування інформації рівень фрагментації буде збільшуватись у залежності від обраного методу (ПЗОП та ПчЗОП-I – рівень фрагментації менше, ПчЗОП-II та ПчЗОП-III – рівень фрагментації трохи більший). На відмінну від способів модифікованого методу приховування інформації, де рівень фрагментації трохи зменшується (у межах 0.1%). А також можна помітити, що рівень фрагментації при використанні модифікованого методу трохи більший ніж при використанні базового метода.

Висновки до розділу 4

У даному розділі були описані критерії програмної реалізації методів приховування інформації у структуру файлової системи шляхом перемішування кластерів покриваючих файлів. Аргументовано переваги програмної симуляції перед розробкою програми, що виконує методи над фізичним носієм інформації. Як результат розроблено та описано програмну симуляцію, надано алгоритмічний опис кожного із способів та методів приховування інформації, також описано інші важливі допоміжні функції. Описано алгоритм дії користувача при використанні програмної симуляції.

Використовуючи розроблену програмну реалізацію було отримано емпіричні оцінки обчислювальної складності по кожному із способів приховування інформації. При цьому оцінка проводилася у залежності від: кількості покриваючих файлів, розміру кожного покриваючого файлу, розміру повідомлення що необхідно приховати та від формату повідомлення (псевдовипадкове повідомлення чи читаєме). Із отриманих результатів зроблено висновок, що в цілому емпіричні дані відповідають теоретичним очікуванням. Навіть способи базового методу ПчЗОП-II та ПчЗОП-III наочно продемонстрували свою перевагу перед іншими способами (кількість кластерів що необхідно перезаписати на 30-40% менша за інші способи). Але для способів модифікованого методу ПчЗОП-IIм та ПчЗОП-IIIм емпірична оцінка кількості перезаписаних кластерів не відповідає теоретично розрахованій, що нівелює перевагу використання даних способів перед іншими. Також висунуто гіпотезу щодо можливого вдосконалення програмної реалізації, що дозволить способам ПчЗОП-IIм та ПчЗОП-IIIм мати таку розрахункову складність, при перезаписі кластерів, як теоретично обчислену.

Було проаналізовано вплив кожного із способів приховування інформації на рівень фрагментації покриваючих файлів. Як висновок можна стверджувати, що такий вплив мінімальний. Для способів базових методів рівень фрагментації зростає з

використанням наступних способів у межах 0.5-1%. Для способів модифікованого методу приховування інформації такої залежності не спостерігається. Загалом при використанні модифікованого методу, рівень фрагментації трохи більший а ніж при використанні базового методу приховування інформації (більший в середньому на 5%).

Подальше дослідження методів приховування інформації потребує аналізу можливих шляхів використання, опис та аналіз систем у яких дані методи могли б бути використані, порівнянні із подібними системами, що використовують інші вже розповсюджені інструменти.

У рамках даного розділу виконана наступна часткова задача дослідження, а саме розробка програмної реалізації запропонованих методів приховування інформації та проведення експериментальних досліджень. А також отримано практичні результати та експериментально встановлено та доведено, що:

- запропонований метод дозволяє збільшити майже у два рази пропускну спроможність утворюваних стеганоканалів;
- удосконалений метод дозволяє зменшити обчислювальні витрати оперативної пам'яті з лінійної залежності (від кількості кластерів покрівельних файлів) до константного значення;
- удосконалений метод дозволяє зменшити кількість циклів перезапису кластерів файлової системи на фізичному носії даних від 10% до 10 разів (в залежності від вихідних співвідношень розмірів інформаційних повідомлень, розмірів кластерів та кількості покрівельних файлів);

Результати досліджень даного розділу наведено в публікації здобувача: [65].

РОЗДІЛ 5

НАПРЯМКИ ВИКОРИСТАННЯ МЕТОДІВ ПРИХОВУВАННЯ ІНФОРМАЦІЇ

Необхідно зазначити, що методи приховування інформації у структуру сімейства кластерних файлових системи FAT шляхом перемішування кластерів покриваючих файлів, що розроблені та проаналізовані у межах даної дисертаційної роботи є інструментами що дозволяє приховати певну інформацію. Метою такого приховування може бути: приховане збереження інформації, прихована передача інформації та збереження ключової інформації для верифікації фізичного носія чи інших даних [81, 82]. Далі розглянуто та проаналізовано наступні шляхи використання методів приховування інформації:

1. Система, що дозволяє приховано зберігати інформацію використовуючи описані вище методи.
2. Система, що дозволяє приховано передавати інформацію.
3. Система, що дозволяє виконувати верифікацію фізичного носія чи інших файлів.

По кожному способу використання надано опис такої системи, надано оцінку ефективності за відповідними критеріями.

5.1 Система прихованого збереження даних

Основною метою такої системи є необхідність зберегти певну інформацію M на фізичному носії у такий спосіб, щоб мінімізувати можливість несанкційного доступу до інформації (мінімізувати вирогідність детектування прихованого повідомлення). Така система умовно може складатися із таких компонентів/акторів [83, 84]:

- структура файлової системи, із такими параметрами $FSize$ – розмір файлової системи у кластерах, $\varphi(FS)$ – середній рівень фрагментації файлової

системи, $t(FS)$ – швидкодія фізичного носія файлової системи (тобто час перезапису одного кластеру даних);

- власник інформації, умовно має наступні параметри: M – інформацію, що необхідно зберігти, t – допустимий час на приховування інформації;
- зловмисник, особа або система від якої необхідно приховати повідомлення, умовно має наступні параметри: D – детекційні здібності особи, ε – доступ до файлової системи, $t(\varepsilon)$ – час доступу до файлової системи зловмисником.

Маючи описаних вище акторів із зазначеними параметрами можна виділити декілька систем, що відрізняються деякими параметрами:

1. Система що складається із персонального комп'ютера власника інформації, доступ до якого має лише він, як і доступ до фізичного носія інформації. У такій системі середній рівень фрагментації файлової системи може бути різним і ніяк не впливає на ефективність методів, також швидкодія фізичного носія впливає лише на зручність користування. Розмір приховуваного повідомлення умовно обмежено лише розміром файлової системи $FSSize$. Допустимий час на приховування інформації також умовно необмежений. Зловмисник у такій системі не має доступу до файлової системи і тому у такій системі не розглядається. Схематичний опис системи зображено на рисунку 5.1 [18, 85, 86].

Прикладом такої системи може виступати програмна реалізація, що інстальована на персональному комп'ютері. Де власник системи обирає фізичний носій, далі вводить імена покриваючих файлів після чого може обрати повідомлення для приховування. Така система може нормально функціонувати при будь яких параметрах файлової системи, так як власник інформації має повний доступ до файлової системи.

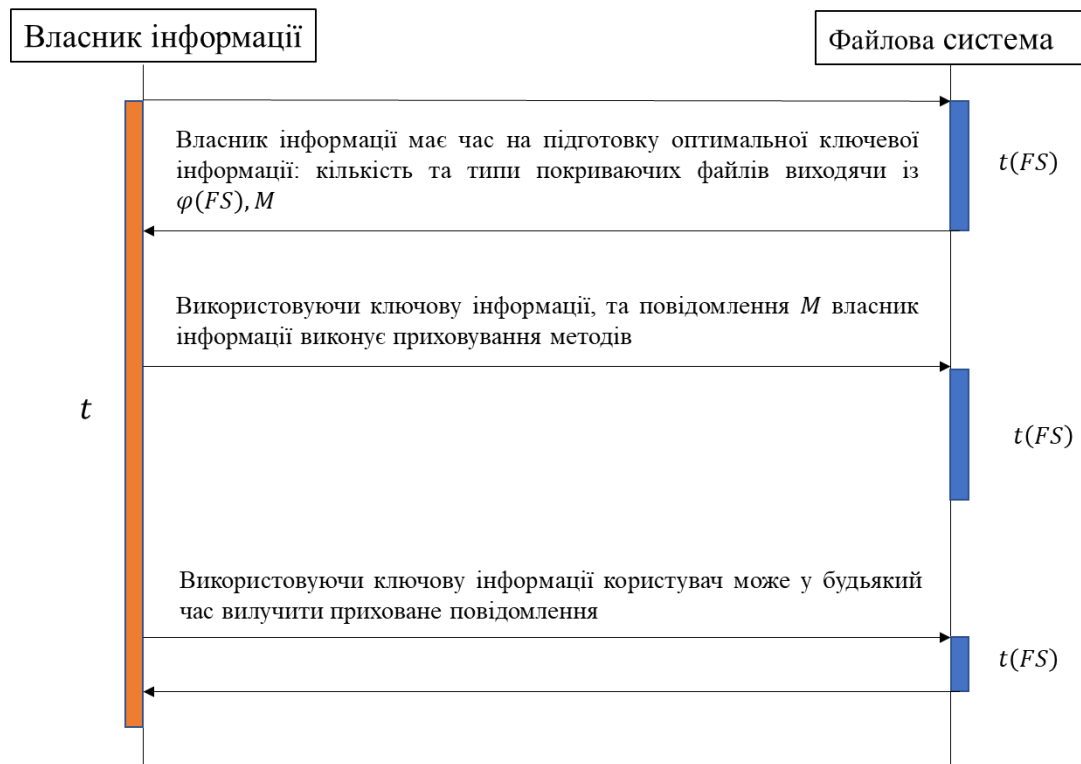


Рис. 5.1 Схематичний опис системи збереження інформації із повним доступом власника інформації

2. Система до якої власник інформації має рівний доступ із зловмисником. Як приклад це може бути загально доступний комп'ютер у публічному місці чи офісі. У такому випадку рівень фрагментації файлової системи матиме значний вплив на можливий розмір приховуваного повідомлення. Та власник інформації матиме обмежений час на підготовку ключової інформації, у той час як зловмисник так само матиме доступ до файлової системи. У такому випадку необхідно приховувати повідомлення у залежності від наявного часу на приховування інформації, та із таким рівнем фрагментації покриваючих файлів щоб зловмисник не зміг детектувати повідомлення.

3. Система до якої власник інформації має обмежений доступ, у той час як зловмисник має необмежений доступ. У такій системі, у власника інформації може не

бути достатнього часу на дослідження файлової системи щоб згенерувати відповідну ключову інформацію. Також необхідно бути обережним із вихідним рівнем фрагментації, так як збільшення рівня фрагментації покриваючих файлів може детектувати приховане повідомлення [85, 86].

Порівняльний аналіз даних систем зазначено у таблиці 5.1

Таблиця 5.1

Порівняльний аналіз систем збереження інформації

Система збереження інформації	Час на підготовку ключової інформації, аналіз ФС	Час на приховування/вилучення повідомлення	Допустимий рівень фрагментації
Власник інформації має повний доступ	Великий	Великий	Будь-який
Власник та злоумисник мають рівний доступ	Середній	Середній	У межах середнього
Власник має обмежений доступ	Малий	Малий	У межах середнього

5.2 Система прихованої передачі даних

Особливістю такої системи є те що для передачі інформації використовуються переносимі фізичні пристрої збереження інформації такі як: флеш-накопичувачі, SD-карти, мобільні пристрої чи переносимі жорсткі диски. Для таких систем необхідно окремо виділити четвертий компонент – отримувач інформації. Отримувач інформації має наступні параметри: доступ до фізичного носія, час допустимий для вилучення повідомлення [87, 88].

Прикладами такої системи може бути ситуація при якій необхідно винести конфіденційну інформацію за межі контрольованої зони (КЗ). Таким чином власник інформації – це особа яка має доступ до КЗ та до режимної інформації. Файлова система – це файлова система на переносимому фізичному носії інформації у структуру якої буде приховано повідомлення. Злоумисник – особа чи система яка

перевіряє фізичні носії на наявність забороненої інформації. Отримувач – особа яка не має доступу до режимної інформації. Отже на етапі збереження інформації Власнику необхідно мати спеціалізоване портативне програмне забезпечення та переносимий пристрій збереження інформації. Використовуючи дані засоби власник приховує інформацію у структуру файлової системи використовуючи один із методів. Власник може заздалегідь підготувати файлову систему та ключову інформацію (набір покриваючих файлів), що забезпечить більш допустимий розмір приховуваного повідомлення та збільшить рівень захищеності від детектування. На етапі приховування інформації власник може мати обмежений час на приховування інформації. На етапі проходження кордону КЗ зловмисник має доступ до фізичного носія, саме на цьому етапі буде виконуватись детектування прихованої інформації. Після успішного проходження перевірки, отримувач інформації має повний та необмежений за часом доступ до прихованої інформації. Таку систему можна розділити по наступним показникам [89]:

- Власник має повний доступ до інформації, а зловмисник немає ніякого доступу: така система являє собою КЗ без належної перевірки переносимих фізичних носіїв інформації.
- Власник має необмежений доступ, а Зловмисник – обмежений доступ: це така система при якій реалізовано поверхнева перевірка фізичного носія.
- Власник та зловмисник мають обмежений доступ: система при якій реалізована поверхнева перевірка фізичного носія та власник має обмежений час на приховування інформації.
- Власник не має доступу: система при якій виконується вилучення фізичних носіїв інформації, таку систему не має сенсу розглядати так як виконання методів приховування інформації у структуру файлової системи неможливе.

