МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кваліфікаційна наукова праця
на правах рукопису

КУЛИНА СЕРГІЙ ВАСИЛЬОВИЧ

УДК 004.056

ДИСЕРТАЦІЯ

МЕТОДИ ТА АЛГОРИТМИ ЗАХИЩЕНОГО РОЗПОДІЛЕНОГО ЗБЕРІГАННЯ ДАНИХ НА ОСНОВІ НАДЛИШКОВОЇ СИСТЕМИ ЗАЛИШКОВИХ КЛАСІВ

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

_____ / Кулина С.В. /

Наукові керівники:

Яцків Василь Васильович, д.т.н., професор

Нємкова Олена Анатоліївна, д.т.н., професор

Львів – 2023

АНОТАЦІЯ

Кулина С.В. Методи та алгоритми захищеного розподіленого зберігання даних на основі надлишкової системи залишкових класів. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 125 Кібербезпека. – Національний університет «Львівська політехніка», Львів, 2023.

Дисертація присвячена розв'язанню актуальної науково-технічної задачі підвищення захищеності та надійності зберігання даних у локальних і мережевих сховищах шляхом розробки методів шифрування та кодування даних на основі розширеного набору модулів системи залишкових класів. Надлишкова система залишкових класів дозволяє здійснити відновлення втрачених даних, що є необхідною умовою для розробки надійних систем зберігання даних.

Дисертаційна робота складається зі вступу, п'яти розділів, висновків, списку використаних джерел та додатків.

Широке впровадження інформаційних систем, побудованих зокрема на основі технологій Інтернет-речей (IoT), в різних галузях зумовило стрімке зростання обсягів даних, які потребують безпечного та надійного зберігання, а системи зберігання даних відіграють ключову роль у забезпеченні доступності та цілісності інформації. У даній роботі автор проводить дослідження актуальної науково-прикладної задачі підвищення захищеності та надійності зберігання даних у локальних та мережевих сховищах на основі надлишкової системи залишкових класів.

У роботі розроблено метод надійного зберігання даних на основі коригуючих кодів системи залишкових класів, який базується на використанні одного перевірочного символу та обчисленні значення геш-функції від файлів залишків, що забезпечило відновлення даних при меншій надлишковість. Подання даних в надлишковій системі залишкових класів та зберігання файлів залишків на різних носіях забезпечує нові можливості для розробки нових та удосконалення існуючих методів шифрування даних.

Об’єктом дослідження є процеси кодування та шифрування інформації у системах зберігання даних на основі надлишкової системи залишкових класів.

Предмет дослідження є методи і алгоритми кодування та шифрування інформації у системах зберігання даних на основі надлишкової системи залишкових класів.

При розв’язку задач у дисертаційній роботі використовувались методи: теорії інформації та кодування, шифрування, теорії чисел, забезпечення цілісності даних, виявлення та виправлення помилок в системі залишкових класів.

У першому розділі **«Сучасний стан та шляхи захисту інформації у системах зберігання даних»** розглянуто існуючі типи кібератак та обґрунтовано переваги децентралізованих систем зберігання інформації над централізованими. Здійснено аналіз програмного забезпечення для шифрування даних, яке дозволяє захистити інформацію від несанкціонованого доступу. На підставі проведеного аналізу встановлено, що використання існуючих методів захисту є недостатнім та зазвичай забезпечує надлишковість, рівну 100% при зберіганні даних, тобто інформація дублюється для забезпечення її доступності та цілісності. Відповідно, для підвищення ефективності системи розподіленого захищеного зберігання даних необхідно розробити методи, які забезпечують меншу надлишковості при заданій надійності, цілісності та конфіденційності даних. Для підвищення надійності зберігання даних, виявлення та виправлення помилок, а також для забезпечення додаткового рівня захисту пропонується використовувати коригуючі коди на основі системи залишкових класів.

У другому розділі **«Методи виявлення та виправлення помилок на основі надлишкової системи залишкових класів»** розглянуто методи прямого та зворотного перетворення даних, а також проаналізовано використання коригуючих кодів на основі розширеного набору модулів системи залишкових класів. Проведено дослідження методів виявлення та виправлення помилок у системах зберігання даних. Запропоновано метод захищеного зберігання даних що ґрунтується на використанні розширеного набору модулів системи

залишкових класів та обчисленні геш-функції і забезпечує зменшення надлишковості та збільшує швидкість зворотного перетворення. Розраховано ймовірність не виявлення помилок в інформаційних символах при використанні одного перевірочного символа різної розрядності, а також досліджено виявлення помилок у послідовностях символів. Проведено оцінку часової складності методів зворотного перетворення системи залишкових класів для систем зберігання даних.

У третьому розділі **«Метод шифрування даних на основі надлишкової системи залишкових класів та псевдовипадкової послідовності»** розраховано оптимальні набори модулів для реалізації систем захищеного зберігання даних та обґрунтовано умови їх вибору. Досліджено криптографічну стійкість подання даних у системі залишкових класів та проведено оцінку криптографічної стійкості їх шифрування. Для оцінки криптографічної стійкості методу шифрування даних, що передаються до віддаленого пристрою зберігання, запропоновано враховувати не тільки загальну криптографічну стійкість на основі китайської теореми про залишки, а й розмір файлів залишків, оскільки при перехопленні повідомлення зловмиснику невідомі розрядності обраних модулів. Для підвищення рівня криптографічної стійкості криптоалгоритму шифрування на основі системи залишкових класів запропоновано удосконалення методу шифрування шляхом зміни позицій залишків із використанням М-послідовності. Проведено дослідження криптографічної стійкості запропонованого методу з класичним методом шифрування даних на основі НСЗК.

У четвертому розділі **«Алгоритми захищеного кодування даних на основі надлишкової системи залишкових класів»** розроблено алгоритми, що забезпечують конфіденційність даних при їх передачі та зберіганні на розподілених носіях, а саме: кодування в системі залишкових класів, шифрування та зберігання залишків, перевірки цілісності файлів залишків, дешифрування та відновлення файлів. Приведено алгоритми для кожного з кроків та детально описано їх застосування. Описано додаткові умови зберігання залишків на фізичних носіях, зокрема залежність від розміру останнього блоку

даних. Виведено математичні формули обчислення кількості помилок в залежності від кількості файлів залишків з підтвердженою цілісністю на основі варіантів пошкодження залишків у системі з $n = 4$. Визначено залежність ефективності відновлення файлу від кількості помилок при різній кількості пошкоджених файлів залишків. Наведено приклад застосування методу відновлення при виникненні трьох помилок у системі із $n = 4$ та при підтвердженні цілісності двох із чотирьох залишків.

У п’ятому розділі **«Прикладна програмна реалізація системи для розподіленого віддаленого зберігання даних»** розроблено архітектуру системи розподіленого захищеного зберігання даних на основі наведених у четвертому розділі алгоритмів. Приведено функціональні схеми розподілу вхідного масиву даних на масиви залишків, модулів запису/зчитування, вибору файлів залишків, зворотного перетворення та відновлення пошкодженого файлу. Представлено прототип програмного продукту на мові програмування Python та описано його інтерфейс. Для підтвердження функціональності програмного рішення представлено розділення обраного файлу на блоки та обчислення залишків по кожному із них. Наведені процедури на мові програмування Python що забезпечують перевірку цілісності при декодуванні. Для підтвердження ефективності програмного рішення проведено дослідження методів кодування та декодування даних на основі реалізованого методу. Проведено порівняння запропонованого методу виявлення та виправлення помилок з існуючим, а саме методом проекцій. Проведене порівняння надлишковості дозволяє зазначити, що розроблений метод для реалізації потребує на один носій менше у порівнянні з відомим методом, забезпечуючи ту саму функціональність, а завдяки порівнянню обчислених значень геш-функції є більш ефективним, зменшуючи складність виявлення та виправлення помилок.

У **висновках** дисертаційної роботи викладено основні результати які випливають з проведених досліджень, представлено та охарактеризовано показники ефективності при використанні запропонованих рішень.

Основні наукові результати дисертації опубліковано в 15 працях, з них 4 статті у наукових фахових виданнях України та 11 публікацій у матеріалах та збірниках доповідей наукових конференцій, з яких дві індексуються у наукометричних базах даних Scopus та Web of Science.

Ключові слова: система залишкових класів, коригуючі коди, захищені системи зберігання даних, розподілене зберігання даних, цілісність даних, шифрування даних, дешифрування даних, конфіденційність даних, відновлення даних, відмовостійкість, псевдовипадкові послідовності, прості числа, інформаційна безпека, кібербезпека.

ABSTRACT

S. V. Kulyna. Methods and algorithms of secure distributed data storage based on redundant residue number system. - Qualification scientific work on the rights of a manuscript.

The thesis for the Philosophy Doctor (Ph.D.) degree in specialty 125 Cybersecurity. – Lviv Polytechnic National University, Lviv, Ukraine, 2023.

The dissertation is dedicated to solving the current scientific and technical problem of enhancing the security and reliability of data storage in local and network repositories by developing methods of encryption and data coding based on an extended set of modules of redundant residue number system. The redundant residue number system allows for the recovery of lost data, which is an essential condition for the development of reliable data storage systems.

The dissertation consists of an introduction, five chapters, conclusions, a list of references used, and appendices.

The widespread adoption of information systems, particularly built upon Internet of Things (IoT) technologies, in various industries has led to a rapid increase in the volume of data requiring secure and reliable storage. Data storage systems play a crucial role in ensuring the availability and integrity of information. In this study, the author addresses the research of a current scientific and applied problem: enhancing

the security and reliability of data storage in local and network repositories using a redundant residue number system.

The study introduces a method for reliable data storage based on error-correcting codes of the redundant residue number system. This method relies on the utilization of a single check symbol and the computation of hash function values from residual files, facilitating data recovery with lower redundancy. The representation of data within the redundant residue number system and the storage of residual files on different media provide new possibilities for the development and improvement of encryption methods.

The research focuses on the processes of encoding and encrypting information within data storage systems based on the redundant residue number system.

The subject of the research encompasses the methods and algorithms for encoding and encrypting information within data storage systems based on the redundant residue number system. The following methods were utilized in solving the tasks within the dissertation work: information theory and coding, encryption, number theory, data integrity assurance, error detection and correction in residue class systems.

In the first chapter, «**Current State and Approaches to Information Protection in Data Storage Systems»**, the existing types of cyberattacks are examined, and the advantages of decentralized information storage systems over centralized ones are substantiated. An analysis of encryption software is conducted to safeguard information from unauthorized access. Based on this analysis, it is established that the use of existing protection methods is insufficient and typically results in redundancy of 100% in data storage, where data is duplicated to ensure availability and integrity. Consequently, to enhance the efficiency of a distributed secure data storage system, methods need to be developed that offer lower redundancy while maintaining the desired reliability, integrity, and data confidentiality. To improve data storage reliability, error detection and correction, as well as an additional layer of protection, it is proposed to employ error-correcting codes based on redundant residue number system.

In the second chapter, «**Methods of Error Detection and Correction Based on Redundant Residue Number System**», methods of direct and inverse data transformation are discussed, along with an analysis of error-correcting codes using an extended set of modules from the redundant residue number system. Research into error detection and correction methods in data storage systems is conducted. A method of secure data storage is proposed, grounded in the use of an extended set of modules from the redundant residue number system and hash functions. This method reduces redundancy and increases the speed of inverse transformation. The probability of error not being detected in information symbols while using a single check symbol of varying bit-length is calculated, along with an exploration of error detection in symbol sequences. The time complexity of inverse transformation methods of the redundant residue number system for data storage systems is evaluated.

In the third chapter, **«Method of Data Encryption Based on Redundant System of Residual Classes and Pseudorandom Sequence»,** optimal sets of modules for implementing secure data storage systems are calculated, and the conditions for their selection are justified. The cryptographic strength of data representation within the redundant residue number system is investigated, and an evaluation of the cryptographic robustness of their encryption is conducted. To assess the cryptographic strength of the data encryption method transmitted to a remote storage device, it is proposed to consider not only the overall cryptographic strength based on the Chinese Remainder Theorem but also the size of the residual files. This is important because eavesdroppers may be unaware of the bit lengths of the chosen modules. To enhance the cryptographic strength of the encryption crypto-algorithm based on the redundant residue number system, an improvement to the encryption method is suggested through rearranging the positions of the residuals using M-sequences. The cryptographic strength of the proposed method is compared with the classical data encryption method based on the redundant residue number system.

In the fourth chapter, **«Algorithms of Secure Data Encoding Based on Redundant System of Residual Classes»,** algorithms ensuring data confidentiality during transmission and storage on distributed media are developed. These include

encoding in the redundant residue number system, encryption and storage of residuals, integrity verification of residual files, decryption, and file recovery. Algorithms for each step are provided along with detailed descriptions of their application. Additional storage conditions for residuals on physical media are described, particularly accounting for the size of the last data block. Mathematical formulas for calculating the number of errors depending on the quantity of residual files with confirmed integrity are derived, focusing on variations of residual corruption in a system with $n = 4$. The relationship between the efficiency of file recovery and the number of errors is determined for different quantities of corrupted residual files. An example of the recovery method's application is provided when three errors occur in a system with $n = 4$, with the integrity of two out of four residuals confirmed.

In the fifth chapter, **«Applied Software Implementation of a System for Distributed Remote Data Storage»,** an architecture for a secure distributed data storage system based on the algorithms outlined in the fourth chapter is developed. Functional diagrams for the distribution of the input data array into residual arrays, write/read modules, selection of residual files, inverse transformation and corrupted file recovery are provided. A prototype of the software product is presented in the Python programming language, along with a description of its interface. To validate the functionality of the software solution, the division of a selected file into blocks and the computation of residuals for each block are presented. Python procedures ensuring integrity verification during decoding are provided. To confirm the efficiency of the software solution, research into encoding and decoding methods based on the implemented method is conducted. A comparison between the proposed error detection and correction method and an existing one, namely the projection method, is carried out. The comparison of redundancy indicates that the developed method requires one less medium compared to the known method while maintaining the same functionality. Additionally, through a comparison of computed hash function values, the developed method proves to be more efficient, reducing error detection and correction complexity.

The **conclusions** of the dissertation work present the main results arising from the conducted research. The effectiveness indicators when using the proposed solutions are presented and characterized.

The main scientific outcomes of the dissertation have been published in 15 works, including 4 articles in specialized scientific journals in Ukraine and 11 publications in materials and proceedings of scientific conferences, of which two are indexed in Scopus and Web of Science databases.

Keywords: redundant residue number system, error-correcting codes, secure data storage systems, distributed data storage, data integrity, data encryption, data decryption, data privacy, data recovery, fault tolerance, pseudorandom sequences, prime numbers, information security, cybersecurity.