За описаними вище варіантами системи прихованої передачі інформації зроблено аналіз результати якого зазначені у таблиці 5.2

Таблиця 5.2

Порівняльний аналіз систем прихованої передачі інформації

Система прихованої передачі інформації	Час на підготовку ключової інформації, аналіз ФС	Час на приховування повідомлення	Час на вилучення повідомлення	Допустимий рівень фрагментації
Власник інформації має повний доступ	Великий	Великий	Великий	Будь-який
Власник має необмежений доступ, а Зловмисник – обмежений доступ	Великий	Великий	Великий	У межах середнього
Власник та Зловмисник мають обмежений доступ	Великий	Середній	Великий	У межах середнього
Власник не має доступу	Великий	—	Великий	—

Оцінюючи результати таблиці 5.2 можна стверджувати, що при прихованій передачі інформації час на підготовку ключової інформації необмежений, це означає що Власник інформації може підготувати файлову систему до приховуваного повідомлення (штучно створити надлишковий рівень фрагментації, задати необхідний розмір покриваючих файлів і так далі). Також при прихованій передачі, час на вилучення повідомлення так само значний, так як якщо Отримувач отримав фізичний носій то він вже має необмежений доступ до інформації. Єдине вразливе місце це аналіз зловмисником допустимого рівня фрагментації покриваючих файлів, але при описаній вище системі такий аналіз виконуватиметься поверхнево так як має обмежений час[88, 90].

Отже з точки зору успішної передачі інформації рекомендується максимально підготувати файлову систему та ключову інформацію перед приховуванням повідомлення. А з точки зору захищеності режимної інформації від прихованої

передачі, рекомендується обмежити можливість використовувати персональні переносимі фізичні пристрої збереження інформації.

5.3 Система верифікації фізичних носіїв та файлів

Особливість такої системи полягає у тому що, Власник інформації завжди має повний доступ до файлової системи під час приховування інформації у структуру файлової системи. А вже потім, така файлова система потрапляє до потенційного зломисника який також матиме повний доступ до файлової системи. Прикладом системи верифікації фізичного носія може бути системи при якій користувачу надається переносний фізичний пристрій збереження інформації. У даний пристрій адміністратором приховано ключ завдяки якому такий пристрій може бути використано у закритій системі (як приклад флеш накопичувачі можуть не відкриватися якщо вони не мають валідного прихованого ключа). У якості покриваючих файлів можуть бути згенеровані файли, що містять персональні дані користувача фізичного носія, Та у якості прихованого повідомлення може бути геш значення від цих даних. Якщо зломисник спробує змінити відкриту інформацію то значення геша не співпадатиме і доступ до системи не буде надано. У разі якщо зломисник спробує видалити відкриту інформацію (видалити покриваючі файли, чи зробити дефрагментацію) то доступ до системи так само не буде надано. Таким чином у даній системі час на підготовку ключової інформації буде необмеженим, час на приховування/вилучення повідомлення також буде необмежено, та допустимий рівень фрагментації буде значним. Так як зломисник і так буде знати що певне повідомлення приховано і у разі видалення чи заміні повідомлення доступ до системи буде заблоковано. З погляду ефективності методів приховування інформації така система є найбільш сприятливою до використання. Але у іншу чергу така система потребує додаткової інфраструктури, а саме: спосіб перевірки геш-значення, блокування фізичного носія інформації у разі невалідних прихованих даних. У межах

даної дисертаційної роботи назвемо таку систему – Система верифікації фізичного носія (СВФН) [91, 92].

Подібною до цього також може бути система дистрибуції фізичних носії інформації. У такому разі виробник зберігає на фізичному носії відкриту інформацію (сертифікат якості, лист подяку, інструкцію і тому подібне) та використовуючи відкриту інформацію у якості покриваючих файлів приховує ідентифікаційний номер пристрою. Використовуючи спеціальне програмне забезпечення надане розробником, користувач може перевірити, що придбаний пристрій збереження інформації сертифікований. У такій системі так само власник інформації має необмежений доступ до файлової системи, допустимий рівень фрагментації може бути будь-яким. Недоліком такої системи є необхідність у додатковій інфраструктурі: розробнику необхідно надати доступ клієнту до можливості перевіряти ідентифікаційний код пристрою через мережу інтернет, також розробник фізичного носія має розповсюджувати програмне забезпечення для вилучення прихованого повідомлення а клієнту необхідно буде інстальовати таке програмне забезпечення, що потребує додаткового часу та ресурсів. У межах даної дисертаційної роботи назвемо таку систему – Система дистрибуції фізичних носії (СДФН) [93, 94].

Наступний спосіб, використання методів приховування інформації у структуру файлової системи шляхом перемішування кластерів покриваючих файлів, полягає у збереженні ключової інформації стосовно зберігаємих на носії файлів, з ціллю подальшої перевірки таких файлів на цілісність. У такої системи є наступні особливості:

1. Власник інформації має необмежений доступ до фізичного носія інформації, та має необмежену кількість часу, щоб підготувати ключову інформацію (покриваючи файли) так щоб знизити ризики детектування прихованої інформації.
2. Файлова система потребує певного мінімального розміру файлів цілісність яких необхідно захищати у такий спосіб. Тобто це означає що у разі

забезпечення цілісності одного файлу що має недостатній розмір у кластерах, може виникнути ситуація коли не вдасться приховати певну верифікаційну інформацію.

3. Зловмисник матиме необмежений доступ до файлової системи після отримання фізичного носія інформації.

Прикладом такої системи може слугувати дистрибуція ліцензійного програмного забезпечення на переносних фізичних носіях. Власник інформації, він же і власник фізичного носія завантажує певне програмне забезпечення на носій. Після цього власник інформації розраховує геш-значення від інформації що збережена на носії. Отримане геш-значення буде повідомленням, чи частиною повідомлення, що буде приховано у структуру файлової системи шляхом перемішування кластерів покриваючих файлів, де покриваючими файлами слугують файли від яких розраховано геш-значення. Користувач, отримавши даний фізичний носій, починає процес інсталяції чи запуску програмного забезпечення. При цьому дане програмне забезпечення повинно мати компонент що дозволить перевірити порядок кластерів покриваючих, тобто вилучити приховане повідомлення. Після чого зіставити вилучене повідомлення із геш-значеннями – таким чином перевірити цілісність файлів. У разі якщо зловмисник спробує скопіювати файли на інший накопичувач то порядок кластерів покриваючих файлів буде перевпорядкованим та геш-значення не співпадатиме із невірною вилученим повідомленням. У разі якщо зловмисник спробує скопіювати повністю файлову систему із збереження порядку кластерів, то у такому випадку необхідно приховувати повідомлення яке не тільки залежить від геш-значення але й від параметрів фізичного носія. У межах даної дисертаційної роботи назовемо таку систему – Система дистрибуції ліцензійного програмного забезпечення (СДЛПЗ) [94].

Ще одним прикладом системи із використанням методів приховування інформації у структуру файлової системи шляхом перемішування кластерів покриваючих файлів із метою забезпечення цілісності файлів може бути система електронного документообігу. У такому випадку власник інформації записує

необхідні файли на фізичний носій. Після чого розраховує геш значення від цих файлів та підписує їх своїм ключем. Тобто виконує операцію подібну до електронного цифрового підпису. Даний підпис і буде повідомленням, що необхідно приховати у структуру файлової системи. Таким чином з точки зору побутового користування підписані файли нічим не відрізняються від не підписаних, так як підпис приховано у черговості кластерів цих файлів. І якщо злоумисник спробує відредагувати дані файли, то перевірка підпису буде не виконана. Коли підписані файли необхідно перевірити, то отримувач, знаючи публічний ключ підписанта та набір покриваючих файлів, зможе вилучити та перевірити підпис. Недоліком такої системи є те, що Власник та Отримувач інформації повинні мати спеціальне програмне забезпечення для приховування та вилучення повідомлення. У межах даної дисертаційної роботи назовемо таку систему – система електронного документообігу (СЕД) [69, 95].

5.4 Опис та дослідження компонентів програмного комплексу StarForce

StarForce – це торгова марка, під якою випускаються різні програмні продукти, розроблені компанією Protection Technology [<https://www.star-force.ru/>]. Основний напрям компанії: інформаційна безпека, захист від несанкційного доступу, аналіз та модифікація (декомпіляція) програмних продуктів. У рамках даної дисертаційної роботи нас цікавлять саме такі програмні продукти: StarForce Disc, StarForce Audio/Video. Дані продукти є подібними системами до систем наведених вище. Та можуть бути порівняні за ефективністю [96].

StarForce Disc – історично перший продукт компанії. Захист базується на зв'язці копії програмного забезпечення до оптичного носія: диску CD чи DVD. Перевагою такої технології (перед іншими розповсюдженими) є те, що відсутня необхідність активації продукту через мережу інтернет. Недоліком може бути те, що при роботі програмного забезпечення диск постійно повинен бути у оптичному приводі користувача. Для перевірки диска використовується певна кількість секторів на

певних кільця спіралі диска із послідовним порівняння отриманих даних із інформацією, закодованої у ключі. При копіюванні оригінального диска на CD-R/DVD-R ці дані гарантовано не співпадають, та процедура перевірки завершиться невдало. Також StarForce використовує захист від емуляторів, для чого встановлює у операційну систему драйвери, щоб відрізнити дійсні оптичні приводи від віртуальних.

Теоретично захист виконуються наступним чином. Алгоритм захисту програмного забезпечення виявляє оригінальний диск за швидкістю зчитування різних секторів диска. На оптичному диску, як відомо, є спіральна доріжка, по якій орієнтується луч лазера при зчитуванні та запису. При виробництві дисків StarForce ця сама спіральна дорожка має не стандартну конфігурацію, таким чином змінюється щільність даних в різних секторах і як наслідок, час зчитування таких секторів також буде різний. На усіх CD-R та CD-RW дисках ця направляюча дорожка наноситься при виробництві, а не при запису диска. Як наслідок, усі подібні носії інформації відбраковуються StarForce Disc-ом. Записати диск захищений технологією StarForce, який би працював так само як і оригінал, самому неможливо. [97]

У даній роботі наведені переваги систем із використанням методів приховування інформації у структуру файлової системи, шляхом перемішування кластерів покриваючих файлів, у порівнянні із програмною реалізацією StarForce Disc.

1. StarForce Disc використовує лише оптичні диски для захисту програмного забезпечення, у той час як методи приховування інформації (описані у даній дисертаційній роботі) абстрагуються від типу фізичного носія та працюють із типом файлової системи. Тобто методи приховування інформації можуть працювати як із оптичними дисками так із флеш накопичувачами, жорсткими дисками, віртуальними системами.

2. StarForce Disc використовує певну особливість оптичних дисків при створенні їх на виробництві, у той час як методи приховування інформації можуть

працювати із будь яким стандартним носієм інформації (тобто не потребує додаткових виробничих потужностей).

3. Для роботи програмного забезпечення захищеного StarForce Disc необхідно щоб оригінальний оптичний диск увесь час знаходився у оптичному приводі. Для систем, що базуються на методах приховування інформації ситуація подібна. Так як необхідно буде перевіряти порядок кластерів покриваючих кластерів на конкретному носії інформації.

4. Для надійного захисту, StarForce Disc потребує встановлення додаткових драйверів у операційну систему (що є стороннім програмним забезпеченням). У той час як методи приховування повідомлення не потребують цього [97].