СПИСОК ПРАЦЬ ОПУБЛІКОВАНИХ ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Статті у фахових виданнях України:

1. Яцків, В. В., Кулина, С. В. (2019). Метод надійного зберігання даних на основі надлишкової системи залишкових класів. *Вісник Хмельницького національного університету. Технічні науки*, 6, 98–104. http://journals.khnu.km.ua/vestnik/wp-content/uploads/2021/01/20-9.pdf.

2. Kulyna, S. (2022). Evaluation of the reverse transformation methods complexity of the residual number system for secure data storage. *Вісник Тернопільського Національного Технічного Університету*, 107(3), 21–28. https://doi.org/10.33108/visnyk_tntu2022.03.021.

3. Кулина, С. В. (2022). Виявлення та виправлення помилок у захищених системах зберігання даних на основі обчислення проекцій числа. *Informatics and Mathematical Methods in Simulation*, 12, 337-345. http://nbuv.gov.ua/UJRN/Itmm_2022_12_4_10.

4. Кулина, С. В. (2023). Система розподіленого захищеного зберігання даних. *Вісник Львівського державного університету безпеки життєдіяльності*, 27, 48-59. https://doi.org/10.32447/20784643.27.2023.06.

Матеріали міжнародних наукових та науково-практичних конференцій, збірники яких входять до міжнародних наукометричних баз:

5. Yatskiv, V., Kulyna, S., Yatskiv, N., & Kulyna, H. (2020). Protected Distributed Data Storage Based on Residue Number System and Cloud Services. *10th International Conference on Advanced Computer Information Technologies (ACIT)*, 796–799. https://doi.org/10.1109/acit49673.2020.9208849.

6. Yatskiv, V., Kulyna, S., Bykovyy, P., Maksymyuk, T., & Sachenko, A. (2020). Method of reliable data storage based on redundant residue number system. *IEEE 5th International Symposium on Smart and Wireless Systems within the*

Conferences on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS-SWS), 1-4. https://doi.org/10.1109/idaacs-sws50031.2020.9297052.

Матеріали міжнародних наукових та науково-практичних конференцій України:

7. Кулина, С.В. (2018). Виявлення помилок на основі коригуючих кодів системи залишкових класів. *Фізико-технологічні проблеми передавання, оброблення та зберігання інформації в інфокомунікаційних системах: Матеріали VII Міжнародної науково-практичної конференції,* 126-127.

8. Кулина, С. В. (2019). Виправлення помилок при передачі даних на основі обчислення проекцій числа. *Прикладні науково-технічні дослідження: матеріали III міжнар. наук.-практ. конф., 3-5 квіт. Академія технічних наук України*, Івано-Франківськ: Симфонія форте, 36.

9. Кулина, С. В., Кондратюк, В. М. (2021). Дослідження систем захищеного зберігання даних. *Збірник матеріалів проблемно-наукової міжгалузевої конференції «Автоматизація та комп'ютерно інтегровані технології» (АКІТ - 2021)*, Тернопіль, 157–159.

10. Kulyna, S. (2021). Comparative analysis of the residue number system inverse transform methods for secure data storage. *Proceedings of VIII-th International Scientific and Technical Conferenc «Information protection and information systems security»*, November 11–12, Lviv, Ukraine, 15-16.

11. Кулина, С. В. (2022). Виявлення та виправлення помилок у захищених системах зберігання даних методом обчислення проекції числа. *Збірник матеріалів науково-практичної конференції молодих вчених, аспірантів та студентів «Кібербезпека та комп'ютерно-інтегровані технології»* (КБКІТ-2022), Тернопіль, 13–16.

12. Кулина, С. В. (2022). Виявлення та виправлення помилок у захищених системах зберігання даних методом обчислення синдрому. *Збірник*

матеріалів проблемної наукової міжгалузевої конференції «Автоматизація та комп'ютерно-інтегровані технології» (АКІТ-2022). Тернопіль, 67–69.

13. Кулина, С. В. (2023). Метод захищеного зберігання даних на основі надлишкової системи залишкових класів та хеш функції. *Захист інформації і безпека інформаційних систем: матеріали IX Міжнар. наук.-техн. конф., 25-26 травня 2023 р*. Львів : Видавництво Львівської політехніки, 71–72.

14. Кулина, С. В. (2023). Використання систем розподіленого захищеного зберігання даних для захисту конфіденційної інформації. *Економічний і соціальний розвиток України в XXI столітті: національна візія та виклики глобалізації: матеріали XX Міжнародної науково-практичної конференції молодих вчених*. Тернопіль, 719–722.

15. Кулина, С. В. (2023). Компоненти системи розподіленого захищеного зберігання даних на основі надлишкової системи залишкових класів. *Збірник матеріалів проблемної наукової міжгалузевої конференції «Автоматизація та комп'ютерно-інтегровані технології» (АКІТ-2023).* Тернопіль, 176–178.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 17
ВСТУП…… 18
РОЗДІЛ 1. СУЧАСНИЙ СТАН ТА ШЛЯХИ ЗАХИСТУ ІНФОРМАЦІЇ У СИСТЕМАХ ЗБЕРІГАННЯ ДАНИХ …… 24
1.1. Атаки на дані та типи резервного копіювання…… 24
1.1.1. Типи кібератак…….. 24
1.1.2. Аналіз систем резервного копіювання…….. 27
1.2. Програмне забезпечення для шифрування файлів та дисків…… 30
1.2.1. Аналіз програмного забезпечення для шифрування даних…... 30
1.2.2. Основні програмні рішення для шифрування файлів та папок.. 31
1.3. Класифікація систем зберігання даних…….. 38
1.3.1. Системи зберігання даних в режимі реального часу……... 38
1.3.2. Аналіз існуючих RAID систем для зберігання даних в режимі реального часу…… 41
1.3.3. Коригуючі коди у системах зберігання інформації……. 45
1.4. Постановка задачі……. 47
Висновки до розділу 1…….. 48
РОЗДІЛ 2. МЕТОДИ ВИЯВЛЕННЯ ТА ВИПРАВЛЕННЯ ПОМИЛОК НА ОСНОВІ НАДЛИШКОВОЇ СИСТЕМИ ЗАЛИШКОВИХ КЛАСІВ.. 49
2.1. Виявлення та виправлення помилок у системах зберігання даних на основі надлишкової системи залишкових класів……..... 49
2.1.1. Пряме та зворотне перетворення системи залишкових класів.. 49
2.1.2. Виправлення помилок методом обчислення синдрому……. 54
2.1.3. Виправлення помилок методом обчислення проекції числа…. 55
2.2. Метод захищеного зберігання даних на основі надлишкової системи залишкових класів та геш-функції……. 58
2.3. Оцінка складності методів зворотного перетворення системи залишкових класів для захищеного зберігання даних…… 62
2.3.1. Ймовірність виявлення помилок у послідовностях символів... 63

2.3.2. Аналіз методів зворотного перетворення системи залишкових класів…… 66
2.3.3. Аналітичні вирази оцінки складностей методів зворотного перетворення системи залишкових класів…….. 69
Висновки до розділу 2…… 72
РОЗДІЛ 3. МЕТОД ШИФРУВАННЯ ДАНИХ НА ОСНОВІ НАДЛИШКОВОЇ СИСТЕМИ ЗАЛИШКОВИХ КЛАСІВ ТА ПСЕВДОВИПАДКОВОЇ ПОСЛІДОВНОСТІ…….. 73
3.1. Розрахунок оптимального набору модулів для реалізації систем захищеного зберігання даних…… 73
3.2. Дослідження криптографічної стійкості шифрування даних в системі залишкових класів…… 82
3.2.1. Дослідження стійкості шифрування даних на основі асимптотичного розподілу простих чисел…….. 82
3.2.2. Оцінки криптографічної стійкості шифрування даних в системі залишкових класів…… 91
3.3. Схема та криптостійкість методу шифрування даних в надлишковій системі залишкових класів……..... 94
3.3.1. Метод шифрування даних в надлишковій системі залишкових класів з використанням псевдовипадкових послідовностей…….. 95
3.3.2. Криптографічна стійкість методу шифрування даних в надлишковій системі залишкових класів……. 96
3.3.3. Оцінка криптографічної стійкості пропонованого методу……. 99
Висновки до розділу 3…… 100
РОЗДІЛ 4. АЛГОРИТМИ ЗАХИШЕНОГО (НАДІЙНОГО) КОДУВАННЯ ДАНИХ НА ОСНОВІ НАДЛИШКОВОЇ СИСТЕМИ ЗАЛИШКОВИХ КЛАСІВ…….. 101
4.1. Шифрування даних на основі надлишкової системи залишкових класів та псевдовипадкової послідовності…….. 101
4.1.1. Алгоритми переведення початкового файлу в масив блоків….. 101

4.1.2. Алгоритм шифрування залишків на основі М-послідовності..… 103
4.1.3. Алгоритм запису файлів залишків та додаткових параметрів…. 105
4.2. Перевірка цілісності файлів залишків……... 107
4.2.1. Алгоритм зчитування файлів залишків та обчислення геш-функції..…… 107
4.2.2. Алгоритм перевірки цілісності файлів залишків…….. 109
4.2.3. Алгоритм зчитування файлів залишків для зворотного перетворення…… 110
4.3. Алгоритми дешифрування та перевірки цілісності файлу……. 112
4.3.1. Алгоритм дешифрування залишків…….. 112
4.3.2. Алгоритм перевірки цілісності дешифрованого файлу……..….. 116
4.4. Відновлення файлу за методом проекцій……..… 118
4.4.1. Алгоритм відновлення при втраті одного із блоків даних…..… 118
4.4.2. Алгоритм запису файлу відновленого за методом проекцій….. 126
Висновки до розділу 4……. 129
РОЗДІЛ 5. ПРИКЛАДНА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ДЛЯ РОЗПОДІЛЕНОГО ВІДДАЛЕНОГО ЗБЕРІГАННЯ ДАНИХ……..… 131
5.1. Архітектура системи розподіленого захищеного зберігання даних... 131
5.2. Програмна реалізація системи…….. 136
5.2.1. Інтерфейс користувача…….. 136
5.2.2. Режими роботи програмного засобу…… 140
5.3. Реалізація методу відновлення даних із блоків…….. 143
5.4. Дослідження розробленого методу та аналіз результатів…….. 147
Висновки до розділу 5…… 154
ВИСНОВКИ……..…… 155
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ…….…….. 157
ДОДАТКИ…….….. 171

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API – Application Programming Interface

FBE – file-based encryption

OTFE – on-the-fly encryption

FEK – File Encryption Key

FUSE – Filesystem in Userspace

AES – Advanced Encryption Standard

DAS – direct-attached storage

NAS – network attached storage

SAN – storage area network

RAID – Redundant array of independent disks

SSD – solid-state drive

СЗК – система залишкових класів

НСЗК – надлишкова система залишкових класів

КТЗ – Китайська теорема про залишки

СРЗЗД – система розподіленого захищеного зберігання даних

ПЗД – пристрій зберігання даних

ПВП – псевдовипадкова послідовність

ВСТУП

Обґрунтування вибору теми дослідження. Системи зберігання даних є одним із найбільш важливих компонентів при побудові та використанні інформаційних систем. Широке впровадження інформаційних систем, побудованих зокрема на основі технологій Інтернет-речей (IoT), в різних галузях зумовило стрімке зростання обсягів даних, які потребують безпечного та надійного зберігання. IoT є основою для створення «розумних» систем охорони здоров'я, ефективних систем керування транспортом, енергетикою, «розумного» землеробства, «розумних» будинків та цілих міст. Іншим джерелом, яке буде генерувати великі обсяги даних, є розвиток кіберфізичних систем та перехід до Індустрії 4.0. Як свідчить світова практика, більшість організацій використовують методи розподіленого зберігання та резервного копіювання даних, а також методи їх зберігання в режимі реального часу для зменшення наслідків втрати інформації. Відповідно розробка та впровадження методів підвищення ефективності вказаного класу систем дозволить значно розширити область їх застосування.

Системи зберігання даних відіграють ключову роль у забезпеченні доступності та цілісності інформації. Дані можуть бути збережені на тривалий термін, упорядковані та організовані таким чином, щоб бути доступними для авторизованих осіб у будь-який момент. Це особливо важливо в сучасному світі, де ефективна організація та доступність великих обсягів даних допомагає приймати важливі рішення в реальному часі. Із зростанням кількості кібератак та зловживань даними необхідно забезпечувати їх захист від несанкціонованого доступу. Застосування шифрування та інших технологій захисту інформації допомагають зберегти конфіденційність даних, запобігти їх втраті чи пошкодженню.

З підвищенням швидкодії процесорів та обсягів пам'яті значно розширилась область досліджень та застосування системи залишкових класів,

зокрема методів шифрування та побудови коригуючих кодів, про що свідчить збільшення кількості публікацій в даній області.

Перспективним підходом підвищення захищеності систем зберігання даних є використання надлишкової системи залишкових класів. Методи виправлення помилок на основі коригуючих кодів широко використовуються для перевірки виконання арифметичних операцій та в системах передачі даних.

Вагомий внесок у розвиток теорії і практики застосування системи залишкових класів внесли такі вітчизняні та закордонні вчені: М. М. Касянчук, С. О. Кошман, В. А. Краснобаєв, Я. М. Николайчук, В. В. Яцків, А. Мохан (A. Mohan), А. Омонді (A. Omondi) та інші.

Однак, на даний час недостатньо досліджено використання надлишкової системи залишкових класів при побудові захищених систем зберігання даних.

Таким чином, актуальною є науково-прикладна задача підвищення захищеності та надійності систем зберігання даних, вирішення якої зумовлює необхідність розробки методів та алгоритмів, які б забезпечували задану надійність при мінімальній надлишковості та унеможливлювали доступ до інформації при компрометації окремих пристроїв розподіленої системи зберігання даних.

Зв'язок роботи з науковими програмами, планами, темами. Розробка основних положень дисертаційного дослідження здійснювалась відповідно до тематичних планів і науково-дослідних робіт, програм і договорів, що виконувалися у Західноукраїнському національному університеті:

1) держбюджетних НДР:

– «Теоретичні основи та апаратні засоби підвищення продуктивності роботи безпровідних сенсорних мереж» (№ державної реєстрації 0117U000414);

– з виконання завдань Перспективного плану розвитку наукового напряму «Технічні науки» ЗУНУ (1 етап: «Розробка методів та алгоритмів захищеного зберігання даних», 2 етап: «Розвиток систем підтримки рішень, керованих моделями та даними, в умовах невизначеності», № державної реєстрації 0121U114705);

2) госпдоговірних тем:

– «Розробка алгоритмів надійного розподіленого зберігання даних на основі модулярних коригуючих кодів» (№ державної реєстрації 0118U100457);

– «Методи та алгоритми захищеного зберігання даних на основі кодів системи залишкових класів» (№ державної реєстрації 0121U107651).

Мета і завдання дослідження. Мета дисертаційної роботи – підвищення захищеності та надійності зберігання даних у локальних та мережевих сховищах, шляхом розробки методів і алгоритмів шифрування та забезпечення цілісності даних на основі надлишкової системи залишкових класів.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1) провести аналіз методів резервного копіювання та зберігання даних в режимі реального часу;

2) розробити метод надійного зберігання даних на основі системи залишкових класів з розширеною системою модулів та обчислення геш-функції;

3) провести дослідження часової складності методів зворотного перетворення СЗК на основі Китайської теореми про залишки;

4) дослідити криптографічну стійкість шифрування даних в системі залишкових класів;

5) розробити метод шифрування даних на основі надлишкової системи залишкових класів та псевдовипадкової послідовності;

6) провести оцінку криптографічної стійкості запропонованого методу шифрування даних на основі надлишкової системи залишкових класів та псевдовипадкової послідовності та порівняти з існуючим;

7) розробити алгоритми та структуру системи надійного та захищеного зберігання даних на основі надлишкової системи залишкових класів;

8) розробити прототип системи надійного та захищеного зберігання даних для підтвердження ефективності запропонованих рішень.

Об’єкт дослідження – процеси кодування та шифрування інформації у системах зберігання даних.

Предмет дослідження – методи і алгоритми кодування та шифрування інформації у системах зберігання даних на основі надлишкової системи залишкових класів.

Методи дослідження. При розв'язку задач у дисертаційній роботі використовувались методи: теорії інформації та кодування, теорії чисел, забезпечення цілісності даних, виявлення та виправлення помилок в системі залишкових класів.

Наукова новизна одержаних результатів:

1) вперше розроблено метод надійного зберігання даних на основі коригуючих кодів системи залишкових класів, який, на відміну від відомих, базується на використанні одного перевірочного символу та обчисленні значення геш-функції від файлів залишків, що дозволило зменшити надлишковість на 33% у порівнянні з використанням коригуючих кодів;

2) вперше отримано аналітичні вирази для оцінки криптографічної стійкості шифрування даних в системі залишкових класів із врахуванням мінімальної довжини файлу, що дало змогу встановити оптимальні значення модулів для реалізації захищеного розподіленого зберігання даних;

3) удосконалено метод шифрування даних в системі залишкових класів шляхом циклічного зсуву позицій залишків з використанням псевдовипадкових послідовностей, що забезпечило вищу, в середньому у три рази, криптографічну стійкість при заданій розрядності модулів.

Практичне значення одержаних результатів. Розробленні у роботі алгоритми, а саме: шифрування залишків на основі запропонованого методу циклічного зсуву згідно з М-послідовністю; перевірки цілісності файлів залишків шляхом порівняння геш-значень; відновлення при втраті одного із блоків даних на основі удосконаленого методу проекцій, дозволили побудувати структурні схеми модулів системи захищеного зберігання даних.

Структурні схеми модулів, представлені у дисертаційному дослідженні, реалізовані у відповідному програмному забезпеченні захищеної системи

зберігання даних, що дозволило виявити переваги запропонованих методів над існуючими.

Теоретичні та практичні результати дисертаційної роботи використано і впроваджено:

– у навчальний процес для викладання дисциплін «Дослідження і проектування систем захисту інформації», «Оцінка складності алгоритмів шифрування» та «Методи шифрування в системі залишкових класів» для студентів освітньо-кваліфікаційного рівня «магістр», що навчаються за спеціальністю 125 «Кібербезпека» в Західноукраїнському національному університеті;

– при виконанні науково-дослідних робіт, які виконувались в Західноукраїнському національному університеті;

– при розробці систем і резервного копіювання даних на підприємстві ТОВ «ТЕРВІКНОПЛАСТ».

Особистий внесок здобувача. Наукові положення, які містяться в дисертації, отримані здобувачем особисто. У роботах, опублікованих у співавторстві, здобувачу належать: метод захищеного зберігання даних [1, 6]; розробка захищеної системи зберігання даних на основі системи залишкових класів [5, 9].

Апробація результатів дисертації. Основні теоретичні положення та практичні результати дисертації доповідалися і обговорювались на семінарах та конференціях: наукових семінарах кафедри кібербезпеки Західноукраїнського національного університету (2019-2023 рр.), Міжнародних науково-практичних конференціях «Фізико-технологічні проблеми передавання, оброблення та зберігання інформації в інфокомунікаційних системах» (Чернівці, Україна, 2018 р.), «Прикладні науково-технічні дослідження» (Івано-Франківськ, Україна, 2019 р.), «Автоматизація та комп’ютерно-інтегровані технології» (Тернопіль, Україна, 2021-2023 рр.), «Кібербезпека та комп’ютерно-інтегровані технології» (Тернопіль, Україна, 2022 рр.), «Економічний і соціальний розвиток України в XXI столітті: національна візія та виклики глобалізації» (Тернопіль,

Україна, 2023 р.), Міжнародній науково-технічній конференції «Захист інформації і безпека інформаційних систем» (Львів, Україна, 2023 р.), Міжнародних конференціях 10th International Conference on Advanced Computer Information Technologies ACIT' 2020 (Deggendorf, Germany, 2020), IEEE 5th International Symposium on Smart and Wireless Systems within the Conferences on Intelligent Data Acquisition and Advanced Computing Systems IDAACS-SWS (Dortmund, Germany, 2020), VIII-th International Scientific and Technical Conference «Information protection and information systems security» (Lviv, Ukraine, 2021).

Публікації. За матеріалами дисертації опубліковано 15 наукових праць, з них 4 статі у наукових фахових виданнях України та 11 публікацій у матеріалах та збірниках доповідей наукових конференцій, з яких дві індексуються у наукометричних базах даних Scopus та Web of Science.

Структура та обсяг роботи. Дисертаційна робота складається із вступу, п’яти розділів, висновків, списку використаних джерел і додатків. Загальний обсяг основного тексту складає 170 сторінок. Робота містить 55 рисунків, 41 таблицю, список використаних джерел із 124 найменувань на 14 сторінках, додатки на 32 сторінках.

РОЗДІЛ 1. СУЧАСНИЙ СТАН ТА ШЛЯХИ ЗАХИСТУ ІНФОРМАЦІЇ У СИСТЕМАХ ЗБЕРІГАННЯ ДАНИХ

1.1. Атаки на дані та типи резервного копіювання

1.1.1. Типи кібератак

Основною метою кібербезпеки є гарантування безпеки даних користувачів. Зазвичай кібератаки спрямовані на пошкодження важливих даних у корпоративній і персональній мережах або отримання доступу до них. Такі атаки можуть здійснюватися як окремими особами, так і цілими організаціями. В більшості випадків метою кіберзлочину є доступ до платіжних даних або інформації, за яку можна отримати гроші [1].

Однак, щоб виконати вимоги кібербезпеки, розробники часто замість того, щоб проектувати стійкі мережі, здійснюють додавання засобів контролю безпеки до існуючої архітектури системи. Контроль безпеки необхідний, проте він не усуває першопричину вразливості, якою є невідповідність знань користувачів правилам інформаційної безпеки [2]. Зловмисникам відома така вразливість, тому більшість методів несанкціонованого доступу включають обман користувача, щоб він відвідав скомпрометований веб-сайт, перейшов за шкідливим посиланням, завантажив зловмисне програмне забезпечення або вставив скомпрометований USB-накопичувач [3, 4].

Стратегії кібербезпеки зосереджуються на навчанні користувачів розпізнавати та уникати спроб зловмисників ввести в оману, проте користувачі з часом все одно можуть стати жертвами обману. Крім того, зловмисники можуть отримати доступ шляхом підкупу працівників цільової фірми [5].

До найпоширеніших типів кібератак можна віднести наступні [6, 7]:

– шкідливе програмне забезпечення;
– фішинг;
– атаки типу «Відмова в обслуговуванні» (DDoS);
– SQL-ін’єкції;

– міжсайтові сценарії;
– бот-мережі;
– програми шифрувальники.

Шкідливе програмне забезпечення або програми зазвичай замасковані під безпечне для користувача вкладення в електронну пошту, різноманітні фальшиві кнопки, натискання яких дає змогу обходити системи мережевої безпеки та інсталювати інше зловмисне програмне забезпечення або просто повністю вивести з ладу операційну систему. Прикладами шкідливого програмного забезпечення можуть слугувати троянські програми, шпигунське програмне забезпечення, хробаки, віруси та рекламне програмне забезпечення [8, 9].

Поширені стратегії боротьби з програмами-вимагачами наголошують на оборонних заходах, хоча досвід показує, що не можливо перешкодити добре забезпеченому противнику, який має намір проникнути в систему цілі [10]. А посібник із найкращих практик програм-вимагачів від Агентства з кібербезпеки та безпеки інфраструктури Міністерства національної безпеки починається з рекомендації «зберігати зашифровані резервні копії даних у автономному режимі та регулярно тестувати свої резервні копії» [11].

Отже, поточні стратегії захисту інформації спрямовані на підвищення стійкості [12]. Вони вимагають підтримки резервного копіювання системи, що може запобігти значній втраті даних, проте потребує додаткових ресурсів для обробки та зберігання інформації. Поширені системи зберігання інформації зазвичай використовують централізовану структуру (рисунок 1.1).

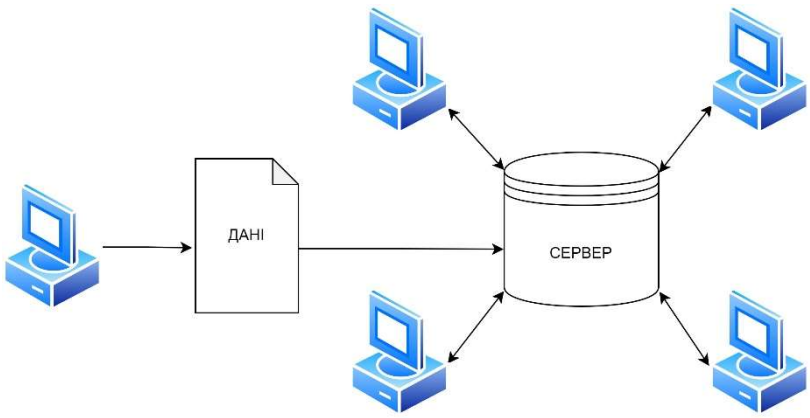

Рисунок 1.1 – Передача інформації в централізованій системі [13]

При передачі даних у централізованій системі за обробку інформації відповідає сервер. Будь-який запит у мережі проходить через нього і у випадку кібератак саме він, в першу чергу, піддається загрозам. Атака може бути здійснена як із зовнішньої мережі, так із робочих станцій, підключених безпосередньо. У випадку пошкодження сервера інформація зазвичай повністю втрачається.

Одним із ефективних шляхів боротьби з кіберзагрозами є налаштування мереж таким чином, щоб збільшити їх стійкість до атак. Зокрема перейти від захисту пристроїв, таких як сервери та робочі станції, до забезпечення можливості негайного відновлення даних на цих пристроях [14].

Таким рішенням можуть слугувати децентралізовані системи зберігання файлів [15, 16], які замість зберігання файлів і даних на центральному сервері розбивають, гешують та шифрують файли, зберігаючи частини даних в різних місцях (рисунок 1.2).

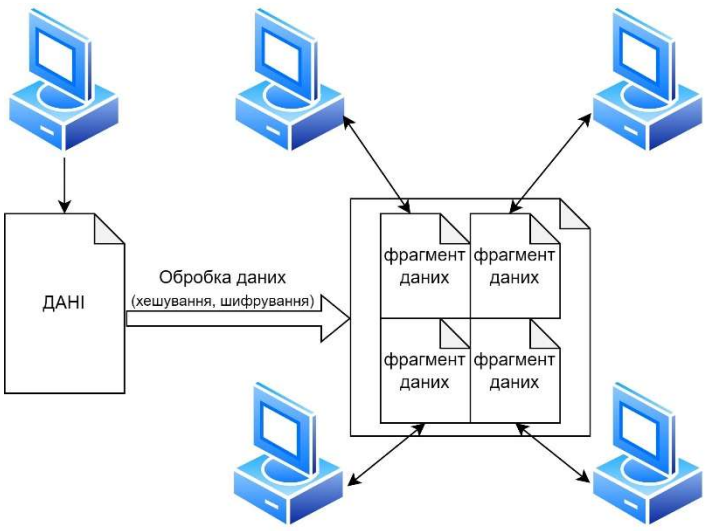

Рисунок 1.2 – Передача інформації в децентралізованій системі [13]

У випадку кібератаки на децентралізовану систему користувачі можуть відмовитися від скомпрометованих пристроїв, використовуючи нові машини, щоб повторно зібрати свої файли та відновити роботу у звичному режимі, а шифрування запобігає використанню їх для вимагання чи інших цілей.

Децентралізовані системи є поширеними рішеннями для керування великими та складними системами, такими як енергетичні підприємства, робототехніка, водопровідні мережі, бездротові сенсорні мережі, керування трафіком тощо [17, 18].

У [19] досліджуються проблеми децентралізованого керування об’єктами та пропонується три підходи до керування великомасштабними взаємопов’язаними системами. Для гарантування роботи стабільної великомасштабної енергетичної системи використовуються децентралізовані контролери із зворотним зв’язком [20].

У [21] досліджується використання автономної децентралізованої технології як один із підходів до забезпечення безпеки в системі управління, що інтегрує функції інформаційного обслуговування. Проте, збереження резервних копій системи і надалі вважається основним засобом пом’якшення наслідків кібератак та підвищення стійкості систем до атак [13].

При резервному копіюванні системи забезпечується певна стійкість, проте може відбутися втрата даних у разі повернення до останнього відомого справного стану даних [22].

Системні резервні копії забезпечують можливість відновлення системи, але їх недостатньо, тому NIST Privacy Framework [23] зазначає, що успіх місії та бізнес-функції залежать від «конфіденційності, цілісності та доступності інформації, яка обробляється, зберігається та передається», тобто захист даних є важливішим, ніж захист пристроїв або навчання користувачів захисту пристроїв.

1.1.2. Аналіз систем резервного копіювання

Як уже зазначалося раніше, резервне копіювання даних дозволяє значно скоротити простій системи у разі втрати даних, кібератак або технічних неполадок, а використання розподіленої інфраструктури для зберігання усуває ризик недоступності інформації після її пошкодження.