Таким чином StarForce Disc є надійним програмним забезпеченням та довела свою ефективність (StarForce Disc утримує неофіційний світовий рекорд за тривалістю злому комп'ютерної гри класу AAA – 422 доби. А саме стільки протрималася “Splinter Cell: Chaos Theory” від Ubisoft). Але інструменти якими виконується захист є застарілими. А отже використання методів приховування інформації здатне значно покращити StarForce Disc.

Програмне забезпечення StarForce Audio/Video дозволяє захистити аудіо та відео файли (потенційно усі види файлів). Нас цікавить саме варіація із прив'язкою файлів до носія інформації.

Прив'язка – це метод захисту від копіювання та несанкційного розповсюдження, що базується на зв'язку із фізичним об'єктом – носієм ліцензії. Такий об'єкт зазвичай володіє унікальними параметрами, які важко скопіювати або відтворити, наприклад геометрія доріжок *CD* и *DVD* (прив'язка до диску), серійні номери компонентів комп'ютера чи флеш-накопичувачів. Прив'язка використовується для надання захисту *CD/DVD* дисків та USB-накопичувачів. Прив'язки є ефективним методом захисту інформації, як зазначає сам розробник.

Робота захищеного файлу можлива лише через спеціальне програмне забезпечення чи драйвер StarForce, які з одного боку забезпечує зв'язок файлу із

об'єктом прив'язки, а з іншого надійно захищає файл від аналізу та модифікації, для запобігання можливого злому та виділення захищеного файлу від об'єкта прив'язки [97, 98, 99].

Таким чином порівнюючи StarForce Audio/Video та системи що базуються на методах приховування інформації (описаних у даній дисертаційній роботі) можна зазначити що, прив'язка у StarForce Audio/Video може не мати прямої залежності між захищеним файлом та об'єктом прив'язки. Тобто файли у вас можуть бути збережені локально на комп'ютері у той час як зв'язаний ключ доступу до файлів може зберігатися окремо на оптичному чи іншому носії. Методи приховування інформації у свою чергу мають пряму зв'язку між файлами та фізичним носієм. Також StarForce Audio/Video потребує додаткового (стороннього) програмного забезпечення, так само як і методи приховування інформації. Загалом можна стверджувати що методи приховування інформації можуть бути додатковим інструментом у системі StarForce Audio/Video, таким чином розширюючи можливості компонента зв'язування [98, 99 100].

Висновки до розділу 5

У даному розділі були описані приклади системи що використовують методи приховування повідомлення у структуру файлової системи шляхом перемішування кластерів покриваючих файлів. Надано оцінку таких систем, їх переваги та недоліки у залежності від різних варіацій системи. Також описано програми сімейства StarForce, а саме StarForce Disc та StarForce Audio/Video. Було надано порівняння між описаними системами та системами StarForce. Узагальнюючи, методи приховування інформації варто вважати інструментами такими, що здатні розширити та вдосконалити інструментарій програмного пакету StarForce. Особливо StarForce Disc так як дане програмне забезпечення здатне працювати лише із оптичними дисками, а із використанням методів приховування, функціонал можна розширити до усіх

фізичних пристроїв, що можуть використовувати кластерну файлову систему сімейства FAT.

У рамках даного розділу отримано практичні результати, а саме розроблено практичні рекомендації щодо впровадження розроблених методів.

Результати досліджень даного розділу наведено в публікаціях здобувача: [34, 44, 49, 65].

ВИСНОВКИ

В дисертаційній роботі розв'язана актуальна задача розробка методу підвищення пропускної здатності у кластерних стеганосистемах та надано модель оцінки параметрів таких систем. Також розроблено удосконалені методи приховування інформації, що дозволяє зменшити час приховування. В цілому можна зробити наступні висновки.

1. Виконана основна мета дисертаційної роботи, а саме підвищення пропускної здатності кластерних стеганосистем при забезпеченні необхідної стійкості до несанкціонованого детектування прихованої інформації, за рахунок введення додаткової залежності між кластерами одного покриваючого файлу. Таке підвищення пропускної здатності дозволяє приховати у двічі більше інформації при тих самих вихідних параметрах файлової системи. А також без значного підвищення рівня фрагментації покриваючих файлів, що дозволяє стверджувати, що необхідний рівень стійкості до несанкціонованого детектування прихованої інформації забезпечено.

2. Проведено дослідження сучасних методів зберігання інформації, властивостей фізичних носіїв та типів файлових систем. Що дало можливість зробити висновок про переваги та недоліки таких технологій зберігання інформації як HDD та SSD. Також надано порівняльний аналіз таких файлових систем FAT32, NTFS, exFAT.

3. Проведено аналіз файлової системи FAT на можливі стеганографічні методи. Були виявлені методи приховування інформації, що використовують структурну особливість файлової системи.

4. На основі методу приховування інформації у структуру файлової системи FAT шляхом перемішування кластерів було розроблено метод підвищення пропускної здатності.

5. Було удосконалено математичну модель оцінки основних параметрів стеганосистем. Це досягається за рахунок комплексної оцінки таких параметрів:

пропускна здатність, стійкість до детектування, обчислювальна складність. У ході дослідження оцінки пропускної здатності було надано формули що дозволяють точно оцінити можливий максимальний розмір стеганограми у залежності від кількості покриваючих файлів, розміру одного кластеру. Достовірність даних формул було підтверджено експериментально. У ході дослідження оцінки стійкості до детектування були використані статистичні методи, а саме оцінювався розподіл фрагментованих файлів у файловій системі. Це дало змогу надати рекомендації щодо параметрів стеганосистеми при яких стійкість до детектування буде найоптимальнішою.

6. Було удосконалено методи приховування інформації у кластерні стеганосистеми. Удосконалення полягає у оптимальному використанні оперативної пам'яті, що дає змогу використовувати досліджувані стеганографічні методи у малоресурсних системах. Друге удосконалення полягає у генерації таблиць перестановки таким чином, щоб зменшити кількість переміщень кластерів. Це дозволяє значно зменшити час на приховування інформації.

7. Було розроблено програмну реалізацію що дозволяє наочно продемонструвати хід виконання методів приховування інформації, а також дозволяє провести експериментальні дослідження. Усього було розроблено та використано такі програмні реалізації:

- SteganoFAT – програмна реалізація розроблена із використання мови програмування C++, використовувалася для досліджень зазначених у розділі 3.3 (<https://drive.google.com/drive/folders/10kLAlZ-v7of7SeSIBjl3KMumFAVjjBtw?usp=sharing>).

- Програма для парсингу даних – написана мовою програмування JavaScript, використовувалася для аналізу лог-файлів наданих програмою Auslogics Disk Defrag, використовувалася для досліджень зазначених у розділі 3.2 (<https://github.com/ShekhaninKyryl/ausdiskdefrag-parser>).

– SteganoSimulation – програмна симуляція написана мовою програмування JavaScript, використовувалася для досліджень зазначених у розділі 3.3, 3.5, 4 (<https://github.com/ShekhaninKyryl/SteganoSimulation>).

Також були отримані наступні нові *науково-обґрунтовані результати*:

1. Вперше отримано метод підвищення пропускної здатності кластерних стеганосистем на основі урахування додаткової залежності місць розміщення кластерів у межах одного покриваючого файлу системи;
2. Удосконалено математичну модель оцінки основних параметрів кластерних стеганосистем за рахунок додаткового урахування елементів конфігурації стеганосистеми, що дозволяє більш повно оцінити пропускну здатність системи;
3. Удосконалено метод приховування інформації у структуру кластерних стеганосистем за рахунок генерації відповідного набору перестановок кластерів, що дозволяє зменшити час приховування інформації;

Таким чином результати дослідження надають змогу підвищити пропускну здатність кластерних стеганоканалів майже у двічі, при збереженні необхідного рівня стійкості до детектування прихованого повідомлення. Також використовуючи удосконалення за оперативною пам'яттю та за кількістю перезаписів можна значно зменшити час на приховування повідомлення та в цілому зменшити вимоги до стеганосистеми. Було встановлено необхідні вихідні параметри стеганосистеми для надійного приховування інформації та набула подальшого розвитку модель оцінювання кластерних стеганосистем, за такими параметрами: пропускна здатність, розрахункова складність, стійкість до детектування.

Використання запропонованих методів приховування інформації надає змогу для створення систем захисту інформації із стеганогіфнічними елементами, як для прихованого збереження чи передачі інформації так і для верифікації фізичного носія чи інформації на ньому. Такі системи наразі широко розповсюдженні, та стеганографічні методи здатні розширити функціонал вже існуючих систем захисту інформації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Горбенко І.Д. Захист інформації в ІТС. Криптографічний захист інформації / І.Д. Горбенко. – Харків : ХНУРЕ, 2004. – 376 с.
2. Конахович Г.Ф., Пузиренко Компьютерная стеганография. Теория и практика / Г.Ф. Конахович, О.Ю. Пузиренко. МК-Пресс: 2006. – 288 с.
3. Грибунин В.Г. Цифровая стеганография / В.Г. Грибунин, И.Н. Оков, И.В. Туринцев. – Москва: Солон-Пресс, 2002. – 272 с.
4. Хорошко В.А. Введение в компьютерную стеганографию / В.А. Хорошко, М.Е. Шелест. – К.: 2002. – 140 с.
5. Kuznetsov A. Method of 3D-steganography. *CS&CS E-journal*. 2018. № 4. P. 4 – 12.
6. Kuznetsov A. Information Hiding Using 3D-Printing Technology / A. Kuznetsov // 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), 2019. – P. 701 – 706.
7. Kuznetsov A. A. 3D steganography information hiding. *Telecommunications and Radio Engineering*. 2019. Vol. 78, Iss. 12.
8. Klima R.E., Klima R., Sigmon N.P., Sigmon N., Klima R., Sigmon N.P., Sigmon N. Cryptology: Classical and Modern. Chapman and Hall/CRC. [Електронний ресурс] – 2018. – Режим доступу: <https://doi.org/10.1201/9781315170664>. – Загл. з екрану.
9. Li N., Hu J., Sun R., Wang S., Luo Z. A High-Capacity 3D Steganography Algorithm With Adjustable Distortion. *IEEE Access*. 2017. Vol. 5, P. 24457–24466. [Електронний ресурс] – 2017. – Режим доступу: <https://doi.org/10.1109/ACCESS.2017.2767072>. – Загл. з екрану.

10. Thiyagarajan P., Natarajan V., Aghila G., Prasanna Venkatesan V., Anitha R. Pattern based 3D image Steganography. 3D Research. 2014. Vol. 4, Iss. 1. [Электронный ресурс] – 2017. – Режим доступа: [https://doi.org/10.1007/3DRes.01\(2013\)1](https://doi.org/10.1007/3DRes.01(2013)1). – Загл. з екрану.
11. Kuznetsov, A.A., Kovalenko, O.Yu. Steganographic information protection using 3D printing. *Information security of the state, society and personality, Tez Vseukr. Science.-Pract. Conf. Kirovograd. KNTU*. 2015. P. 91-92.
12. Кузнецов А.А., Коваленко О.Ю. Стеганографическая защита информации с использованием 3D-печати. // Інформаційна безпека держави, суспільства та особистості : зб. тез доповідей Всеукр. наук.-практ. конф, 16 квітня 2015 року, Кіровоград / КНТУ, 2015. – С. 91 – 92.
13. Пескова О.Ю., Халабурда Г.Ю. Применение сетевой стеганографии для защиты данных, передаваемых по открытым каналам интернет // Информ. системы для науч.х исследований (IMS-2012). Тр. XV Всерос. объединенной конф. "Интернет и современное общество" (IMS-2012). Санкт-Петербургский нац. иссл. ун-т информ. технологий, механики и оптики, 2012. С. 348-354. [Электронный ресурс] – 2018. – Режим доступа: <http://ojs.ifmo.ru/index.php/IMS/article/download/132/132>. – Загл. з екрану.
14. Lubacz J., Mazurczyk W., Szczypiorski K. Principles and Overview of Network Steganography. IEEE Communications Magazine. 2012. Vol. 52. [Электронный ресурс] – 2018. – Режим доступа: <https://doi.org/10.1109/MCOM.2014.6815916>. – Загл. з екрану.
15. Wang, M., Gu, W., & Ma, C. A Multimode Network Steganography for Covert Wireless Communication Based on BitTorrent. Security and Communication Networks, 2020. [Электронный ресурс] – 2020. – Режим доступа: <https://www.hindawi.com/journals/scn/2020/8848315>. – Загл. з екрану.
16. Seo J.O., Manoharan S., Mahanti A. Network steganography and steganalysis – a concise review // 2016 2nd .International Conference on Applied and Theoretical

Computing and Communication Technology (iCATccT). 2016. P. 368–371. [Електронний ресурс] – 2018. – Режим доступу: <https://doi.org/10.1109/ICATCCCT.2016.7912025>. – Загл. з екрану.