Згідно словника Storage Networking Industry Association резервна копія (англ. backup copy) – це копія даних, що зберігається на енергонезалежному носії

інформації з ціллю їх відновлення у випадку втрати або пошкодження оригіналу [24].

Всі системи резервного копіювання даних згідно методу копіювання діляться на три типи копіювання: пофайлове, блочне або на рівні додатків. Загальна класифікація систем резервного копіювання за методом підключення, методом копіювання, за регулярністю копіювання та повнотою копіювання наведена на рисунку 1.3 [25-29].

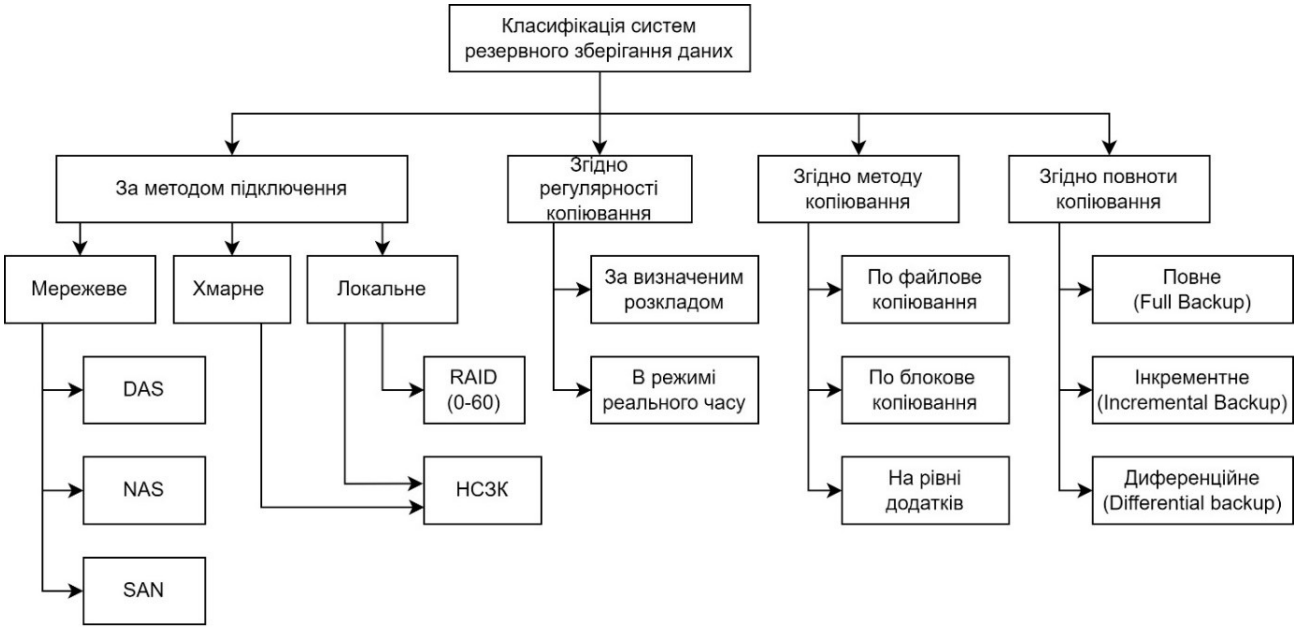

Рисунок 1.3 – Класифікація систем резервного зберігання даних

Операції резервного копіювання на файловому рівні використовують файлову систему. При цьому відносно простим завданням є відновлення деяких конкретних файлів, а операції резервного копіювання тривають довше, що призводить до додаткового завантаження операційної системи, а також зростає вартість резервного копіювання та відновлення втрачених даних [25].

Система блочного резервного копіювання працює безпосередньо із носієм, ігноруючи файлову структуру і зберігаючи усі матеріали повністю – операційну систему, робочі дані, налаштування та інше. Перевагою виконання даного виду резервного копіювання є висока швидкість. Однак зазвичай при виконанні

операцій копіювання потрібно призупинити роботу додатків, щоб копія була цілісною.

Деякі системи виявляють невикористовувані блоки і виключають їх копіювання. При використанні блокового резервного копіювання досить складним є відновлення приватних файлів [30].

Резервне копіювання може здійснюватися на рівні додатків [31]. Операції копіювання і відновлення виконуються за допомогою використання спеціально передбаченого в резервованому додатку програмного інтерфейсу API. Резервна копія являє собою набір файлів і можливо інших об’єктів, визначених самим додатком, які в сукупності є відображенням стану додатку у певний момент часу. При такому способі резервного копіювання може мати місце проблема сумісності між різними версіями додатків і систем резервного копіювання, що реалізують відповідний інтерфейс.

Сучасні системи резервного копіювання реалізуються як програмно, так і апаратно, а також в поєднанні програмних і апаратних компонентів.

Використання тільки програмних засобів значно дешевше і є більш універсальним. Вони виконують свої завдання незалежно від того, де і як розташоване приміщення для серверів, або від того, як здійснюється доступ до корпоративних додатків з робочих станцій співробітників. Програмні рішення слабо залежать від прийнятої архітектури зберігання і захисту даних.

Одним із найбільш відомих на сьогодні рішень є використання хмарних сховищ [32]. Сплачуючи лише за використані ресурси, компанії швидко роблять бекап, реплікацію і відновлення даних з хмари. Користувач сам встановлює частоту створення бекапу і визначає термін зберігання даних в залежності від своїх потреб [26]. Проте варто пам’ятати, що хмара – це чужий ресурс і не належить користувачу, а тому інформація в ній може бути втрачена у будь-який момент часу.

На сьогодні, окрім повного, застосовуються ще два типи резервного копіювання даних: диференційне, коли кожен змінений файл копіюється заново, замінюючи раніше скопійовану версію, та інкрементне резервне копіювання,

коли копіюються тільки змінені файли. Перший тип прискорює відновлення даних, а другий – прискорює процес копіювання, однак ускладнює процес відновлення, збільшуючи його часомісткість [33].

Кожен із згаданих вище шляхів захисту даних від втрат має свої переваги та недоліки, проте об’єднує їх одне спільне, а саме, наявність надлишкової інформації [34]. Надлишковість при створенні копії існуючої інформації дорівнює 100%, тобто інформація дублюється для зберігання.

Іншим важливим недоліком є те, що інформація не доступна в режимі реального часу, тобто повинен пройти певний час для заміни пошкодженого устаткування та відновлення інформації.

1.2. Програмне забезпечення для шифрування файлів та дисків

1.2.1. Аналіз програмного забезпечення для шифрування даних

Все більшого поширення набуває спеціальне програмне забезпечення, яке дозволяє захистити дані від сторонніх осіб або приховати вміст певних файлів, папок або навіть цілих розділів. Таке програмне забезпечення поділяють на програмне забезпечення для шифрування на рівні файлової системи та програмне забезпечення для шифрування дисків (логічних томів) [35].

Шифрування на рівні файлової системи, яке часто називають шифруванням на основі файлів (FBE), або шифруванням файлів/каталогів, є формою шифрування диска, де окремі файли чи каталоги шифруються самою файловою системою.

Шифрування на рівні файлової системи не шифрує метадані файлової системи, такі як структура каталогів, імена файлів, розміри або мітки часу модифікації. Це може бути проблемою у випадку, якщо метадані повинні бути конфіденційними. Іншими словами, якщо файли зберігаються з іменами, які можна ідентифікувати, будь-хто, хто має доступ до фізичного диску, може знати, які файли зберігаються на диску, але не їхній вміст [36].

У свою чергу, програмне забезпечення для шифрування дисків – це програмне забезпечення для захисту комп’ютера, яке захищає конфіденційність даних, що зберігаються на жорсткому диску, дискеті чи пристрої USB, за допомогою шифрування диска [27].

Цей тип шифрування даних пасивно захищає їх конфіденційність навіть тоді, коли ОС неактивна, наприклад, під час зчитування даних безпосередньо з апаратного пристрою чи іншою ОС. Крім того, такі технології позбавляють від необхідності видаляти дані в кінці життєвого циклу накопичувача, перетворюючи їх на нечитабельний код, який неавторизовані особи не можуть легко розшифрувати.

Шифрування дисків (логічних томів) називають прозорим шифруванням, шифруванням на льоту OTFE або шифруванням у реальному часі, оскільки дані автоматично шифруються або розшифровуються під час завантаження чи збереження [37].

Завдяки прозорому шифруванню файли стають доступними одразу після введення ключа чи пароля, а весь том зазвичай монтується так, ніби це фізичний диск, що робить файли доступними. Жодні дані, що зберігаються на зашифрованому томі, не можна прочитати (розшифрувати) без використання правильного паролю або відповідних ключів шифрування. У будь-який інший час без авторизації користувача вся файлова система, включаючи імена файлів, папки, вміст файлів та інші метадані в логічному розділі, шифрується.

В додатку А наведено основні технічні характеристики поширених на даний час програмних рішень для шифрування інформації на пристроях обробки інформації.

1.2.2. Основні програмні рішення для шифрування файлів та папок

Такі програми як BitLocker [38], FileVault [39], VeraCrypt [40], TrueCrypt [41], dm-crypt/LUKS [42] та інші, які є представниками програм для шифрування дисків, спрацьовують тільки при вході в систему або запуску

робочої машини. Така реалізація захисту інформації є ефективною тоді, коли є тільки один користувач системи.

Програма Tomb для ОС Linux дозволяє створити сховище за таким самим принципом і працювати з ним, проте в кінці користувач має «закрити» своє сховище [43].

Варто зазначити, що шифрування диска не завжди ефективніше, ніж шифрування файлів, тому іноді застосовують поєднання обох методів для забезпечення більш безпечної реалізації порівняно з шифруванням файлів.

Оскільки шифрування диска зазвичай використовує той самий ключ для шифрування всього диска, усі дані можна розшифрувати під час роботи системи, тому деякі рішення для шифрування диска використовують декілька ключів для шифрування різних томів.

Окрім цього варто враховувати те, що коли зловмисник отримує доступ до комп'ютера під час його роботи, він отримує доступ до всіх файлів. Звичайне шифрування файлів і папок дозволяє використовувати різні ключі для різних частин диска. Завдяки цьому зловмисник не зможе отримати доступ до повторно зашифрованих файлів і папок [44].

На пристроях, до яких звертаються більше ніж один користувач, зазвичай використовують шифрування FBE для захисту як від зловмисників, так і від співробітників організації.

Прикладом такого ПЗ є EFS – утиліта для ОС Windows, яка при шифруванні використовує випадково згенерований ключ (FEK) для кожного файлу (рисунок 1.4) [45].

FEK – симетричний ключ шифрування, що використовує відкритий ключ користувача, зашифрований файл і алгоритм RSA. Довжина ключа в північноамериканській версії EFS 128 біт, а в міжнародній версії EFS використовується довжина ключа 40 або 56 біт.

Зашифрований таким чином ключ FEK зберігається в альтернативному потоці $EFS файлової системи NTFS.

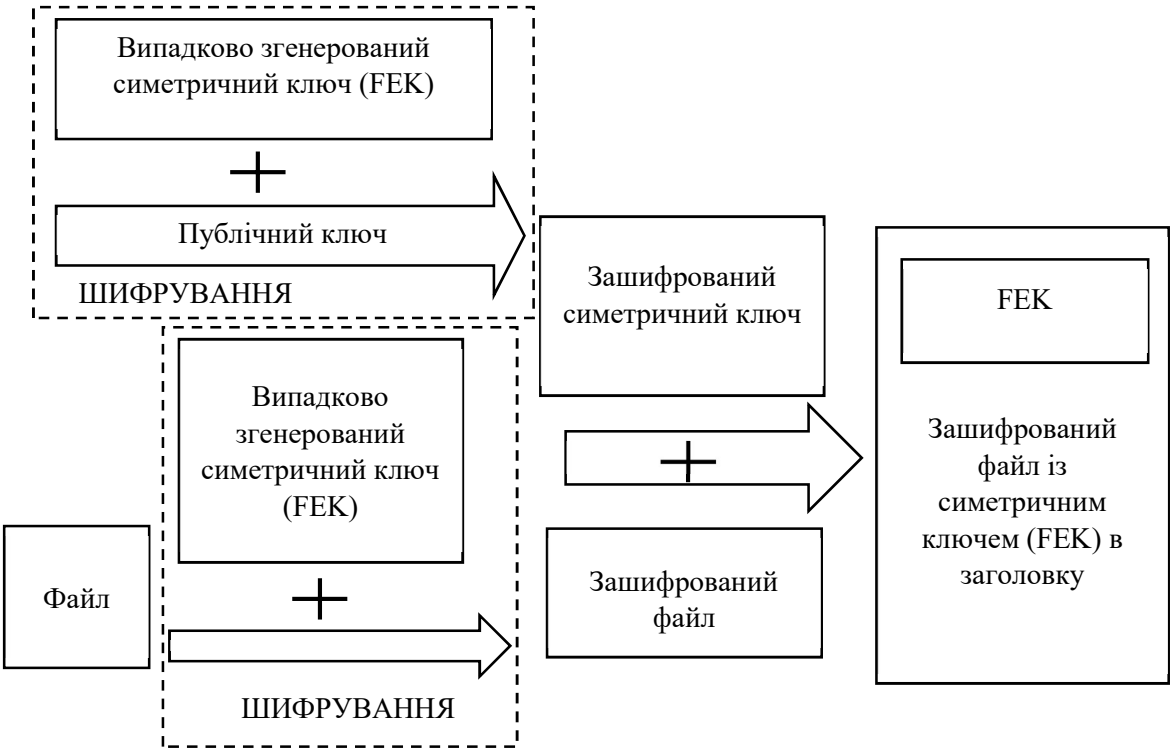

Рисунок 1.4 – Шифрування даних за допомогою FEK [45]

Для розшифрування даних драйвер файлової системи розшифровує FEK, використовуючи закритий ключ користувача, а потім і необхідний файл з допомогою розшифрованого файлового ключа (рисунок 1.5).