17. Khan, H., Javed, M., Khayam, S. A., Mirza, F. Designing a cluster-based covert channel to evade disk investigation and forensics. *Computers & Security*. 2011. Vol. 30, Iss. 41. P. 35-49.

18. Venčkauskas A. Covert Channel for Cluster-based File Systems Using Multiple Cover Files. *Information Technology and Control*. 2013. Vol. 42, Iss. 3. P. 260–267.

19. Shekhanin K.Yu., Kolhatin A.O., Demenko E.E., Kuznetsov A. A.. On Hiding Data Into the Structure of the FAT Family File System. *Telecommunications and Radio Engineering*. 2019. Vol. 78, Iss. 11, P. 973–985.

20. Шеханін К., Колгатін А., Деменко Є., Кузнецов О. Data hiding in the FAT family file system structure. *Радіоелектроніка*. 2018. №. 193. С. 169–178.

21. Winter R. SSD vs HDD – Data recovery and destruction. *Network Security*. 2013. Vol. 3. P. 12 – 14. [Електронний ресурс] – 2018. – Режим доступу: <https://www.scopus.com/record/display.uri?eid=2-s2.0-84875203304&origin=resultslist>. – Загл. з екрану.

22. Rizvi S.S., Chung T.-S. Flash SSD vs HDD: High performance oriented modern embedded and multimedia storage systems. *2010 2nd International Conference on Computer Engineering and Technology*. 2010. Vol. 7. [Електронний ресурс] – 2018. – Режим доступу: <https://www.scopus.com/record/display.uri?eid=2-s2.0-77958063178&origin=resultslist>. – Загл. з екрану.

23. Microsoft Extensible Firmware Initiative FAT32 File System Specification Version 1.03 : Затв. групою технічної підтримки Microsoft 12.06.2006. Microsoft Corporation. 2006. С. 34. [Електронний ресурс] – 2018. – Режим доступу: <https://download.microsoft.com/download/1/6/1/161ba512-40e2-4cc9-843a-923143f3456c/fatgen103.doc>. – Загл. з екрану.

24. Руссон Р. NTFS Documentation. Microsoft Corporation. 2006. С. 260. [Електронний ресурс] – 2018. – Режим доступу: <https://docs.microsoft.com/en-us/windows-server/storage/file-server/ntfs-overview>. – Загл. з екрану.
25. Обмеження файлової системи FAT32. Група технічної підтримки Microsoft. [Електронний ресурс] – 2018. – Режим доступу: <http://support.microsoft.com/en-en/184006>. – Загл. з екрану.
26. Файлова система exFAT. Група технічної підтримки Microsoft. [Електронний ресурс] – 2018. – Режим доступу: <http://macdaily.me/reviews/exfat-universal-file-system>. – Загл. з екрану.
27. Shullich, R. Reverse Engineering the Microsoft Extended FAT File System (exFAT). *SANS Institute*. 2009. Р. 86.
28. Description of the FAT32 File System: Microsoft Knowledge Base Article. [Електронний ресурс] – 2018. – Режим доступу: <http://support.microsoft.com/kb/154997>. – Загл. з екрану.
29. FAT File System. [Електронний ресурс] – 2020. – Режим доступу: https://www.keil.com/pack/doc/mw/FileSystem/html/fat_fs.html. – Загл. з екрану.
30. FAT File Systems. FAT32, FAT16, FAT12. NTFS.com. [Електронний ресурс] – 2020. – Режим доступу: https://www.ntfs.com/fat_systems.htm. – Загл. з екрану.
31. Порівняння файлових систем NTFS і FAT32. Група технічної підтримки Microsoft. [Електронний ресурс] – 2018. – Режим доступу: <http://windows.microsoft.com/uk-ua/windows7/comparing-ntfs-and-fat32-file-systems>. – Загл. з екрану.
32. Сравнение файловых систем Windows Embedded Standard 2009. Група технічної підтримки Microsoft. [Електронний ресурс] – 2017. – Режим доступу: [https://msdn.microsoft.com/ru-ru/library/bb521542\(v=winembedded.51\).aspx](https://msdn.microsoft.com/ru-ru/library/bb521542(v=winembedded.51).aspx). – Загл. з екрану.

33. Overview of FAT, HPFS, and NTFS File Systems. Microsoft Knowledge Base Article. [Электронный ресурс] – 2017. – Режим доступа: <http://support.microsoft.com/kb/100108/>. – Загл. з екрану.
34. Shekhanin K., Gorbenko Y., Gorbachova L., Kuznetsov, A. Study of storage devices properties for steganographic data hiding in cluster file systems. *Radiotekhnika*. 2020. Vol. 203. P. 109–120.
35. Liu S. F., Pei S., Huang X. Y., Tian L.. File hiding based on FAT file system. *2009 IEEE International Symposium on IT in Medicine & Education*. Jinan. 2009. P. 1198 – 1201.
36. Davis J., MacLean J. and Dampier D.. Methods of Information Hiding and Detection in File Systems. *2010 Fifth IEEE International Workshop on Systematic Approaches to Digital Forensic Engineering*. Oakland. CA. 2010, P. 66 – 69.
37. Shekhanin, K., Kolhatin, A., Kuznetsova, K., Kavun, S.. Steganographic hiding information in a file system structure. *2018 International Conference on Information and Telecommunication Technologies and Radio Electronics, UkrMiCo 2018 – Proceeding*. September 2018. No 9047551. [Электронный ресурс] – 2018. – Режим доступа: <https://www.scopus.com/record/display.uri?eid=2-s2.0-85083488842&origin=inward>. – Загл. з екрану.
38. Schöttle P., Böhme R. Game Theory and Adaptive Steganography. *IEEE Transactions on Information Forensics and Security*. 2016. Vol. 11, P. 760–773. [Электронный ресурс] – 2018. – Режим доступа: <https://doi.org/10.1109/TIFS.2015.2509941>. – Загл. з екрану.
39. Shekhanin K., Kuznetsov A., Krasnobayev V., Smirnov O. Detecting Hidden Information. *IJCNIS*. 2020. Vol. 12, P. 33–43.
40. Aycock J., de Castro D.M.N. Permutation Steganography in FAT Filesystems. *Transactions on Data Hiding and Multimedia Security X. Springer*, 2015. P. 92–105.
41. Moulin P. Information-theoretic analysis of information hiding. *IEEE TRANSACTIONS ON INFORMATION THEORY*. 2003. Vol. 49, Iss. 3. P. 43.

42. Petitcolas F., Anderson R. J., Kuhn M. G. Information Hiding. *Proceedings IEEE*. 1999. Vol. 87, №. 7. P. 1069-1078.
43. Anderson R. Workshop on Information Hiding: Lecture Notes in Computer Science. *Springer-Verlag*, Cambridge. 1996. Vol. 78. №. 4. P. 578 -601.
44. Kuznetsov A., Shekhanin K., Kolhatin A., Mikheev I., Belozertsev I. Hiding data in the structure of the FAT family file system. *2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT)*. Kyiv, Ukraine. 2018. P. 337 – 342.
45. Kuznetsov A., Shekhanin K., Kolgatin A., Kuznetsova K., Demenko Y. Hiding data in the file structure. *Комп'ютерні науки та кібербезпека*. 2018. № 1 (9). С. 43 – 52.
46. Anderson R. J. The steganographic file system. *Proceedings of the Second International Workshop on Information Hiding*. London, UK. 1998. Vol. 32, No.16. P. 73 – 82.
47. Khan H., Javed M., Khayam S.A., Mirza F.. Evading Disk Investigation and Forensics using a Cluster-Based Covert Channel. *National University of Science & Technology (NUST)*. Pakistan. Vol. 44.
48. Manoj I.V.S. Cryptography and Steganography. *IJCA*. 2010. Vol. 1. P. 63–68.
49. Shekhanin, K., Kuznetsov, A., Krasnobayev, V., Smirnov, O: Detecting hidden information in FAT. *International Journal of Computer Network and Information Security*. 2020. Vol. 12, No. 3, P. 33-43.
50. Noskov A. Analysis of Network Protocols: The Ability of Concealing the Information. *Computer and Network Security*. 2020. [Електронний ресурс] – 2018. – Режим доступу: <https://doi.org/10.5772/intechopen.88098>. – Загл. з екрану.
51. Ojeniyi J. A., Edward E. O., Abdulhamid S. M. Security Risk Analysis in Online Banking Transactions: Using Diamond Bank as a Case Study. *International Journal of Education and Management Engineering*. 2019. Vol. 9, No. 2. P. 1 – 14.
52. Fataliyev T. K., Mehdiyev S. A.. Analysis and New Approaches to the Solution of Problems of Operation of Oil and Gas Complex as Cyber-Physical System. *International*

Journal of Information Technology and Computer Science. 2018. Vol. 10, No. 11. P. 67 – 76.

53. Gnatyuk S.. Critical Aviation Information Systems Cybersecurity. *Meeting Security Challenges Through Data Analytics and Decision Support, NATO Science for Peace and Security Series, D. Information and Communication Security. IOS Press Ebooks*. 2016. Vol. 47, No. 3. P. 308 – 316.

54. Czekaj, M., Jamro, E., Wiatr, K.. Estimating the memory consumption of a hardware ip defragmentation block. *Electronics*. Switzerland., 2 August 2021. Vol. 10, No. 16. [Електронний ресурс] – 2018. – Режим доступу: <https://www.scopus.com/record/display.uri?eid=2-s2.0-85113160336&origin=resultslist>. – Загл. з екрану.

55. Neuner S., Voyiatzis A.G., Schmiedecker M., Brunthaler S., Katzenbeisser S., Weippl E.R. Time is on my side: Steganography in filesystem metadata. *Digital Investigation*. 2016. Vol. 18. P. 76 – 86.

56. Vokorokos L., Madoš B., Ádám N., Baláž A., Porubán J., Chovancová E. Multi-Carrier Steganographic Algorithm Using File Fragmentation of FAT FS. *COMPUTING AND INFORMATICS*. 2019. Vol. 38. P. 343 – 366.

57. Partitioning, Partition Sizes and Drive Lettering. The PC Guide. [Електронний ресурс] – 2018. – Режим доступу: <http://www.pcguides.com/ref/hdd/file/part.htm>. – Загл. з екрану.

58. Switches: Sector copy. Symantec. [Електронний ресурс] – 2018. – Режим доступу: https://support.symantec.com/en_US/article.TECH107956.html – Загл. з екрану.

59. Zaliskyi M., Odarchenko R. , Gnatyuk S., Petrova Yu., Chaplits A.. Method of traffic monitoring for DDoS attacks detection in e-health systems and networks. *CEUR Workshop Proceedings*. 2018. Vol. 2255. P. 193 – 204.

60. Hassan Z., Odarchenko R., Gnatyuk S., Zaman A., Shah M.. Detection of Distributed Denial of Service Attacks Using Snort Rules in Cloud Computing & Remote

Control Systems. *Proceedings of the 2018 IEEE 5th International Conference on Methods and Systems of Navigation and Motion Control*. 2018. Kyiv, Ukraine. P. 283-288.

61. Rajkumar G. P. Malemath V. S.. Video Steganography: Secure Data Hiding Technique. *International Journal of Computer Network and Information Security*. 2017. Vol. 9, No. 9. P. 38 – 45.

62. Khoroshko, V.A. Shelest, M.E.. Introduction to Computer Steganography. Kyiv, Ukraine. 2002. P. 140.

63. Gribunin, V.G., Okov, I.N., Turintsev, I.V.. Digital Steganography. Solon-Press. 2002. Moscow, Russia. P. 272.

64. Konakhovich, G.F. Puzirenko, O.Yu.. Computer Steganography. Theory and Practice. MK-Press. Kyiv, Ukraine. 2006. P. 288.

65. Шеханін К.Ю., Пшенична С.В., Кузнецов О.О. Дослідження обчислювальної складності методів приховування інформації у кластерні стеганосистеми. *Радіоелектроніка*. 2021. №. 206. С. 77 – 87.