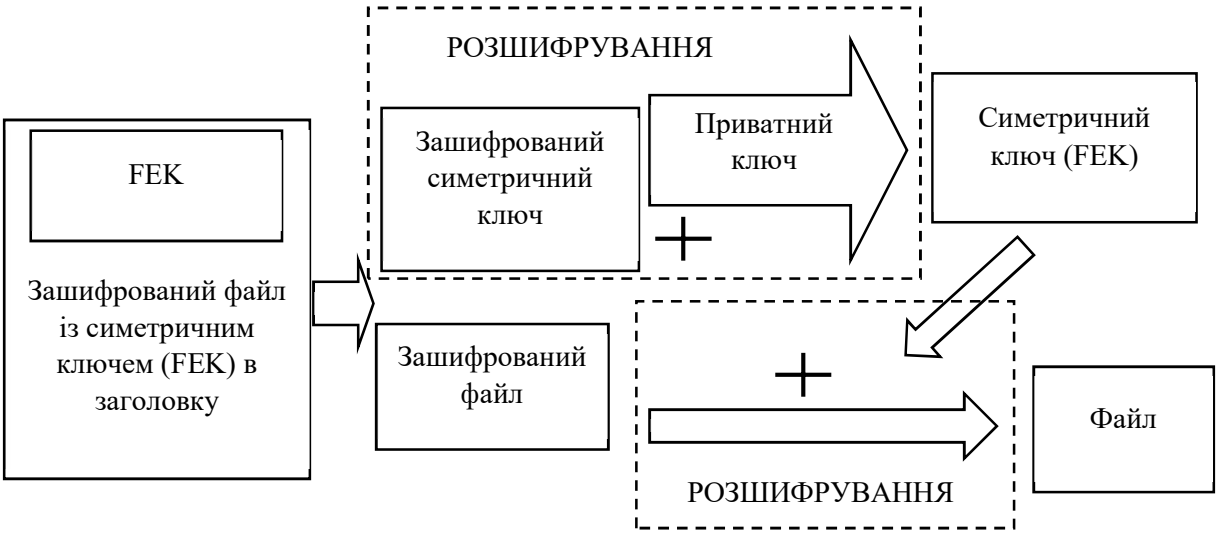

Рисунок 1.5 – Розшифрування даних за допомогою FEK [45]

Оскільки шифрування та розшифрування файлів відбувається за допомогою драйвера файлової системи, який є надбудовою над NTFS, то воно

відбувається прозоро для користувача та додатків. Проте варто зауважити, що EFS не працює із домашніми версіями Windows та не шифрує файли, що передаються мережею. Тому для захисту даних, які передаються, необхідно використовувати інші протоколи захисту (IPSec або WebDAV).

Іншою системою, яку варто згадати, є EncFS – відкрита криптографічна файлова система, що прозоро шифрує файли, використовуючи довільний каталог в якості місця для зберігання зашифрованих файлів [46]. Ця система також набула поширення серед користувачів Linux та Mac OS, оскільки спрямована на захист даних із мінімальними труднощами шляхом використання файлової системи у просторі користувача (FUSE) для монтування зашифрованого каталогу в інший каталог, вказаний користувачем.

FUSE – це програмний інтерфейс для Unix і Unix-подібних комп'ютерних операційних систем, який дозволяє непривілейованим користувачам створювати власні файлові системи без редагування коду ядра [47]. Для шифрування даних використовується бібліотека OpenSSL, що надає алгоритми шифрування AES (16-байтний блоковий шифр з довжиною ключа 128-256 біт) і Blowfish (8-байтний блоковий шифр з довжиною ключа 128-256 біт). Значною перевагою цієї програми є те, що імена файлів шифруються і потім кодуються в 64-бітове кодування, паралельно позбавляючись символів «.» та «/».

Всі операції читання/запису в EncFS є блоковими. Розмір блоків визначається користувачем під час створення файлової системи та варіюється від 64 до 4096 байт. Маленький розмір блоків зменшує час довільного доступу, але збільшує кількість запитів під час читання/запису великих фрагментів файлів. Великий розмір блоків сприяє збільшенню швидкості обробки даних, але збільшує час довільного доступу. На відміну від реальних файлових систем, великий розмір блоків не призводить до втрати дискового простору (незаповнені до кінця блоки не заповнюються нулями).

Іншим рішенням, яке є популярним серед користувачів Linux та Mac OS X можна назвати CryFS, що є криптографічною файловою системою з відкритим кодом, створеною спеціально для хмарного сховища [48, 49].

Щоб приховати розміри файлів, CryFS розбиває вміст файлу на блоки однакового розміру та шифрує ці блоки окремо. Деревоподібна структура використовується для запам’ятовування того, як блоки з’єднуються разом, щоб сформувати файл (рисунок 1.6).

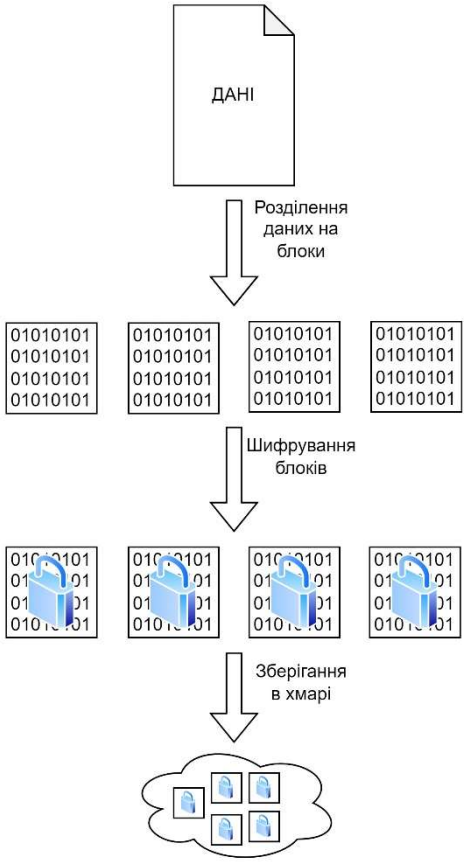

Рисунок 1.6 – Деревоподібна структура CryFS [48]

Ця деревоподібна структура має невеликі накладні витрати і сама також зберігається за допомогою зашифрованих блоків однакового розміру. Щоб приховати метадані файлів і структуру каталогів, їх також представляють за допомогою зашифрованих блоків однакового розміру.

Кожен зашифрований блок зберігається як файл у базовому каталозі з використанням випадкового ідентифікатора в якості імені файлу. Потім базовий каталог можна налаштувати для синхронізації з хмарним постачальником, таким як Dropbox. Зловмисник, який має доступ до основного каталогу, може бачити лише набір блоків зашифрованого тексту однакового розміру і не може бачити вміст, розміри файлів, метадані чи структуру каталогів ваших даних.

Блоки шифруються за допомогою блочного шифру, який обирає користувач. Під час створення файлової системи генерується випадковий ключ шифрування. Цей ключ зберігається у файлі конфігурації, який потім шифрується за допомогою пароля, вибраного користувачем. Оскільки конфігураційний файл зашифрований, його можна зберігати разом із блоками зашифрованого тексту в хмарі, що є типовою практикою.

Більш складним програмним рішенням можна вважати багаторівневі криптографічні файлові системи, яскравим прикладом яких можна вважати eCryptfs для ОС Linux. Її відмінність від усіх попередньо розглянутих програмних рішень полягає в тому, що всі криптографічні мета-дані зберігаються всередині зашифрованого файлу. Це дозволяє переміщувати такі файли через довірені канали, зберігаючи можливість авторизованим особам отримати доступ до вмісту файлів [50].

eCryptfs реалізована як модуль ядра Linux та доповнена різноманітними утилітами для роботи з ключами. Цей модуль шифрує вміст файлів, використовуючи криптографічне API ядра (рисунок 1.7).

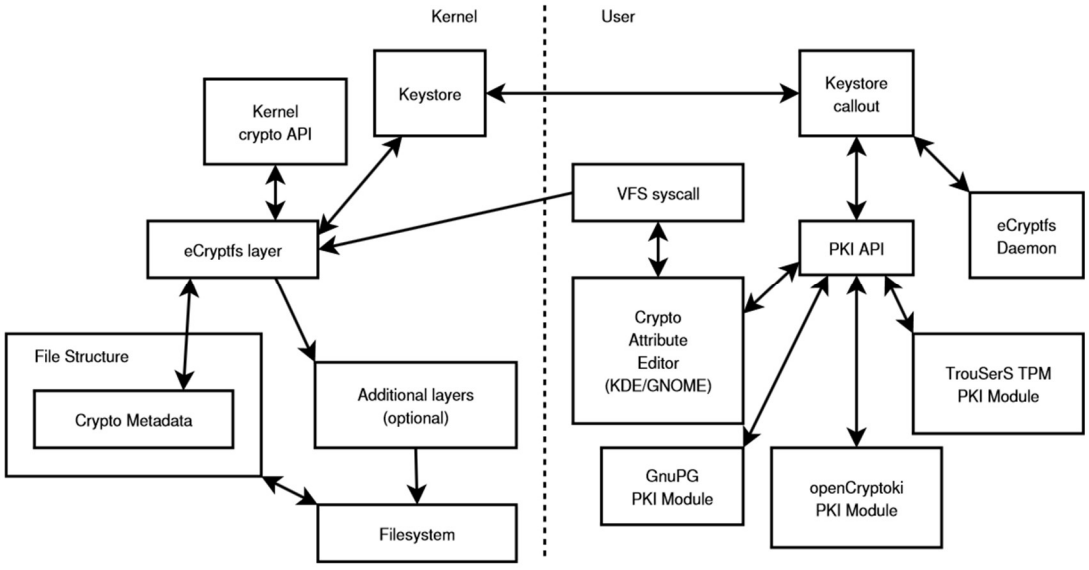

Рисунок 1.7 – Архітектура системи eCryptfs [50]

Модуль зберігання ключів зчитує інформацію із заголовків окремих файлів і відправляє ці дані до програми, яка здійснює шифрування. На основі отриманої

інформації визначаються правила шифрування, згідно з якими приймається рішення щодо подальших дій (наприклад, запропонувати користувачеві ввести пароль або розшифрувати сесійний ключ за допомогою закритого ключа).

В основі eCryptfs лежить формат файлу OpenPGP, описаний в RFC2440 [51]. При цьому, щоб зберегти можливість довільного доступу до даних у файлі, розробники відхилилися від стандарту. Відповідно до формату OpenPGP, операції шифрування та розшифрування повинні проводитися над усім вмістом файлу. Це призводить до того, що не можна прочитати жодного байту з файлу доти, доки він не розшифрований повністю. Щоб обійти цю проблему та не погіршити безпеку системи, eCryptfs розбиває дані на екстенти. За замовчуванням ці фрагменти мають розмір сторінки файлової системи (задається в ядрі, як правило, це 4096 байт). Щоб прочитати дані з одного із фрагментів, потрібно повністю розшифрувати лише цей фрагмент, а щоб записати дані в блок, потрібно шифрувати весь блок.

Також доцільно розглянути комерційні універсальні програмні рішення, які не надають доступ до алгоритмів роботи своїх продуктів, проте забезпечують необхідний рівень захисту даних користувачів.

Одним із таких продуктів є AxCrypt, який має три режими роботи: безкоштовний, преміум та бізнес [52]. Завдяки тому, що програма є платною, вона поширюється на всіх платформах та має мобільний додаток. За функціональністю безкоштовна версія дає можливість тільки переглядати та розкодовувати файли, надані іншими учасниками. У роботі програми використовується алгоритм шифрування AES-128 та резервне копіювання ключів на сервер компанії.

У преміум версії доступно більше послуг, таких як ключ спільного використання, шифрування AES-256, робота з файлами за допомогою мобільного додатку, спрощене відзначення файлів та папок для шифрування, вбудований генератор паролів та підтримка деяких хмарних сховищ.

Бізнес версія дозволяє все вище згадане, а також надає прямий контакт з адміністрацією проєкту. Проте єдиним, на що варто звернути увагу у бізнес версії є майстер-ключ, який дозволяє відновити видалені файли.

Іншим платним продуктом є NordLocker [53]. Він забезпечує, як і більшість програм для шифрування файлів, асинхронне наскрізне шифрування AES-256, а також двофакторну автентифікацію, яку можна використовувати як для шифрування файлів на персональній машині клієнта, так і для збереження на хмарі сервісу в межах оплаченого обсягу сховища.

Окрім цього сервіс гарантує доступ до профілів користувачів через сервіс конфіденційності з нульовим розголошенням (англ. Zero-knowledge proof), який є безпечною альтернативою поточним системам автентифікації [54].

На основі проведених досліджень можна зробити узагальнення, що більшість програмних рішень для захисту файлів та папок працюють з симетричним алгоритмом шифрування AES, а отже саме цей алгоритм забезпечує необхідний рівень захисту даних при зберіганні.

1.3. Класифікація систем зберігання даних

1.3.1. Системи зберігання даних в режимі реального часу

Одним із методів захисту від втрати інформації є використання методів розподіленого (віддаленого) зберігання та резервного копіювання даних. На сьогодні вони використовуються, як при побудові баз даних, так і сховищ даних, як для корпоративних рішень окремих компаній, так і для організації мережних та хмарних сховищ даних [28].

Для зберігання даних в режимі реального часу застосовують наступні типи систем [55]:

1) пряме сховище (DAS);

2) мережний пристрій зберігання даних (NAS);

3) мережа зберігання даних (SAN);

4) хмарне сховище;

5) резервний масив незалежних дисків (RAID).

DAS – це сховище, яке складається тільки з накопичувача даних (жорсткий диск або USB-накопичувач), який безпосередньо під'єднаний до комп'ютера чи робочої станції через контролер шини даних [56]. Іншими словами, DAS не є частиною мережі зберігання даних. Найпростішим прикладом DAS є внутрішній жорсткий диск в ноутбуці або настільному ПК. На практиці пряме сховище DAS згадують найчастіше в контексті використання масивів зберігання, які приєднані безпосередньо до серверів.

Основними перевагами DAS є простота та низька вартість встановлення [57]. Використання мережевих систем зберігання вимагає більшого планування, а також придбання та розгортання мережевого обладнання, такого як маршрутизатори та комутатори, на додаток до відповідних кабелів і з'єднань. Проте такі технології зберігання даних як DAS стикаються з великими труднощами в роботі з великими обсягом даних [58].

NAS – це пристрій, підключений до мережі, який дозволяє авторизованим користувачами зберігати та отримувати дані з централізованого сховища та забезпечує високу швидкість доступу до даних [59].

Перевага NAS полягає в тому, що він спрощує обмін файлами між декількома користувачами, потенційно забезпечуючи більш високу продуктивність, ніж традиційний файловий сервер [60].

Для малих і середніх організацій рішення, які зосереджені на продуктивності зберігання, віддають перевагу DAS або NAS, проте вони мають обмеження порівняно з SAN [61].

SAN – це високошвидкісна мережа пристроїв зберігання даних, які також з'єднують ці пристрої зберігання даних з серверами. Вона забезпечує систему блочного копіювання даних у сховище, до якої можуть отримати доступ програми, що працюють на будь-яких мережевих серверах [62].