66. Dogan S. A New Approach for Data Hiding based on Pixel Pairs and Chaotic Map. *International Journal of Computer Network and Information Security*. 2018. Vol. 10, No. 1. P. 1 – 9.

67. Hosam O.. Attacking Image Watermarking and Steganography. A Survey. *International Journal of Information Technology and Computer Science*. 2019. Vol. 11, No. 3. P. 23 – 37.

68. Ogras H.. An Efficient Steganography Technique for Images using Chaotic Bitstream. *International Journal of Computer Network and Information Security*. 2019. Vol. 11, No. 2. P. 21 – 27.

69. Peskova O.Yu. Halaburda, G.Yu.. The use of network steganography to protect data transmitted via open Internet channels, *Inform. systems for scientific research (IMS-2012), XV Conf. Internet and modern society (IMS-2012)*. St. Petersburg, Russia: Nat. ex. un-t inform. technology, mechanics and optics. 2012. P. 348 – 354. [Електронний ресурс]

– 2019. – Режим доступа: <http://ojs.ifmo.ru/index.php/IMS/article/download/132/132> – Загл. з экрану.

70. Smith G. StegFS: a steganographic file system. *Proceedings of the 19th International Conference on Data Engineering*. 2003. Vol. 1, No. 8. P. 657 – 668.

71. Mao D. GBDE - GEOM Based Disk Encryption. *Poul-Henning Kamp*. 2016. Vol. 9, No. 1. P. 57 – 68.

72. Anderson R. J. Stretching the Limits of Steganography. *Proceedings published by Springer as Lecture Notes in Computer Science*. 1997. Vol. 1174. No 52. P. 39–48.

73. Samanta D.. Classic Data Structures. *Prentice Hall India Pvt*. 2004. Vol. 480. P. 76 – 127.

74. Ousterhout J.K.. Scheduling Techniques for Concurrent Systems. *Proceedings of Third International Conference on Distributed Computing Systems*. USA. 1983. P. 22 – 30.

75. Fraczek W., Mazurczyk W., Szczypiorski K. Stream Control Transmission Protocol Steganography. *2010 International Conference on Multimedia Information Networking and Security*. 2010. P. 829 – 834.

76. Childs L.N. Cryptology and Error Correction: An Algebraic Introduction and Real-World Applications. *Springer International Publishing*, Cham. 2019. [Электронный ресурс] – 2019. – Режим доступа: <https://doi.org/10.1007/978-3-030-15453-0> – Загл. з экрану.

77. Yahya A. Introduction to Steganography. *Steganography Techniques for Digital Images*. *Springer International Publishing*, Cham. 2019. P. 1–7. [Электронный ресурс] – 2019. – Режим доступа: https://doi.org/10.1007/978-3-319-78597-4_1 – Загл. з экрану.

78. Yahya A. Steganography Techniques. *Steganography Techniques for Digital Images*. *Springer International Publishing*, Cham. 2019. P. 9–42. [Электронный ресурс] – 2019. – Режим доступа: https://doi.org/10.1007/978-3-319-78597-4_2 – Загл. з экрану.

79. Fridrich J. Steganography in Digital Media: Principles, Algorithms, and Applications. *Cambridge University Press*. Cambridge, New York. 2009.
80. Mazurczyk W., Smolarczyk M., Szczypiorski K. On information hiding in retransmissions. *Telecommun Syst.* 2013. Vol. 52. P. 1113 – 1121.
81. Eckstein K. Data hiding in journaling file systems. *Refereed Proceedings of the 5th Annual Digital Forensic Research Workshop*, New Orleans. Louisiana, USA. 2005. Vol.52. No.17. P. 1 – 8.
82. Anderson R. J. Tamper Resistance — a Cautionary Note. *Proceedings of the Second Usenix Workshop on Electronic Commerce*. 2008. Vol. 32, Iss. 96. P. 1 – 11.
83. Fraczek W., Mazurczyk W., Szczypiorski K. How Hidden Can be Even More Hidden? *2011 Third International Conference on Multimedia Information Networking and Security*. 2011. P. 581 – 585.
84. Szczypiorski K. Hidden communication system for corrupted networks. *International Multi-Conference on Advanced Computer Systems*. 2003. P. 31 – 40.
85. Klima R.E., Klima R., Sigmon N.P., Sigmon N., Klima R., Sigmon N.P., Sigmon N.. Cryptology: Classical and Modern. *Chapman and Hall/CRC*. 2018. [Электронный ресурс] – 2019. – Режим доступа: <https://doi.org/10.1201/9781315170664> – Загл. з экрана.
86. Delfs H., Knebl H. Introduction to Cryptography. Springer Berlin Heidelberg. Berlin, Heidelberg. 2015. [Электронный ресурс] – 2019. – Режим доступа: <https://doi.org/10.1007/978-3-662-47974-2> – Загл. з экрана.
87. Qin J., Luo Y., Xiang X., Tan Y., Huang H. Coverless Image Steganography. 2019. Vol. 7. P. 171372 – 171394. [Электронный ресурс] – 2019. – Режим доступа: <https://doi.org/10.1109/ACCESS.2019.2955452> – Загл. з экрана.
88. Kim C.R., Lee S.H., Lee J.H., Park J.-I. Blind decoding of image steganography using entropy model. *Electronics Letters*. 2018. Vol. 54. P. 626–628.
89. Rowland C.H. Covert channels in the TCP/IP protocol suite. [Электронный ресурс] – 2020. – Режим

доступу: <https://firstmonday.org/ojs/index.php/fm/article/download/528/449?inline=1> –
Загл. з екрану.

90. Mazurczyk W., Lubacz J.. LACK – a VoIP steganographic method. *Telecommun Syst.* 2010. Vol. 45. P. 153–163.

91. Cox I., Miller M., Bloom J., Fridrich J., Kalker T. Digital Watermarking and Steganography. *2nd Ed. Morgan Kaufmann.* Amsterdam, Boston. 2007.

92. Cauich E., Gómez Cárdenas R., Watanabe R. Data Hiding in Identification and Offset IP Fields. *Advanced Distributed Systems.* 2005. Springer, Berlin, Heidelberg. P. 118–125. [Електронний ресурс] – 2020. – Режим доступу: https://doi.org/10.1007/11533962_11 – Загл. з екрану.

93. Gorbenko, I.D., Gorbenko, Yu.I. Applied Cryptology. *Theory. Practice. Application.* 2013. Kharkiv, Ukraine: Fort. P. 880.

94. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.. Handbook of Applied Cryptography. *CRC Press.* 1997. P. 794.

95. Ferguson N., Schneier B.. Practical Cryptography. *John Wiley & Sons.* 2003. P. 432.

96. Johnson D., Ketel M.. IoT: Application Protocols and Security. *International Journal of Computer Network and Information Security.* 2019. Vol. 11, No. 4 P. 1 – 8.

97. Mikhailov A.P., Petrov A.P., Kalinichenko M.I., Polyakov S.V.. Modeling the simultaneous distribution of legal and counterfeit copies of innovative products. *Mathematical Models and Computer Simulations.* 2014. Vol. 6, No. 1. P. 25 – 31.

98. Gnatyuk S., Aleksander M., Vorona P., Polishchuk Yu., Akhmetova J.. Network-centric Approach to Destructive Manipulative Influence Evaluation in Social Media. *CEUR Workshop Proceedings.* 2019. Vol. 2392. P. 273 – 285.

99. Горбенко І.Д., Горбенко Ю.І. Прикладна криптологія. Теорія. Практика. Застосування : підручник для вищих навч. закладів. Харків : Форт, 2013. С. 880.

1. Кузнецов А.А., Швагер А.С., Фесенко Д.А. Соккрытие данных в кластерных файловых системах. *Радиотехника.* 2015. № 181. С. 86 – 100.

ДОДАТОК А. СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІ

***Публікації у періодичних наукових закордонних виданнях, проіндексованих у
базах даних Web of Science Core Collection та/або Scopus:***

1. Shekhanin K., Kuznetsov A., Krasnobayev V., Smirnov O. Detecting hidden information in FAT. *International Journal of Computer Network and Information Security*. 2020. Vol. 12, Iss. 3, P. 33 – 43 (Hong Kong, Scopus).

DOI: 10.5815/ijcnis.2020.03.04

URL: <https://www.scopus.com/record/display.uri?eid=2-s2.0-85086029655&origin=resultslist>;

(Особистий внесок здобувача: дослідження статистичних властивостей розподілу файлів за рівнем фрагментації, дослідження стійкості до детектування методів приховування інформації).

2. Shekhanin K.Yu., Kolhatin A.O., Demenko E.E., Kuznetsov A. A. On Hiding Data Into the Structure of the FAT Family File System. *Telecommunications and Radio Engineering*. 2019. Vol. 78, Iss. 11, P. 973 – 985 (United States, Scopus).

DOI: 10.1615/TelecomRadEng.v78.i11.50;

URL <https://www.scopus.com/record/display.uri?eid=2-s2.0-85070406462&origin=resultslist>;

(Особистий внесок здобувача: розробка та аналіз методу підвищення пропускної здатності у кластерних стеганосистемах).

***Публікації у виданнях, включених до переліку фахових видань України з
присвоєнням категорії «Б»:***

3. Шеханін К.Ю., Пшенична С.В., Кузнецов О.О. Дослідження обчислювальної складності методів приховування інформації у кластерні стеганосистеми. *Радіoeлектроніка*. 2021. Вип. 206. С. 77 – 87.

URL: <http://rt.nure.ua/issue/archive>;

(Особистий внесок здобувача: розробка удосконалених методів приховування інформації).

4. Шеханін, К., Горбенко, Ю., Горбачова, Л., Кузнецов, О. Дослідження властивостей носіїв інформації для стеганографічного приховування даних в кластерних файлових системах. *Радіотехніка*. 2020. Вип. 203. С. 109 – 120.

DOI: 10.30837/rt.2020.4.203.10

URL: <http://rt.nure.ua/article/view/229343>;

(Особистий внесок здобувача: дослідження технологій фізичних носіїв інформації).

Публікації у виданнях, включених до переліку фахових видань України

5. Кузнецов О., Тимченко В., Лисицький К., Родінко М., Луценко М., Шеханін К., Колгатін А.. Дослідження швидкодії та статистичної безпеки алгоритмів криптографічного гешування. *Радіотехніка*. 2019. Вип. 198. С. 75 – 95.

DOI: 10.30837/rt.2019.3.198.06;

URL: <http://rt.nure.ua/article/view/184661>;

(Особистий внесок здобувача: аналіз швидкодії та обчислювальної складності алгоритмів гешування).

6. Шеханін К., Колгатін А., Деменко Є., Кузнецов О. Приховування даних у структуру файлової системи сімейства FAT. *Радіотехніка*. 2018. Вип. 193. С. 169 – 178.

URL: <http://rt.nure.ua/article/view/175804>;

(Особистий внесок здобувача: опис та порівняльний аналіз існуючих стеганографічних методів у структурі кластерних файлових систем).

Публікації, які засвідчують апробацію матеріалів дисертації:

7. Steganographic hiding information in a file system structure / K. Shekhanin, A. Kolhatin, K. Kuznetsova, S Kavun // 2018 International Conference on Information and Telecommunication Technologies and Radio Electronics : Proceeding of UkrMiCo 2018, 10–14 September 2018, Odessa. (Scopus).

DOI: 10.1109/UkrMiCo43733.2018.9047551

URL: <https://www.scopus.com/record/display.uri?eid=2-s2.0-85083488842&origin=resultslist>;

(Особистий внесок здобувача: опис методу підвищення пропускну́ї здатності методів приховування у кластерні файлові системи).

8. Hiding data in the structure of the FAT family file system / Kuznetsov A., Shekhanin K., Kolhatin A. [etc.] // Dependable Systems, Services and Technologies (DESSERT) : conference proceedings of 2018 IEEE 9th International Conference, 24–27 May 2018, Kyiv, 2018 / Kyiv, 2018. – P. 337 – 342. (Scopus).

DOI: 10.1109/DESSERT.2018.8409155;

URL: <https://www.scopus.com/record/display.uri?eid=2-s2.0-85050684345&origin=resultslist>;

(Особистий внесок здобувача: порівняльний аналіз методів приховування інформації у структуру файлових систем FAT).

Публікації, які додатково відображають наукові результати дисертації:

Публікації у виданнях України:

9. Kuznetsov A., Shekhanin K., Kolgatin A., Kuznetsova K., Demenko Y. Hiding data in the file structure. *Комп'ютерні науки та кібербезпека*. 2018. № 1 (9). С. 43-52.

URL: <https://periodicals.karazin.ua/cscs/article/view/12013>

(Особистий внесок здобувача: описані та досліджені методи приховування інформації у структуру файлових систем).

Патенти:

10. Спосіб стеганографічного приховування інформації за допомогою технологій 3d-друку: пат. 131986, Україна: № u201808262; заявл. 26.07.2018; опубл. 11.02.2019.

(Особистий внесок здобувача: дослідження стеганографічних властивостей заявленого методу).

11. Спосіб стеганографічного приховування даних в кластерних файлових системах: пат. 132302, Україна: № u201808261; заявл. 26.07.2018; опубл. 25.02.2019.
(Особистий внесок здобувача: розробка способу стеганографічного приховування даних в кластерних файлових системах).

Розділ у колективній монографії:

12. Detecting Hidden Information in FAT: collective monograph / Kuznetsov A., Shekhanin K., Smirnov O., Chepurko I. – *ISCI'2019: Information Security in Critical Infrastructures* : collective monograph. USA. 2019. P. 412 – 429.

DOI 10.17605/OSF.IO/E8PY6;

URL: <https://osf.io/u3bt4>;

(Особистий внесок здобувача: розробка методу оцінки пропускної здатності та стійкості до детектування методів приховування)

ДОДАТОК Б. ОПИС КЛАСУ ФАЙЛОВОЇ СИСТЕМИ

```

export class FileSystem implements IFileSystem {

  constructor(options: IFSOption) {
    this.type = options.type ? options.type : FS_TYPES.WITHOUT_TYPE;

    this.clusters = options.size ? this.generateClusters(options.size) : this.generateClusters(FS_CONSTANTS.FS_SIZE_DEFAULT);

    this.files = this.generateFiles(options.fileOptions);
  }

  type = FS_TYPES.WITHOUT_TYPE;
  clusters: Cluster[] = [];
  clustersInMemory: Cluster[] = [];
  files: File[] = [];

  private generateClusters = (size: number): Cluster[] => {
    const clusters = (new Array(size)).fill("");
    return clusters.map((_, index) => new Cluster(index + FS_CONSTANTS.START_FS_INDEX, this));
  }

  private generateFiles = (fileOptions?: IFileOption[]): File[] => {
    if (!fileOptions) return [];
    const files: File[] = [];
    fileOptions.forEach(f => {
      files.push(new File({ fs: this, name: f.name || `File_${files.length}`, color: f.color, sizeInClusters: f.sizeInClusters }));
    });
    return files;
  }

  public swapClusters = (first: Cluster, second: Cluster) => {
    const tmp = { ...first };

    first.file = second.file;
    first.fileIndex = second.fileIndex;

    second.file = tmp.file;
    second.fileIndex = tmp.fileIndex;
  }

  public setClusters = (newClusters: Cluster[]) => this.clusters = newClusters;

```

```

public getMinState = (): IMinificatedCluster[] => this.files.reduce(
  (result, f, index) =>
    [
      ...result,
      ...f.clusters.map(
        c => (
          {
            fsIndex: c.fsIndex,
            fileIndex: c.fileIndex,
            file: index
          }
        )
      )
    ], [] as IMinificatedCluster[])
  .sort((a, b) => a.fsIndex - b.fsIndex);

```

```

public readClusterToMemory = (fsIndex: number) => {
  const cluster = this.clusters.find(c => c.fsIndex === fsIndex);
  if (!cluster) {
    console.warn("Cluster was not found!");
    return;
  }
  cluster.file?.removeCluster(cluster);
  this.clustersInMemory.push({ ...cluster });
  cluster.resumeFile();
}

```

```

public writeClusterFromMemory = (fsIndex: number) => {
  const cluster = this.clusters.find(c => c.fsIndex === fsIndex);
  const [memoriedCluster] = this.clustersInMemory;

  if (!memoriedCluster.file || memoriedCluster.fileIndex === undefined) {
    console.warn("Memocluster doesn't have file or file index!");
    return;
  }
  if (!cluster) {
    console.warn("Cluster was not found!");
    return;
  }
  cluster.bindToFile(memoriedCluster.file, memoriedCluster.fileIndex);
  memoriedCluster.file.addCluster(cluster);
  this.clustersInMemory.shift();
}
}

```

ДОДАТОК В. ОПИС КЛАСУ ФАЙЛУ

```

export class File implements IFile {
  constructor(fileOptions: IFileOptions) {
    this.name = fileOptions.name ? fileOptions.name : `File_${FILE_COUNTER}`; FILE_COUNTER++;
    this.fs = fileOptions.fs;
    this.color = takeColor(COLORS, fileOptions.color);
    this.generateClusters(fileOptions.sizeInClusters)
  }

  name = "";
  fs;
  color = "";
  clusters: FileCluster[] = [];
  needForBind = 0;

  private generateClusters = (numberClusters = 10) => {
    while (this.clusters.length < numberClusters) {
      const freeCluster = this.fs.clusters.find(c => c.isFree());
      if (!freeCluster) throw new Error(`File ${this.name} cannot find free cluster in file system ${this.fs}`);

      freeCluster.bindToFile(this, this.clusters.length);
      this.clusters.push(freeCluster);
    }
  };

  public setUpNewClusters = (clustersIndex: number[], retrospective = false) => {
    if (!retrospective) this.cleareClusters();
    else this.needForBind = this.clusters.length > clustersIndex.length ? this.clusters.length - clustersIndex.length : 0;

    this.clusters.forEach(c => c.resumeFile());
    this.clusters = [];

    clustersIndex.forEach(cI => {
      const foundCluster = this.fs.clusters[cI];
      if (!foundCluster) throw new Error(`Cluster ${cI} is not found`);
      foundCluster.bindToFile(this, this.clusters.length);
      this.clusters.push(foundCluster);
    });
  }

  public cleareClusters = () => {
    this.clusters.forEach(c => { if (c instanceof Cluster) c.resumeFile() });
    this.clusters = [];
  }

```

```

}
public addMissingClusters = (numberOfClusters: number, availableClusterIndexes?: number[]) => {
  let availableIndexes = availableClusterIndexes || this.fs.clusters.map(c => c.fsIndex);
  availableIndexes.forEach(aI => {
    if (numberOfClusters <= 0) return;
    const isFreeCluster = this.fs.clusters.find(c => c.fsIndex === aI && c.isFree());
    if (!isFreeCluster) return;
    isFreeCluster.bindToFile(this, this.clusters.length);
    this.clusters.push(isFreeCluster)
    numberOfClusters--;
  })
};
public removeCluster = (cluster: Cluster) => {
  const index = this.clusters.findIndex(c => c.file === cluster.file && c.fileIndex === cluster.fileIndex);
  this.clusters.splice(index, 1);
};
public addCluster = (cluster: Cluster) => {
  this.clusters.push(cluster);
  this.clusters = this.clusters.sort((a, b) => a.fsIndex - b.fsIndex);
};
}

```

ДОДАТОК Г. ОПИС КЛАСУ КЛАСТЕРУ

```
export class Cluster implements ICluster {  
  constructor(fsIndex: number, fs: IFileSystem) {  
    this.fs = fs;  
    this.fsIndex = fsIndex;  
  }  
  
  fs;  
  fsIndex;  
  
  file?: File;  
  fileIndex?: IFileIndex;  
  
  bindToFile = (file: File, fileIndex: number) => {  
    this.file = file;  
    this.fileIndex = fileIndex;  
  }  
  
  resumeFile = () => {  
    this.fileIndex = undefined;  
    this.file = undefined;  
  };  
  
  isFree = () => this.file === undefined && this.fileIndex === undefined;  
}
```


ДОДАТОК Г. ОПИС КЛАСУ СТАТИСТИКИ

```

class Statistics implements IStatistics {
  fileSystemStart?: FileSystem;
  fileSystemEnd?: FileSystem;
  message?: Boolean[] | string;

  permutation?: number[][];
  headerMoves?: number;
  clustersRead?: number;
  clustersWrite?: number;
  memorySize?: number;
  fragmentations?: {
    before: number;
    after: number;
  }[];

  setStartState = (state: FileSystem) => {
    this.fileSystemStart = new FileSystem({ size: state.clusters.length, fileOptions: state.files.map(f => ({ color: f.color, name: f.name, sizeInClusters: f.clusters.length }))) );
    return this
  };

  setEndState = (state: FileSystem) => { this.fileSystemEnd = state; return this };

  setMessage = (message: Boolean[] | string) => { this.message = message; return this };

  setPermutation = (permutation: number[][] ) => { this.permutation = permutation; return this }

  calculateClustersRead = () => {
    if (this.clustersWrite) {
      this.clustersRead = this.clustersWrite;
      return this;
    }

    if (this.permutation) {
      this.clustersRead = 0;
      this.permutation.forEach(onePermutation => {
        if (onePermutation.length === 1) return;
        if (this.clustersRead === undefined) this.clustersRead = 0;
        this.clustersRead = this.clustersRead + onePermutation.length;
      })
    } else {
      const movedClusters = this.fileSystemEnd?.clusters
    }
  }
}

```

```

.filter((eC, index) => {
  const sC = this.fileSystemStart?.clusters[index];
  if (!sC) return false;
  if (!eC.file || eC.fileIndex === undefined) return false;
  if (!sC.file || sC.fileIndex === undefined) return false;
  if (
    eC.fsIndex === sC.fsIndex &&
    eC.fileIndex === sC.fileIndex &&
    eC.file.name === sC.file.name
  ) {
    return false;
  }
  return true;
});
this.clustersRead = movedClusters?.length;
}
return this;
}
calculateClustersWrite = () => {
  if (this.clustersRead) {
    this.clustersWrite = this.clustersRead;
    return this;
  }
  this.clustersWrite = 0;
  if (this.permutation) {
    this.permutation.forEach(onePermutation => {
      if (onePermutation.length === 1) return;
      if (this.clustersWrite === undefined) this.clustersWrite = 0;
      this.clustersWrite = this.clustersWrite + onePermutation.length;
    })
  } else {
    const movedClusters = this.fileSystemEnd?.clusters.filter((eC, index) => {
      const sC = this.fileSystemStart?.clusters[index];
      if (!sC) return false;
      if (!eC.file || eC.fileIndex === undefined) return false;
      if (!sC.file || sC.fileIndex === undefined) return false;
      if (
        eC.fsIndex === sC.fsIndex &&
        eC.fileIndex === sC.fileIndex &&
        eC.file.name === sC.file.name
      ) return false;
      return true;
    });
    this.clustersWrite = movedClusters?.length;
  }
  return this;
}

```

```

calculateMemorySize = () => {
  if (this.permutation) {
    this.memorySize = 2;
    return this;
  }

  const movedClusters = this.fileSystemEnd?.clusters.filter((eC, index) => {
    const sC = this.fileSystemStart?.clusters[index];
    if (!sC) return false;
    if (!eC.file || eC.fileIndex === undefined) return false;
    if (!sC.file || sC.fileIndex === undefined) return false;
    if (
      eC.fsIndex === sC.fsIndex &&
      eC.fileIndex === sC.fileIndex &&
      eC.file.name === sC.file.name
    ) return false;
    return true;
  });
  this.memorySize = movedClusters?.length;
  return this;
};

calculateHeaderMoves = () => {
  if (!this.permutation) {
    if (!this.memorySize) {
      console.error("Please set up memorySize");
      return this;
    }
    this.headerMoves = this.memorySize * 2;
    return this;
  }

  let sum = 0;
  let prevIndex: number;
  this.permutation.forEach(onePermutation => {
    if (onePermutation.length === 1) return;
    onePermutation.forEach(value => {
      if (prevIndex !== undefined) sum += Math.abs(value - prevIndex);
      prevIndex = value;
    })
  });

  this.headerMoves = sum;

  return this;
};