Завдяки високошвидкісним з'єднанням SAN часто забезпечує кращу продуктивність, ніж DAS [58]. Крім того, SAN пропонує кілька підключень до серверів даних, що значно підвищує доступність даних, швидкість передачі та

обсяг даних [63]. Крім того, відокремлення сховища від серверів звільняє обчислювальні ресурси на серверах для інших завдань, не пов’язаних зі зберіганням інформації [64].

Хмарне сховище – це вид віддаленого сховища, при застосуванні якого використовується простір на накопичувачах в дата-центрі (Google Drive, iCloud, Dropbox та ін.) [29].

Існує багато переваг використання хмарних сховищ, найбільш вагомою є доступність даних. Доступ до інформації, що зберігається в хмарі, можна отримати в будь-який час з будь-якого місця, якщо є доступ до Інтернету. Іншою перевагою є те, що хмарне зберігання надає організаціям можливість здійснювати віддалене резервне копіювання даних, що знижує витрати, пов’язані з аварійним відновленням.

Зручність при використанні хмарних сховищ приваблює багатьох користувачів, проте занепокоєння щодо безпеки, конфіденційності, надійності та інших факторів якості обслуговування стримує їх впровадження [65].

RAID – технологія віртуального сховища даних, яка об’єднує декілька дисків у логічний том з надлишковістю даних з метою забезпечення відмовостійкості або для поліпшення загальної продуктивності накопичувачів.

RAID дозволяє зберігати однакові дані надлишково збалансованим способом, щоб поліпшити загальну продуктивність. Диски RAID часто використовуються на серверах, але, як правило, не потрібні для персональних комп’ютерів. Набором пристроїв, з яких складається система зберігання, керує особливий RAID-контролер. Він забезпечує зв’язок між дисками, розміщенням даних та дозволяє представляти весь масив як один логічний пристрій зберігання. За рахунок виконання паралельних операцій читання та запису на декількох дисках, такий підхід підвищує швидкість обміну порівняно з одним великим диском [66].

Масиви також можуть забезпечувати надійне зберігання інформації для того, щоб дані не були втрачені у разі виходу з ладу одного із дисків. Залежно

від обраного рівня RAID проводиться або віддзеркалення, або рівномірний розподіл інформації на дисках системи.

1.3.2. Аналіз існуючих RAID систем для зберігання даних в режимі реального часу

Дискові масиви з надмірністю даних, які прийнято називати RAID, були вперше представлені в 1987 році у Каліфорнійському університеті в Берклі. RAID 1 було визначено як дзеркальний дисковий масив, RAID 2 – як масив, у якому застосовується код Хеммінга. Рівні RAID 3, 4, 5 використовують парність для захисту даних, RAID 0 був представлений індустрією як не відмовостійкий дисковий масив. Ця класифікація RAID рівнів була прийнята як стандарт [67].

Виділяють три базові шляхи реалізації RAID системи:

– програмну;
– апаратну на основі шини;
– апаратну на основі зовнішнього контролера.

Основними відмінностями між ними є те, яким чином здійснюється керування побудованою системою.

Програмна реалізація забезпечує виконання всіх операцій за допомогою центрального процесора, що дозволяє зменшити вартість системи. Проте є і недоліки, основними з яких є низька продуктивність та додаткове навантаження на центральний процесор. Не варто також забувати про те, що весь потік даних буде проходити через пристрій, на якому реалізована система. Відтак, будь-яка інша інформація на цьому пристрої буде проходити через ту ж шину даних, що й інформація, яка збережена в системі, а це значно зменшує її доступність.

Апаратний RAID на основі шини – це тип апаратного RAID, який найчастіше використовується для систем нижнього рівня (рисунок 1.8).

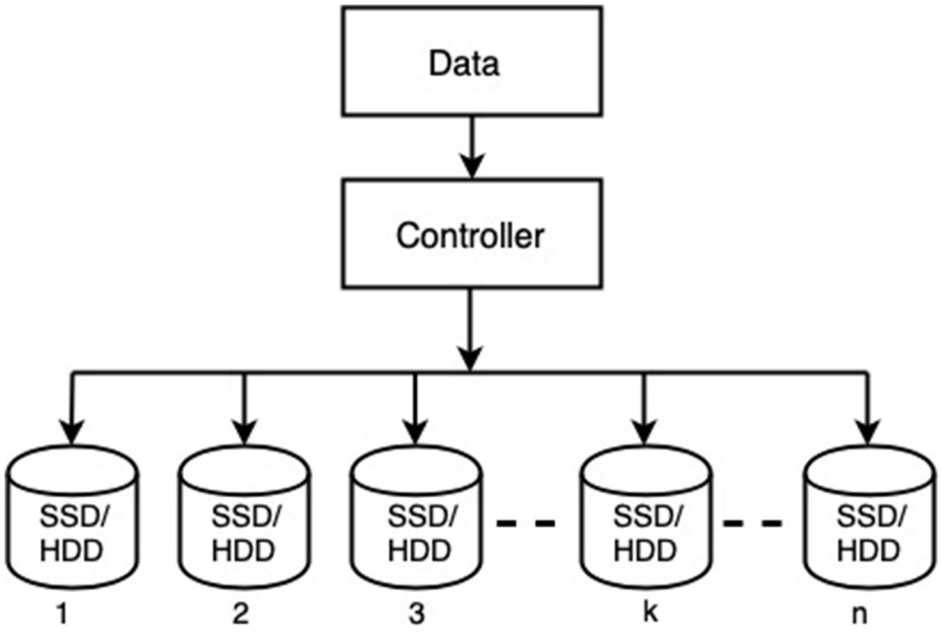

Рисунок 1.8 – Структура RAID системи [67]

На ПК або сервер встановлено спеціалізований адаптер шини хоста, до якого приєднані накопичувачі. RAID на основі шини забезпечує суттєве збільшення продуктивності та можливостей у порівнянні із програмним рішенням RAID. Адаптер шини хоста може мати вбудований процесор для керування деякими або всіма функціями RAID і залишає центральний процесор вільним. Адаптер шини забезпечує поліпшену продуктивність і керованість за допомогою систем на основі програмного забезпечення.

Крім переваг продуктивності, підвищується вартість, проте це легко може бути виправдано поліпшеною продуктивністю і керованістю у порівнянні з системами на основі програмного забезпечення.

Проте система, що базується на шині, все ще підключається за допомогою хост-системи. RAID може вийти з ладу, якщо хост-система перестане функціонувати належним чином. Рівень стійкості до відмов хост-сервера (або його відсутність) є основним недоліком використання апаратного RAID на основі шини. Окрім цього, апаратний RAID на основі шини обмежено максимум RAID 0, RAID 1, RAID 10, RAID 5 і RAID 50. З сучасними дисками об’ємами більше 1 Т схеми захисту подвійного паритету, такі як RAID 4, RAID 5 та RAID 6, стають практичними та необхідними.

Заснований на зовнішньому контролері, RAID має декілька переваг у порівнянні з RAID-системою на основі шини. Зовнішня підсистема RAID стійка до відмови головного сервера, що зводить до мінімуму недоліки системи на основі шини. RAID на основі зовнішнього контролера змонтований в окремий корпус, що забезпечує декілька функцій, які не доступні при використанні внутрішнього контролера, а саме:

– окрема резервна батарея може підтримувати потужність поки дані будуть записані в дискову підсистему;

– можливості гарячої заміни (заміни компонентів без вимкнення основної системи), що підвищує відмовостійкість дискової підсистеми;

– гешування даних для зворотного запису з додатковою безпекою захисту при відключенні електроенергії.

Тривалий час основним рішенням у сфері безпечного зберігання інформації залишається RAID технологія. Проте подібне класичне рішення, поряд з перевагами, має ряд обмежень. До них відносяться:

– відносно низька захищеність (наприклад, RAID 6 контролює вихід з ладу не більше двох дисків);

– необхідність мати в резерві запасні диски;

– неможливість контролю розподілу «parity» в блоках (це може бути важливим для окремих типів дисків);

– складність у підтримці «гетерогенних» дисків.

Одним із показників, що характеризують ефективність RAID масивів, вважається відмовостійкість, тобто допустима кількість дисків, що вийшли з ладу і не вплинули на роботу системи. Порівняння ефективності відомих RAID масивів приведено в таблиці 1.1 [68].

Незважаючи на згадані недоліки, дослідження RAID масивів є об’єктом актуальних наукових досліджень.

У [69-71] пропонуються нові механізми побудови RAID, що працює як звичайна система введення-виведення з оптимізацією навантаження на більший дисковий простір. Запропоновані нові архітектури зберігання даних в RAID

масивах. Розроблені методики оцінки початкових параметрів надійності, які використовуються в моделі надійності та наведено приклади розрахунку середнього часу втрати даних.

Таблиця 1.1

Порівняння ефективності різних типів RAID масивів

Тип RAID масиву	Ефективний обсяг, I	Мінімальна кількість дисків	Відмовостійкість, дисків
RAID 0	$I = S \cdot N$	2	-
RAID 1	$I = S$	2	(N-1)
RAID 2	$I = S \cdot (N - 2)$		1
RAID 3	$I = S \cdot (N - 1)$	3	1
RAID 4	$I = S \cdot (N - 1)$	3	1
RAID 5	$I = S \cdot (N - 1)$	3	1
RAID 6	$I = S \cdot (N - 2)$	4	2
RAID DP	$I = S \cdot (N - 2)$	6	2
RAID 10	$I = S \cdot N/2$	4	від 1 до (N/2)
RAID 50	$I = S \cdot (N - 2)$	6	від 1 до 2
RAID 60	$I = S \cdot (N - 4)$	8	від 2 до 4

де N – кількість дисків в масиві, S – об’єм найменшого диска.

Стаття [72] присвячена питанням побудови систем зберігання даних на твердосплавних накопичувачах (SSD), які широко використовувались у персональних комп’ютерах та центрах обробки даних. Зокрема, розглядалася проблема впливу різних розподілів паритетів на кілька пристроїв, що впливає на надійність масиву SSD RAID. Слід відмітити, що автори не приділяють належної уваги виявленню та виправленню помилок, які виникають внаслідок обмеженого ресурсу використання твердосплавних накопичувачів.

Послуги зберігання даних дають нові можливості для технології розподіленого збереження інформації, а саме хмарні технології. В [73]

представлено систему, яка дозволяє використовувати одночасно декілька різних провайдерів дискового простору та забезпечує доступність і шифрування. Проте, в роботі не досліджено ймовірність втрати зв'язку з одним із провайдерів, що унеможливить доступ до інформації [73]. Використання коригуючих кодів на основі модулярної арифметики для цієї системи надало б можливість мати постійний доступ до даних, незалежно від втрати одного або декількох з них, та забезпечило б шифрування інформації.

Частково виправити вказані недоліки дозволяють коригуючі коди, однак вони також характеризуються високою обчислювальною складністю алгоритмів виправлення помилок [74].

1.3.3. Коригуючі коди у системах зберігання інформації

Ідея коригуючих кодів полягає у доповненні інформаційних послідовностей додатковими символами таким чином, щоб за певними правилами можна було знайти та виправити помилки [75].

Відомо велике число коригуючих кодів, які класифікуються за різними ознаками, проте їх можна розділити на два основні класи: блокові та безперервні. При блочному кодуванні послідовність елементарних повідомлень джерела розбивається на сегменти і кожному з них ставиться у відповідність певна послідовність закодованих символів, яка називається кодовою комбінацією [76].

Довжина сегменту може бути як постійною, так і змінною. Розрізняють рівномірні і нерівномірні блокові коди. Коригуючі коди є, як правило, рівномірними.

Блокові коди бувають роздільні та нероздільні. До роздільних відносяться коди, в яких символи за їх призначенням можуть бути розділені на інформаційні символи, які мають інформацію про повідомлення, та перевірочні. Такі коди позначаються як (*n, k*), де n-довжина коду, *k*-число інформаційних символів. Число комбінацій в коді не перевищує *2 ^ k*.

До нероздільних відносяться коди, символи яких не можна розділити за їх призначенням на інформаційні та перевірочні.

Серед роздільних кодів розрізняють лінійні та нелінійні. До лінійних відносяться коди, в яких порозрядна сума по модулю 2 будь-яких двох кодових слів також є кодовим словом. Лінійний код називається систематичним, якщо перші k символів його кодової комбінації є інформаційними, решта (n-k) символів – перевірочними.

Серед лінійних систематичних кодів найбільш простий код (n, nk), що містить один перевірочний символ, який дорівнює сумі по модулю 2 всіх інформаційних символів. Цей код, званий кодом з перевіркою на парність, дозволяє виявити всі поєднання помилок непарної кратності.

Технології, що засновані на використанні коригуючих кодів, надають значно більше можливостей у порівнянні з класичною RAID системою для вирішення проблем із захистом даних від пошкоджень та при зберіганні даних у розподілених системах [77].

В [74] розроблені ефективні коригуючі коди СЗК, здатні виявляти та виправляти пакети помилок. Для зменшення апаратних витрат у [78] пропонується використання спеціальних модулів СЗК. Проте, оскільки виявлення та виправлення помилок відбувається з використанням алгоритмів зворотного перетворення, яке потребує відновлення позиційного подання повідомлення, процес декодування при зростанні кількості символів у повідомленні та розрядності символів значно ускладнюється.

Коригуючі коди на основі залишкових класів є важливою складовою сучасних систем зберігання та передачі даних [79]. Вони використовуються для виявлення та виправлення помилок, які можуть виникнути під час передачі або зберігання інформації.

Під час передачі мережею або запису на носій інформація може піддаватися впливу різноманітних факторів, таких як шум, перешкоди або пошкодження носія. Коригуючі коди здатні виправляти помилки, що особливо ефективно у системах зберігання, де носій даних піддається деяким пошкодженням або зношуванню з часом. Застосування коригуючих кодів дозволяє відновлювати пошкоджені дані, забезпечуючи високу надійність і

довговічність систем зберігання.

Відповідно, коригуючі коди на основі залишкових класів відіграють важливу роль у забезпеченні надійності та цілісності даних в системах зберігання та передачі, що дозволяє виявляти та виправляти помилки, забезпечуючи точність та доступність інформації [80]. Тому використання коригуючих кодів є важливою практикою в сучасних системах зберігання даних.

1.4. Постановка задачі

З розвитком інформаційного середовища все більшої актуальності набуває питання забезпечення захисту інформації від пошкодження, втрати чи підробки. Тому важливою науковою задачею є підвищення захищеності та надійності зберігання даних у локальних та мережевих сховищах шляхом розробки методів і алгоритмів шифрування та забезпечення цілісності даних на основі надлишкової системи залишкових класів.

Для вирішення сформульованої наукової задачі необхідно розв’язати наступні завдання:

– розробити метод надійного зберігання даних на основі системи залишкових класів з розширеною системою модулів та обчислення геш-функції;

– провести дослідження часової складності методів зворотного перетворення СЗК на основі Китайської теореми про залишки;

– дослідити криптографічну стійкість шифрування даних в системі залишкових класів;

– розробити метод шифрування даних на основі надлишкової системи залишкових класів та псевдовипадкової послідовності;

– провести оцінку криптографічної стійкості запропонованого методу шифрування даних на основі надлишкової системи залишкових класів та псевдовипадкової послідовності та порівняти з існуючим;

– розробити алгоритми та структуру системи надійного та захищеного зберігання даних на основі надлишкової системи залишкових класів;

– розробити прототип системи надійного та захищеного зберігання даних для підтвердження ефективності запропонованих рішень.

Висновки до розділу 1

У розділі обґрунтовано переваги децентралізованих систем зберігання інформації над централізованими.

Розглянуто підходи до побудови систем резервного копіювання, а саме файлове, блочне та копіювання даних на рівні додатків. Проаналізовано операції копіювання та відновлення даних у кожному з підходів.

Проаналізовано програмне забезпечення для шифрування даних, що дозволяє захистити інформацію від несанкціонованого доступу або приховати вміст певних файлів, папок або цілих логічних розділів. Значна частина розглянутого програмного забезпечення функціонує тільки в одній із операційних систем: Windows, Mac OS, Linux, FreeBSD.

Варто зазначити, що більшість спеціального програмного забезпечення використовує блочні алгоритми шифрування, зокрема AES із 128 або 256-бітним ключем, що забезпечує достатній рівень захисту даних при зберіганні. Завдяки використанню ключів шифрування доступ до інформації отримують тільки авторизовані користувачі.

У розділі окрім програмних рішень розглянуто використання апаратно-програмних систем зберігання даних, а саме DAS, NAS, SAN та RAID сховища. Проаналізовано апаратну реалізацію RAID сховища, його переваги та обмеження. Проведено порівняння ефективності різних типів RAID масивів.

На основі проведеного аналізу обґрунтовано переваги використання коригуючих кодів для підвищення надійності зберігання та забезпечення додаткового рівня захисту даних.

РОЗДІЛ 2. МЕТОДИ ВИЯВЛЕННЯ ТА ВИПРАВЛЕННЯ ПОМИЛОК НА ОСНОВІ НАДЛИШКОВОЇ СИСТЕМИ ЗАЛИШКОВИХ КЛАСІВ

2.1. Виявлення та виправлення помилок у системах зберігання даних на основі надлишкової системи залишкових класів

При передачі та зберіганні інформації інколи відбувається її втрата або пошкодження під впливом різних зовнішніх чинників. Тому одним із важливих напрямів наукових досліджень є захист цілісності та конфіденційність даних. Для забезпечення цілісності даних використовують коригуючі коди, які забезпечують виявлення та виправлення помилок у процесі їх передачі та зберігання. На сьогодні розроблено та використовується на практиці значна кількість коригуючих кодів, серед яких важливе місце займають коди на основі СЗК [81, 82].

2.1.1. Пряме та зворотне перетворення системи залишкових класів

В СЗК інформацію представляють у вигляді залишків ($x_1, x_2, \ldots, x_i, \ldots, x_k$) від ділення даних на кожен елемент системи взаємно простих модулів ($p_1, p_2, \ldots, p_i, \ldots, p_k$), де k – кількість модулів системи [83].

Числове значення даних X, подане в позиційній системі числення, повинно лежати в діапазоні $0<X<H$, а значення робочого діапазону H обчислюється за формулою:

$$H = \prod_{i=1}^{k} p_i . \tag{2.1}$$

Пошук залишків відбувається за формулою:

$$x_i = X \left(\mathrm{mod}\ p_i \right). \tag{2.2}$$

Розглянемо систему із 3 модулів: $p_i = \{3, 5, 7\}$. Загальний діапазон даної системи модулів становитиме $P = 105$, а будь-яке значення X можна знайти згідно формули обчислення залишків (2.2).

Для прикладу, нехай $X = 10$, тоді представлення цього числа у СЗК матиме вигляд (1, 0, 3).

СЗК дозволяє забезпечити цілісність інформації при розширенні початкового діапазону обчислення за рахунок введення надлишкових модулів. Завдяки цьому є можливість виявляти помилки з ймовірністю 100% при використанні одного додаткового модуля [84].

При роботі з надлишковими модулями їх називають перевірочними та вводять поняття робочого та загального діапазонів.

Перевірочні символи в коригуючих кодах СЗК обчислюються за формулою [85]:

$$x_{k+r} = X \left(\mathrm{mod}\, p_{k+r} \right), \tag{2.3}$$

де k – кількість інформаційних модулів, а r – кількість перевірочних модулів.

Зазвичай суму k та r позначають n, а для виявлення помилок, окрім робочого (формула 2.1), використовують поняття загального діапазону:

$$P = \prod_{i=1}^{n} p_i \, . \tag{2.4}$$

Для відновлення позиційного представлення числа X використовується формула зворотного перетворення на основі китайської теореми про залишки [86]:

$$X = \left(\sum_{i=1}^{n} x_i \cdot b_i \right) \mathrm{mod} P, \tag{2.5}$$

де b_i – базисні числа СЗК, які обчислюються згідно рівняння:

$$b_i = \frac{P}{p_i} \cdot w_i \equiv 1 (\mathrm{mod}\, p_i), \tag{2.6}$$

де w_i – набір коефіцієнтів, які забезпечують ортогональність перетворень та задовольняють умову: $0 < w_i < p_i$.

При додаванні перевірочних модулів використана раніше система з k = 3 і r = 1 набуде вигляду:

$p_i = \{3,5,7,11\}$ зі значенням робочого діапазону H = 105 та загальним P = 1155.

Масив базисних чисел для зворотного перетворення матиме значення:

$b = \{385, 231, 330, 210\}$, а початкове значення X визначається за наступною формулою:

$$X=(x_0b_0+x_1b_1+x_2b_2+x_3b_3) \bmod P = (x_0 \cdot 385 + x_1 \cdot 231 + x_2 \cdot 330 + \\ +x_3 \cdot 210) \bmod 1155.$$

Візьмемо для прикладу X= 25 =(1, 0, 4, 3).

У таблиці 2.1 представлено можливі значення за модулем p_0, правильні значення модулів p_1-p_3, а також відповідні значення обчисленого X^*.

Таблиця 2.1

Можливі значення за модулем p_0

p_0	p_1	p_2	p_3	X^*
0	0	4	3	795
1	0	4	3	25
2	0	4	3	410

У таблиці 2.2 представлено можливі значення за модулем p_1, правильні значення модулів p_0, p_2, p_3, а також відповідні значення обчисленого X^*.

Таблиця 2.2

Можливі значення за модулем p_1

p_0	$\boldsymbol{p_1}$	p_2	p_3	X^*
1	**0**	4	3	25
1	**1**	4	3	256
1	**2**	4	3	487
1	**3**	4	3	718
1	**4**	4	3	949

У таблиці 2.3 представлено можливі значення за модулем p_2, правильні значення модулів p_0, p_1, p_3, а також відповідні значення обчисленого X^*.

Таблиця 2.3

Можливі значення за модулем p_2

p_0	p_1	$\boldsymbol{p_2}$	p_3	X^*
1	0	**0**	3	1015
1	0	**1**	3	190
1	0	**2**	3	520
1	0	**3**	3	850
1	0	**4**	3	25
1	0	**5**	3	355
1	0	**6**	3	685

У таблиці 2.4 представлено можливі значення за модулем p_3, правильні значення модулів p_0-p_2, а також відповідні значення обчисленого X^*.

Таблиця 2.4

Можливі значення за модулем p_3

p_0	p_1	p_2	$\boldsymbol{p_3}$	X^*
1	0	4	**0**	550
1	0	4	**1**	760
1	0	4	**2**	970
1	0	4	**3**	25
1	0	4	**4**	235
1	0	4	**5**	445
1	0	4	**6**	655
1	0	4	**7**	865
1	0	4	**8**	1075
1	0	4	**9**	130
1	0	4	**10**	340

Для виправлення однієї помилки, на основі таблиць 2.1-2.4, при використанні одного перевірочного модуля, потрібно перебрати всі можливі комбінації залишків. Максимальна кількість залишків D визначається за формулою:

$$D = \sum_{i=1}^{n} p_i \ . \tag{2.7}$$

Згідно формули (2.7), для розглянутого вище прикладу, D = 26, проте, зі збільшенням розрядності модулів зростає складність пошуку та виправлення помилки методом перебору. Тому для виправлення помилок застосовуються інші методи, що спрощують їх виправлення [87].

На даний час основними методами виявлення та виправлення помилок із використанням коригуючих кодів на основі СЗК є виправлення помилок за допомогою обчислення синдрому та проекції числа [88, 89].

2.1.2. Виправлення помилок методом обчислення синдрому

Згідно із алгоритмом виявлення помилок методом обчислення синдрому відновлення позиційного числа відбувається окремо за кожним модулем робочого та перевірочного діапазонів (формула 2.5) [88]:

$$Y = \left(\sum_{i=1}^{k} x_i \cdot b_i \right) \bmod P, \tag{2.8}$$

$$E = \left(\sum_{i=k+1}^{n} x_i \cdot b_i \right) \bmod P_c, \tag{2.9}$$

де $P_c = \prod_{i=k+1}^{n} p_i$ і задовольняється умова $P_c > P$.

Саме обчислення синдрому відбувається за формулою:

$$s = (Y - E) \bmod P_c. \tag{2.10}$$

Приклад. Розглянемо використання системи з трьома інформаційними та двома перевірочними модулями:

– система модулів: $p_i = \{3,5,7,11,13\}$;

– робочий діапазон даної системи модулів буде становити: $P = 105$;

– перевірочний діапазон: $P_c = 143$.

Іншим способом реалізації системи з таким самим робочим діапазоном може бути використання набору модулів з трьома інформаційними і одним перевірочним $p_i = \{3,5,7,107\}$, що дозволить задовільнити умову $P_c > P$. При цьому, як можна бачити на основі формул (2.8-2.10), розрядність перевірочного модуля значно зростає, що ускладнює реалізацію відновлення [90].

Основою виявлення помилок за допомогою обчислення синдрому є створення таблиці, в якій кожному значенню помилки відповідає лише одне значення синдрому. Кількість елементів у такій таблиці буде рівне P_c. Вона дозволяє зменшити кількість необхідних обчислень, проте залишається

проблема відновлення позиційного числа при зростанні кількості символів у повідомленні [88].

Також варто враховувати те, що при роботі з багаторозрядними модулями таблиця значно зростає, що збільшує час пошуку та ускладнює вибірку необхідного значення синдрому.

2.1.3. Виправлення помилок методом обчислення проекції числа

Іншим поширеним способом виявлення та виправлення помилок при пошкоджені цілісності у СЗК є метод обчислення проекцій числа, який базується на визначенні: якщо з числа $X = (x_1, x_2, \dots, x_k)$ вилучити модуль p_i, то саме значення X не зміниться, а проекції цього числа за всіма модулями будуть рівні [91].

Відповідно для виявлення помилки необхідно обчислити позиційне представлення числа X. При цьому, якщо отримане число у діапазоні $[0, H)$, то помилки немає або пошкоджень зазнали залишки за двома і більше модулями.

У випадку, якщо значення $X > H$, то сталася помилка в одному із залишків. Для її виправлення необхідно обчислити проекції числа X за всіма модулями p_i. Якщо значення проекції $X_i < H$, то помилка відбулася за модулем p_i.

Даний метод потребує використання $2 \cdot r$ надлишкових модулів для виправлення r помилок. Також варто зауважити, що при обчисленні проекцій X_i необхідно розраховувати базисні числа b_i для кожної із проекцій, тобто при використанні системи з п’яти модулів необхідно розрахувати та зберегти, окрім базового набору, додатково ще п’ять наборів по п’ять модулів для систем проекцій [92].

Приклад. Взявши той самий набір модулів $p_i = \{3, 5, 7, 11, 13\}$ із значенням робочого діапазону $H = 105$ та загальним $P = 15015$, обчисливши масив базисних чисел для зворотного перетворення $b_i = \{5005, 6006, 10725, 1365, 6930\}$ і значення $X = 25 = \{1, 0, 4, 3, 12\}$, допускаємо, що відбулось спотворення в четвертому залишку: $X_{спот} = \{1, 0, 4, \mathbf{5}^*, 12\}$.

Обчисливши десяткове значення, отримаємо: $X_{спот} = 2755$, що значно

перевищує значення робочого діапазону $H = 105$.

Для виправлення необхідно обчислити b_i^* для кожної із проекцій початкового набору модулів $p_i = \{3, 5, 7, 11, 13\}$:

– $p_i^{*1} = \{5, 7, 11, 13\}$, $P=5005$, $b_i = \{1001, 715, 1365, 1925\}$;
– $p_i^{*2} = \{3, 7, 11, 13\}$, $P=3003$, $b_i = \{2002, 1716, 1365, 924\}$;
– $p_i^{*3} = \{3, 5, 11, 13\}$, $P=2145$, $b_i = \{715, 1716, 1365, 495\}$;
– $p_i^{*4} = \{3, 5, 7, 13\}$, $P=1365$, $b_i = \{910, 546, 1170, 105\}$;
– $p_i^{*5} = \{3, 5, 7, 11\}$, $P=1155$, $b_i = \{385, 231, 330, 210\}$.

Відповідно, підставивши необхідні значення, обчислюємо $X_{поч}$ для кожної проекції (таблиця 2.5).

Таблиця 2.5

Обчислення значень $X_{поч}$ згідно номера проекції

$X_{поч}$	p_0	p_1	p_2	p_3	p_4	j
2755		0	4	5	12	1
2755	1		4	5	12	2
610	1	0		5	12	3
25	1	0	4		12	4
445	1	0	4	5		5

Як бачимо, всі проекції, окрім $j = 4$, більше $H = 105$. Отже, обчисливши значення проекцій, можна виправити одну помилку, маючи два перевірочних модулі.

Для знаходження проекцій числа X_i пропонується використовувати базисні числа b_i, які були розраховані для числа X, в результаті формула зворотного перетворення матиме вигляд:

$$X_j = \left(\sum_{i=1}^{n} x_i \cdot b_i \right) \bmod \frac{P}{p_j}, \qquad (2.11)$$

де j – номер проекції.