```

```

calculateFragmentation = () => {
  console.log("here")
  const before: number[] = [];
  const after: number[] = [];
  this.fragmentations = [];

  this.fileSystemStart?.files.forEach(file => {
    let fragmentationBefore = 1;
    let prevIndex: number;
    file.clusters
      .sort((a, b) => Number(a.fileIndex) - Number(b.fileIndex))
      .forEach(cluster => {
        if (prevIndex === undefined) {
          prevIndex = cluster.fsIndex;
          return;
        }
        fragmentationBefore += (cluster.fsIndex - prevIndex) === 1 ? 0 : 1;
        prevIndex = cluster.fsIndex;
      });
    before.push(fragmentationBefore);
  });

  this.fileSystemEnd?.files.forEach(file => {
    let fragmentationAfter = 1;
    let prevIndex: number;
    file.clusters
      .sort((a, b) => Number(a.fileIndex) - Number(b.fileIndex))
      .forEach(cluster => {
        if (prevIndex === undefined) {
          prevIndex = cluster.fsIndex;
          return;
        }
        fragmentationAfter += (cluster.fsIndex - prevIndex) === 1 ? 0 : 1;
        prevIndex = cluster.fsIndex;
      });
    after.push(fragmentationAfter);
  });

  before.forEach((value, index) => this.fragmentations?.push({
    before: value,
    after: after[index],
  }));
  return this;
}

```

ДОДАТОК Д. ФУНКЦІЇ ОБРОБКИ ПОВІДОМЛЕННЯ У СТЕГАНОВІДПОВІДНОСТІ

```
import { convertBooleanToStegano } from "../convertBooleanToStegano";
import { convertStringToBoolean } from "../convertStringToBoolean";
import { FileSystem } from "../../entities/FileSystem/FileSystem";

export type ISteganoMessage = number[];
export type ISteganoMessageImproved = {
  basic: ISteganoMessage;
  [key: number]: number[];
}
```

```

export const getSteganoMessage = (message: Boolean[] | string, fileSystem: FileSystem): ISteganoMessage => {
  const bitPerCluster = Math.log2(fileSystem.files.length);
  switch (typeof message) {
    case "string": {
      return convertBooleanToStegano(convertStringToBoolean(message), bitPerCluster);
    }
    case "object": {
      return convertBooleanToStegano(message, bitPerCluster);
    }
  };
}

```

```

export const getSteganoMessageImproved = (message: Boolean[] | string, fileSystem: FileSystem): ISteganoMessageImproved => {
  const bitPerCluster = Math.log2(fileSystem.files.length) + 1;
  let fullMessage = [];
  switch (typeof message) {
    case "string": {
      fullMessage = convertBooleanToStegano(convertStringToBoolean(message), bitPerCluster);
      break;
    }
    case "object": {
      fullMessage = convertBooleanToStegano(message, bitPerCluster);
      break;
    }
  };

  const steganoMessage: ISteganoMessageImproved = {
    basic: [],
  }
  fullMessage.forEach(m => {
    let corectBlock = m.toString(2);
    while (corectBlock.length < bitPerCluster) corectBlock = "0" + corectBlock;
    const basicPart = parseInt(corectBlock.slice(0, bitPerCluster - 1), 2);
    const improvedPart = parseInt(corectBlock.slice(-1), 2);

    steganoMessage.basic.push(basicPart);
    if (steganoMessage[basicPart] === undefined) steganoMessage[basicPart] = [];
    steganoMessage[basicPart].push(improvedPart)
  });

  return steganoMessage;
}

```

ДОДАТОК Е. ФУНКЦІЇ РОБОТИ ІЗ ПЕРЕСТАНОВКОЮ

```

import { IMinificatedCluster } from "../../entities/FileSystem/FileSystem";

export type IPermutation = number[];

type IMarkedCluster = IMinificatedCluster & { selected: boolean }

const sortStates = (a: IMinificatedCluster, b: IMinificatedCluster) => a.fsIndex - b.fsIndex;

export const getPermutation = (initState: IMinificatedCluster[], endState: IMinificatedCluster[]): IPermutation[] => {
  const permutations: IPermutation[] = [];

  const localInitState: IMarkedCluster[] = initState.sort(sortStates).map(s => ({ ...s, selected: false }));
  const localEndState: IMarkedCluster[] = endState.sort(sortStates).map(s => ({ ...s, selected: false }));

  let currentCluster: IMarkedCluster = localInitState[0];
  let index = 0;
  const getEndStateIndex = (s: IMarkedCluster) => s.file === currentCluster.file && s.fileIndex === currentCluster.fileIndex

  while (index !== -1) {
    const permutation: IPermutation = [];

    while (!localInitState[index].selected) {
      localInitState[index].selected = true;
      currentCluster = localInitState[index];

      permutation.push(index);

      index = localEndState.findIndex(getEndStateIndex);
    }
    permutations.push(permutation);
    index = localInitState.findIndex(s => !s.selected);
  }

  return permutations;
}

```

```

import { IPermutation } from "../getPermutation";
import { FileSystem } from "../../entities/FileSystem/FileSystem";

export const usePermutation = (permutations: IPermutation[], fileSystem: FileSystem) => {
  permutations.forEach(permutation => {
    if (permutation.length === 1) return;

    fileSystem.readClusterToMemory(permutation[0]);
    permutation.slice(1).forEach((iPermutation) => {
      fileSystem.readClusterToMemory(iPermutation);
      fileSystem.writeClusterFromMemory(iPermutation);
    });
    fileSystem.writeClusterFromMemory(permutation[0]);
  });
}

```


ДОДАТОК Є. ФУНКЦІЇ БАЗОВОГО МЕТОДУ ПРИХОВУВАННЯ

```

export const I_Basic = (message: Boolean[] | string, fileSystem: FileSystem) => {
  let steganoMessage = getSteganoMessage(message, fileSystem);

  if (!isEnoughBasic(steganoMessage, fileSystem)) throw new CustomError({
    message: ErrorTypes.MessageToLarge,
    basic: steganoMessage,
    fileSystem: fileSystem,
  });
  const { files } = fileSystem;

  const spreadClustersIndex: { [key: number]: number[] } = {};
  steganoMessage.forEach((steganoBlock, index) => {
    if (spreadClustersIndex[steganoBlock] === undefined) spreadClustersIndex[steganoBlock] = [];
    spreadClustersIndex[steganoBlock].push(index)
  });

  const newFs = new FileSystem({ size: fileSystem.clusters.length, fileOptions: files.map(f => ({ color: f.color, name: f.name, sizeInClusters: 0 })), [] });

  newFs.files.forEach((file, index) => {
    if (spreadClustersIndex[index] === undefined) spreadClustersIndex[index] = [];
    file.setUpNewClusters(spreadClustersIndex[index], true);
  });

  let availableClusterIndexes = files.reduce((res, f) => ([...res, ...f.clusters.map(c => c.fsIndex)]), [] as number[]);

  newFs.files.forEach((f, index) => f.addMissingClusters(fileSystem.files[index].clusters.length - f.clusters.length, availableClusterIndexes));

  return newFs;
}

```

```

export const II_Basic = (message: Boolean[] | string, fileSystem: FileSystem) => {
  let steganoMessage = getSteganoMessage(message, fileSystem);

  if (!isEnoughBasic(steganoMessage, fileSystem)) throw new CustomError({
    message: ErrorTypes.MessageTooLarge,
    basic: steganoMessage,
    fileSystem: fileSystem,
  });

  const initState = fileSystem.getMinState();
  const fsIndexes = initState.map(iS => iS.fsIndex);
  let endState: IMinificatedCluster[] = [];
  const copyOfInitState: IMinificatedCluster[] = JSON.parse(JSON.stringify(initState));

  steganoMessage.forEach(s => {
    const foundIndex = copyOfInitState.findIndex(iS => iS.file === s);
    endState.push(copyOfInitState[foundIndex]);
    copyOfInitState.splice(foundIndex, 1);
  });

  copyOfInitState
    .sort((a, b) => a.file - b.file)
    .forEach(iS => endState.push(iS));
  endState = endState.map((eS, index) => ({ ...eS, fsIndex: fsIndexes[index] }));

  const permutations = getPermutation(initState, endState);
  usePermutation(permutations, fileSystem);
}

```

```

export const III_Basic = (message: Boolean[] | string, fileSystem: FileSystem) => {
  let steganoMessage = getSteganoMessage(message, fileSystem);

  if (!isEnoughBasic(steganoMessage, fileSystem)) throw new CustomError({
    message: ErrorTypes.MessageTooLarge,
    basic: steganoMessage,
    fileSystem: fileSystem,
  });

  const initState = fileSystem.getMinState();
  const fsIndexes = initState.map(iS => iS.fsIndex);
  let endState: IMinificatedCluster[] = [];
  const copyOfInitState: IMinificatedCluster[] = JSON.parse(JSON.stringify(initState));

  steganoMessage.forEach((s, index) => {
    const foundIndex = copyOfInitState.findIndex(iS => iS.file === s && !iS.moved);
    endState.push({ ...copyOfInitState[foundIndex], fsIndex: fsIndexes[index] });
    copyOfInitState[foundIndex].moved = true;
  });

  copyOfInitState.forEach((currentInitState, index) => {
    if (currentInitState.moved) return;
    const currentEndState = endState[index];
    if (currentEndState) {
      const foundFirstMoved = copyOfInitState.find(iS => iS.moved && !endState.find(eS => eS.fsIndex === iS.fsIndex));
      if (!foundFirstMoved) {
        endState.push(currentInitState);
        return;
      }
      endState.push({ ...currentInitState, fsIndex: foundFirstMoved.fsIndex });
      return;
    }
    endState.push(currentInitState);
  });
  endState = endState.sort((a, b) => a.fsIndex - b.fsIndex);

  const permutations = getPermutation(initState, endState);
  usePermutation(permutations, fileSystem);
}

```

```

export const IV_Basic = (message: Boolean[] | string, fileSystem: FileSystem) => {
  let steganoMessage = getSteganoMessage(message, fileSystem);

  if (!isEnoughBasic(steganoMessage, fileSystem)) throw new CustomError({
    message: ErrorTypes.MessageTooLarge,
    basic: steganoMessage,
    fileSystem: fileSystem,
  });

  const initState = fileSystem.getMinState();
  const fsIndexes = initState.map(iS => iS.fsIndex);
  let endState: IMinificatedCluster[] = [];
  const copyOfInitState: IMinificatedCluster[] = JSON.parse(JSON.stringify(initState));

  steganoMessage.forEach((s, index) => {
    const foundUnmovedIndex = copyOfInitState.findIndex((iS, iSIndex) => iS.file === s && !iS.moved && index === iSIndex);
    const foundIndex = foundUnmovedIndex === -1 ? copyOfInitState.findIndex(iS => iS.file === s && !iS.moved) : foundUnmovedIndex;
    endState.push({ ...copyOfInitState[foundIndex], fsIndex: fsIndexes[index] });
    copyOfInitState[foundIndex].moved = true;
  });

  copyOfInitState.forEach((currentInitState, index) => {
    if (currentInitState.moved) return;
    const currentEndState = endState[index];
    if (currentEndState) {
      const foundFirstMoved = copyOfInitState.find(iS => iS.moved && !endState.find(eS => eS.fsIndex === iS.fsIndex));
      if (!foundFirstMoved) {
        endState.push(currentInitState);
        return;
      }
      endState.push({ ...currentInitState, fsIndex: foundFirstMoved.fsIndex });
      return;
    }
    endState.push(currentInitState);
  });
  endState = endState.sort((a, b) => a.fsIndex - b.fsIndex);

  const permutations = getPermutation(initState, endState);
  usePermutation(permutations, fileSystem);
}

```

ДОДАТОК Ж. ФУНКЦІЇ МОДИФІКОВАНОГО МЕТОДУ ПРИХОВУВАННЯ

```

export const replaceClustersImproved = (fileClusters: Cluster[] | IMinificatedCluster[], message: number[]) => {
  if (fileClusters.length <= message.length) throw new Error("File length should be more than message on 1 cluster");
  const serries = getSerries(message);
  const fileIndexes = fileClusters.map((c: Cluster | IMinificatedCluster) => c.fileIndex).sort();

  let currentClusterIndex = 0;
  serries.forEach(oneSerie => {
    oneSerie.forEach(value => {
      if (value) {
        fileClusters[currentClusterIndex].fileIndex = fileIndexes.pop();
        currentClusterIndex++;
      } else {
        fileClusters[currentClusterIndex].fileIndex = fileIndexes.shift();
        currentClusterIndex++;
      }
    })
  });

  while (fileIndexes.length) {
    fileClusters[currentClusterIndex].fileIndex = fileIndexes.shift();
    currentClusterIndex++;
  }
};

```

```

export const I_Improved = (message: Boolean[] | string, fileSystem: FileSystem) => {
  let { basic, ...rest } = getSteganoMessageImproved(message, fileSystem);

  if (!isEnoughImproved({ basic, ...rest }, fileSystem)) throw new CustomError({
    message: ErrorTypes.MessageTooLarge,
    basic: basic,
    improved: rest,
    fileSystem: fileSystem,
  });

  const { files } = fileSystem;

  const spreadClustersIndex: { [key: number]: number[] } = {};
  basic.forEach((steganoBlock, index) => {
    if (spreadClustersIndex[steganoBlock] === undefined) spreadClustersIndex[steganoBlock] = [];
    spreadClustersIndex[steganoBlock].push(index)
  });

  const newFs = new FileSystem({ size: fileSystem.clusters.length, fileOptions: files.map(f => ({ color: f.color, name: f.name, sizeInClusters: 0 })), [] });

  newFs.files.forEach((file, index) => {
    if (spreadClustersIndex[index] === undefined) spreadClustersIndex[index] = [];
    file.setUpNewClusters(spreadClustersIndex[index], true);
  });

  let availableClusterIndexes = files.reduce((res, f) => ([...res, ...f.clusters.map(c => c.fsIndex)]), [] as number[]);

  newFs.files.forEach((f, index) => f.addMissingClusters(fileSystem.files[index].clusters.length - f.clusters.length, availableClusterIndexes));

  newFs.files.forEach((file, index) => {
    if (!rest[index]) return;
    replaceClustersImproved(file.clusters, rest[index]);
  })

  return newFs;
}

```

```

export const II_Improved = (message: Boolean[] | string, fileSystem: FileSystem) => {
  let { basic, ...rest } = getSteganoMessageImproved(message, fileSystem);

  if (!isEnoughImproved({ basic, ...rest }, fileSystem)) throw new CustomError({
    message: ErrorTypes.MessageTooLarge,
    basic: basic,
    improved: rest,
    fileSystem: fileSystem,
  });

  const initState = fileSystem.getMinState();
  const fsIndexes = initState.map(iS => iS.fsIndex);
  let endState: IMinificatedCluster[] = [];
  const copyOfInitState: IMinificatedCluster[] = JSON.parse(JSON.stringify(initState));

  basic.forEach(s => {
    const foundIndex = copyOfInitState.findIndex(iS => iS.file === s);
    endState.push(copyOfInitState[foundIndex]);
    copyOfInitState.splice(foundIndex, 1);
  });

  copyOfInitState
    .sort((a, b) => a.file - b.file)
    .forEach(iS => endState.push(iS));
  endState = endState.map((eS, index) => ({ ...eS, fsIndex: fsIndexes[index] }));

  const splited = splitByFiles(endState);

  splited.forEach((clusters, index) => {
    if (!rest[index]) return;
    replaceClustersImproved(clusters, rest[index]);
  })

  endState = splited.reduce((result: IMinificatedCluster[], state) => {
    return [...result, ...state]
  });

  const permutations = getPermutation(initState, endState);
  usePermutation(permutations, fileSystem);

  console.log(statistic);
}

```

```

export const III_Improved = (message: Boolean[] | string, fileSystem: FileSystem) => {
  let { basic, ...rest } = getSteganoMessageImproved(message, fileSystem);

  if (!isEnoughImproved({ basic, ...rest }, fileSystem)) throw new CustomError({
    message: ErrorTypes.MessageTooLarge,
    basic: basic,
    improved: rest,
    fileSystem: fileSystem,
  });

  const initState = fileSystem.getMinState();
  const fsIndexes = initState.map(iS => iS.fsIndex);
  let endState: IMinificatedCluster[] = [];
  const copyOfInitState: IMinificatedCluster[] = JSON.parse(JSON.stringify(initState));

  basic.forEach((s, index) => {
    const foundIndex = copyOfInitState.findIndex(iS => iS.file === s && !iS.moved);
    endState.push({ ...copyOfInitState[foundIndex], fsIndex: fsIndexes[index] });
    copyOfInitState[foundIndex].moved = true;
  });

  copyOfInitState.forEach((currentInitState, index) => {
    if (currentInitState.moved) return;
    const currentEndState = endState[index];
    if (currentEndState) {
      const foundFirstMoved = copyOfInitState.find(iS => iS.moved && !endState.find(eS => eS.fsIndex === iS.fsIndex));
      if (!foundFirstMoved) {
        endState.push(currentInitState);
        return
      }
      endState.push({ ...currentInitState, fsIndex: foundFirstMoved.fsIndex });
      return;
    }
    endState.push(currentInitState);
  });

  const splited = splitByFiles(endState);
  splited.forEach((clusters, index) => {
    if (!rest[index]) return;
    replaceClustersImproved(clusters, rest[index]);
  })
  endState = splited.reduce((result: IMinificatedCluster[], state) => {
    return [...result, ...state]
  });

  const permutations = getPermutation(initState, endState);
  usePermutation(permutations, fileSystem);
}

```



```

export const IV_Improved = (message: Boolean[] | string, fileSystem: FileSystem) => {
  let { basic, ...rest } = getSteganoMessageImproved(message, fileSystem);
  if (!isEnoughImproved({ basic, ...rest }, fileSystem)) throw new CustomError({
    message: ErrorTypes.MessageTooLarge,
    basic: basic,
    improved: rest,
    fileSystem: fileSystem,
  });
  const initState = fileSystem.getMinState();
  const fsIndexes = initState.map(iS => iS.fsIndex);
  let endState: IMinificatedCluster[] = [];
  const copyOfInitState: IMinificatedCluster[] = JSON.parse(JSON.stringify(initState));
  basic.forEach((s, index) => {
    const foundUnmovedIndex = copyOfInitState.findIndex((iS, iSIndex) => iS.file === s && !iS.moved && index === iSIndex);
    const foundIndex = foundUnmovedIndex === -1 ? copyOfInitState.findIndex(iS => iS.file === s && !iS.moved) : foundUnmovedIndex;
    endState.push({ ...copyOfInitState[foundIndex], fsIndex: fsIndexes[index], moved: foundUnmovedIndex !== -1 ? false : undefined });
    copyOfInitState[foundIndex].moved = true;
  });
  copyOfInitState.forEach((currentInitState, index) => {
    if (currentInitState.moved) return;
    const currentEndState = endState[index];
    if (currentEndState) {
      const foundFirstMoved = copyOfInitState.find(iS => iS.moved && !endState.find(eS => eS.fsIndex === iS.fsIndex));
      if (!foundFirstMoved) {
        endState.push(currentInitState);
        return;
      }
      endState.push({ ...currentInitState, fsIndex: foundFirstMoved.fsIndex });
      return;
    }
    endState.push(currentInitState);
  });
  endState = endState.sort((a, b) => a.fsIndex - b.fsIndex);
  const splited = splitByFiles(endState);
  splited.forEach((clusters, index) => {
    if (!rest[index]) return;
    replaceClustersImproved(clusters, rest[index]);
  });
  endState = splited.reduce((result: IMinificatedCluster[], state) => {
    return [...result, ...state]
  });
  const permutations = getPermutation(initState, endState);
  usePermutation(permutations, fileSystem);
}

```

ДОДАТОК 3. АКТИ ВПРОВАДЖЕННЯ

ЗАТВЕРДЖУЮ

Проректор з наукової роботи
Харківського національного
університету імені В.Н. Каразіна

 Виктор КАТРИЧ
“21” серпня 2021 р.



АКТ

реалізації наукових досліджень Шеханіна Кирила Юрійовича при проведенні науково-дослідних робіт, які виконувалися у 2018-2020 роках на кафедрі безпеки інформаційних систем і технологій Харківського національного університету імені В.Н. Каразіна

Комісія у складі:

голови комісії – завідувача кафедри безпеки інформаційних систем і технологій доктора технічних наук, доцента Рассомахіна С.Г.

членів комісії

- професора кафедри безпеки інформаційних систем і технологій доктора технічних наук, професора Кузнецова О.О.,
- професора кафедри безпеки інформаційних систем і технологій доктора технічних наук, доцента Олійникова Р.В.

склала дійсний акт про те, що при проведенні науково-дослідних робіт:

- «Аналіз, дослідження, розробка та стандартизація криптографічних систем для захисту інформації в пост-квантовому середовищі, в умовах інформаційних і гібридних війн», господарчий договір № 1-41-18;

які виконувалися у 2018-2020 роках на кафедрі безпеки інформаційних систем і технологій Харківського національного університету імені В.Н. Каразіна реалізовано такі наукові та практичні результати наукових досліджень Шеханіна Кирила Юрійовича:

1. Аналіз стійкості існуючих методів та механізмів КЗІ, практичних методик оцінки стійкості до атак. Реалізовано та впроваджено при підготовці розділу 3 звіту з НДР (Аналіз, дослідження, розробка та стандартизація криптографічних систем для захисту інформації в пост-квантовому середовищі, в умовах інформаційних і гібридних війн);

Впровадження отриманих наукових та практичних результатів дозволяє застосувати вдосконалену модель оцінки криптографічних систем захисту

інформації; отримувати оцінки стійкості до атак на основі вивчення властивостей зменшених моделей шифрів та апроксимації результатів до повнорозмірних версій шифрів.

Голова комісії

Завідувач кафедри безпеки інформаційних систем і технологій
доктор технічних наук, доцент



Сергій РАССОМАХІН

Члени комісії:

Професор кафедри безпеки інформаційних систем і технологій
доктор технічних наук, професор



Олександр КУЗНЕЦОВ

Старший науковий співробітник

Професор кафедри безпеки інформаційних систем і технологій
доктор технічних наук, доцент



Роман ОЛІЙНИКОВ

ЗАТВЕРДЖУЮ

Проректор з науково-педагогічної
роботи Харківського національного
університету імені В.Н. Каразіна

Антон МАНТЕЛЕЙМОНОВ
“12” грудня 2020 р.

АКТ

реалізації наукових досліджень Шеханіна Кирила Юрійовича при проведенні
лекційних та лабораторних занять з дисципліни «Стеганографія», які
виконувалися у 2017-2020 роках на
кафедрі безпеки інформаційних систем і технологій
Харківського національного університету імені В.Н. Каразіна

Комісія у складі:

голови комісії – завідувача кафедри безпеки інформаційних систем і
технологій доктора технічних наук, доцента Рассомахіна С.Г.

членів комісії

- професора кафедри безпеки інформаційних систем і технологій доктора
технічних наук, професора Кузнецова О.О.,
- професора кафедри безпеки інформаційних систем і технологій доктора
технічних наук, доцента Олійникова Р.В.

склала дійсний акт про те, що при проведенні навчальних занять з
дисципліни «Стеганографія», у 2017-2020 роках на кафедрі безпеки
інформаційних систем і технологій Харківського національного університету
імені В.Н. Каразіна реалізовано такі наукові та практичні результати
наукових досліджень Шеханіна Кирила Юрійовича:

1. Аналіз існуючих методів приховування даних у кластерних файлових
системах. Реалізовано та впроваджено при підготовці розділу 4 –
Лінгвістична та технічна стеганографія, тема 13 – Приховування даних
у кластерних файлових системах робочої програми навчальної
дисципліни «Стеганографія»;
2. Опис методу підвищення пропускної здатності у кластерних
стеганоканалах. Реалізовано та впроваджено при підготовці розділу 4 –
Лінгвістична та технічна стеганографія, тема 13 – Приховування даних

у кластерних файлових системах робочої програми навчальної дисципліни «Стеганографія»;

3. Модель оцінки пропускну́ї здатності кластерних стеганоканалів. Реалізовано та впроваджено при підготовці розділу 4 – Лінгвістична та технічна стеганографія, тема 13 – Приховування даних у кластерних файлових системах робочої програми навчальної дисципліни «Стеганографія»;

Впровадження отриманих наукових та практичних результатів дозволяє використовувати описані методи приховування даних у структуру кластерних файлових систем під час проведення лабораторних та практичних занять із дисципліни «Стеганографія», застосовувати вдосконалену модель оцінки стеганографічних систем.

Голова комісії

Завідувач кафедри безпеки інформаційних систем і технологій
доктор технічних наук, доцент

Сергій РАССОМАХІН

Члени комісії:

Професор кафедри безпеки інформаційних систем і технологій
доктор технічних наук, професор

Олександр КУЗНЕЦОВ

Старший науковий співробітник

Професор кафедри безпеки інформаційних систем і технологій
доктор технічних наук, доцент

Роман ОЛІЙНИКОВ