Завдяки використанню базисних чисел, обчислених для зворотного перетворення, зменшується в n раз кількість операцій множення, необхідних для відновлення початкового значення числа X, що спрощує алгоритм виправлення помилок на основі коригуючих кодів СЗК [93].

Приклад. Розглянемо приклад використання системи з чотирьох інформаційних та двох перевірочних модулів.

Система модулів: $p_i = \{5, 7, 9, 11, 13, 17\}$.

Загальний діапазон для даної системи модулів буде становити P = 765765.

Робочий діапазон: H = 3465.

Базисні числа: b_i = {306306, 656370, 680680, 556920, 412335, 450450}.

Нехай число X = 73, яке у СЗК з обраними модулями матиме вигляд (3, 3, 1, 7, 8, 5), отримало спотворення в третьому розряді. Тоді спотворене повідомлення буде мати вигляд (3, 3, **0***, 7, 8, 5).

Обчислені згідно формули 2.11 значення $X_{поч}$ наведені в таблиці 2.6.

Таблиця 2.6

Обчислення значень $X_{поч}$ згідно номера проекції

$X_{поч}$	p_1	p_2	p_3	p_4	p_5	p_6	j
85158		3	0	7	8	5	1
85158	3		0	7	8	5	2
73	3	3		7	8	5	3
15543	3	3	0		8	5	4
26253	3	3	0	7		5	5
40113	3	3	0	7	8		6

Як видно з даних таблиці 2.6, при обчисленні значення $X_{поч}$ у всіх випадках, окрім вірного значення $X_{поч} > H$. Це наочно показує, що при втраті або пошкодженні значення за одним із модулів можна відновити втрату, маючи два перевірочних символи.

2.2. Метод захищеного зберігання даних на основі надлишкової системи залишкових класів та геш-функції

На основі наведених методів прямого та зворотного перетворення СЗК, а також методів виправлення помилок, автором запропоновано метод захищеного зберігання даних на основі надлишкової системи залишкових класів (НСЗК) та геш-функції [94-96].

Пропонований метод ґрунтується на використанні розширеного набору модулів СЗК та обчисленні геш-функції, що забезпечує зменшення надлишковості та збільшує швидкість зворотного перетворення і складається з процедур кодування та декодування, які містять наступні кроки.

Кодування:

1. Кодування. Процес кодування полягає у виборі послідовності інформаційних k та перевірочних r модулів, які будуть використовуватися при проектуванні системи.

2. Розділення на блоки. Згідно із робочим діапазоном H, розрахованим на основі модулів системи, визначається розмір блоку даних, який буде зчитуватись із початкового файлу в процесі роботи.

3. Розділення. З кожного блоку даних обчислюються залишки за вибраною на кроці 1 системою модулів p_i та записуються у відповідні файли F_i.

4. Обчислення *hash* (F_i). Для кожного із файлів залишків обчислюється *hash* (F_i) для підтвердження цілісності файлу на етапі відновлення.

5. Обчислення *CRC-32*. Для перевірки коректності та цілісності файлу від *hash* (F_i) обчислюється відповідна контрольна суми *CRC-32*.

6. Зберігання. Для доступності *CRC-32* і *hash* (F_i) зберігаються в розширенні для кожного файлу залишку.

7. Переміщення. В кінці кроку кодування сформовані файли залишків із системною інформацією в назві переміщуються на відповідні носії або у хмарні сховища.

На рисунку 2.1 зображено послідовність кроків роботи з даними та їх результат.

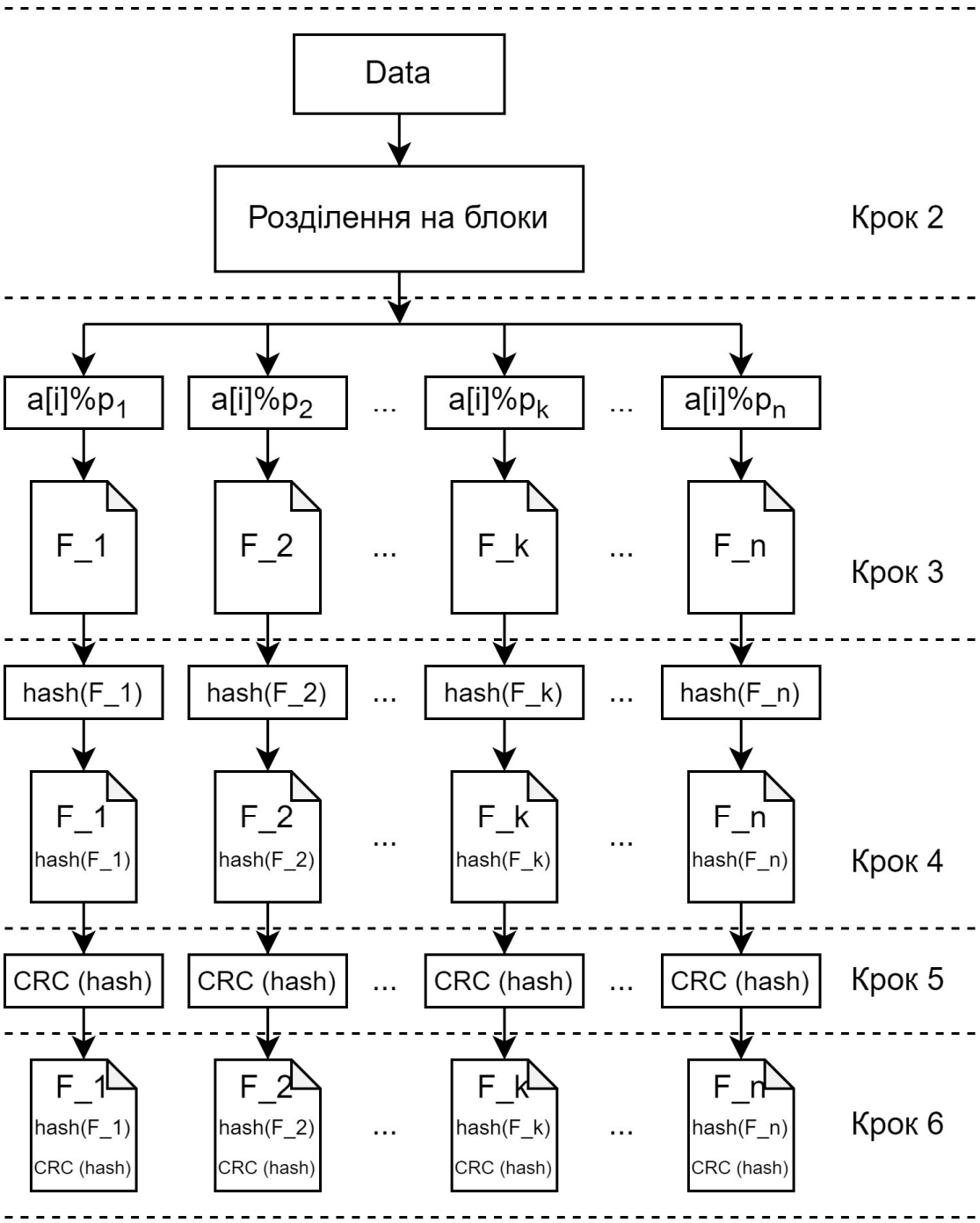

Рисунок 2.1 – Процедура кодування

Процедура декодування містить такі кроки:

1. Системна інформація. Отримання системної інформації про назви файлів залишків, набір залишків, а також шляхи для завантаження та збереження файлів, з якими проводиться робота на даному етапі.

2. Завантаження. Завантаження *k* файлів залишків із відповідних носіїв або хмарних сховищ.

3. Перевірка *CRC-32*. Перевірка контрольної суми *CRC-32*, який зберігається в назві файлу. Даний пункт повторюється для кожного завантаженого файлу залишків.

4. Обчислення *hash* (F_i). Для кожного файлу залишку проводиться обчислення геш-функції.

5. Перевірка *hash* (F_i). Порівняння геш-значення, обчисленого на кроці 4, зі збереженим значенням, обчисленим при кодуванні.

6. Співпадіння *hash*. Якщо у результаті порівняння геш-значення *k* файлів залишків співпадають зі збереженими значеннями, то здійснюється відновлення даних за формулою зворотного перетворення (2.3).

7. Не співпадіння *hash*. Якщо геш-значення одного із *k* файлів залишків не співпадає зі збереженим значеннями, то здійснюється завантаження наступного (*k+1)* файлу залишку.

8. Повторна перевірка *hash* (F_i). Якщо геш-значення (*k+1)*-го файлу залишку співпадає зі збереженим значеннями, то здійснюється відновлення даних за формулою (2.3).

Як зазначалося вище, в загальну кількість модулів, окрім інформаційних *k*, враховується ще і кількість надлишкових *r*, що дозволяє розрахувати надлишковість системи:

$$R = \frac{100 \cdot k}{n}\% . \qquad (2.12)$$

Враховуючи, що в коригуючих кодах СЗК для виявлення та виправлення помилок в залишках по одному модулю необхідно два перевірочні модулі,

запропонований метод із застосуванням геш-функції забезпечує аналогічні параметри з одним перевірочним модулем. При цьому надлишковість значно зменшується.

У таблиці 2.7 на прикладі набору модулів з $k = 3$ представлено порівняння складності та надлишковості виправлення однієї помилки за допомогою запропонованого методу (№3) та методами виявлення та виправлення помилок (№1 та №2), які згадувалися раніше.

Таблиця 2.7

Порівняння методів виявлення та виправлення однієї помилки, при $k = 3$

№з/п	k	r	R	Виявлення	Виправлення	Кількість необхідних операцій для відновлення даних
1.	3	1	33 %	+	+	D
2.	3	2	66 %	+	+	n
3.	3	1	33 %	+	+	1

Як видно з таблиці 2.7, запропонований метод (№3) має значно нижчу надлишковість та завдяки застосуванню геш-функції забезпечує відновлення початкового вмісту файлу при виникненні одиничної помилки за одну операцію відновлення, що значно перевищує можливості методу проекцій (метод №2).

Метод №1, який має таку саму надлишковість, як і пропонований, потребує значно більшої кількості операцій для відновлення пошкоджених даних, а також нездатний виправити дві і більше помилок.

На рисунку 2.2 представлено порівняння надлишковості, необхідної для виправлення однієї, двох і трьох помилок за допомогою запропонованого методу та методу проекцій в залежності від кількості інформаційних модулів.

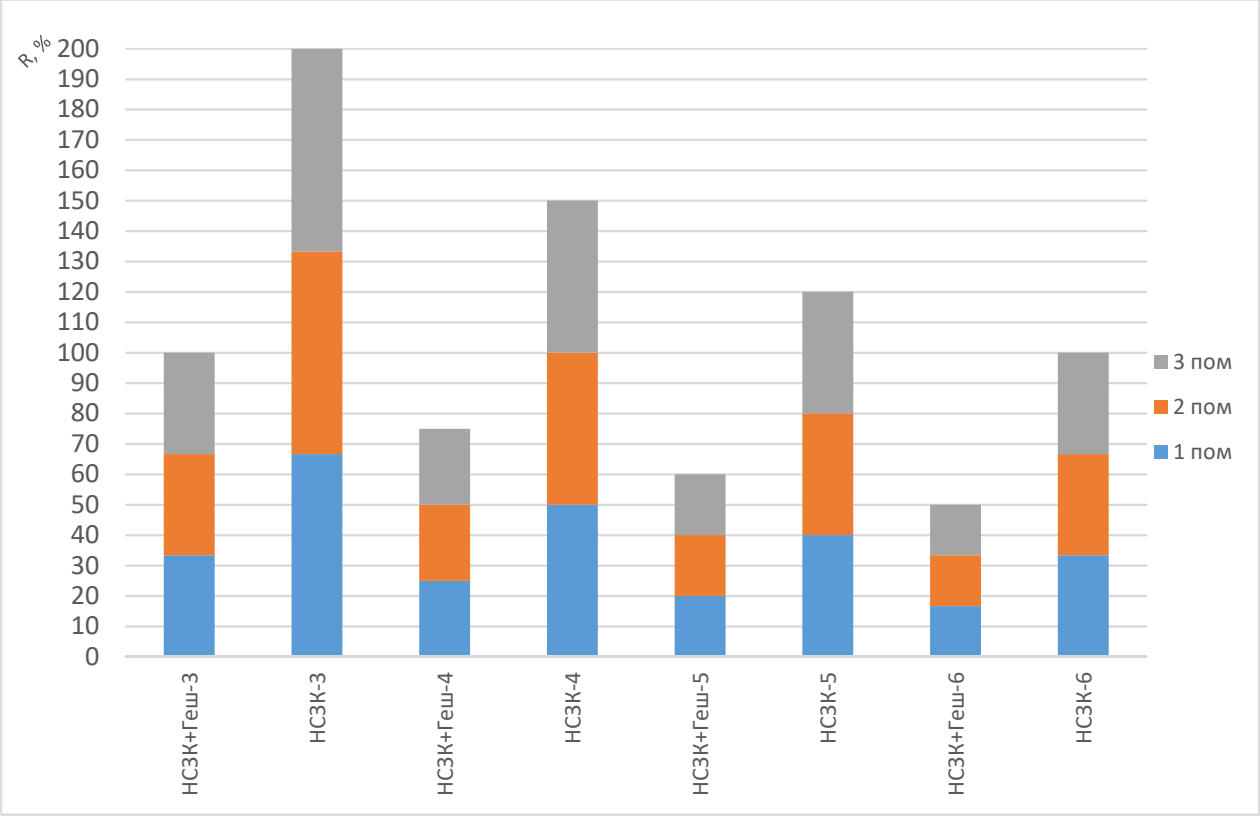

Рисунок 2.2 – Порівняння надлишковості необхідної для виправлення однієї, двох та трьох помилок

Як видно з рисунку 2.2, розроблений метод забезпечує значно вищу ефективність у порівнянні із відомим методом.

2.3. Оцінка складності методів зворотного перетворення системи залишкових класів для захищеного зберігання даних

Умову перевірки на спотворення залишків можна використати для виявлення помилок у двох і більше символах. Для цього були проведені експериментальні дослідження, у результаті яких визначено відсоток виявлення помилок в послідовностях інформаційних символів [93].

2.3.1. Ймовірність виявлення помилок у послідовностях символів

При проведених дослідженнях використовувався набір модулів, достатній для передачі 24-бітного інформаційного повідомлення та перевірочний символ різної розрядності, а саме:

- набір інформаційних модулів p_0=257, p_1=263, p_2=269;
- перевірочний модуль різної розрядності, починаючи з p_3=271.

Оскільки незначне збільшення перевірочного модуля суттєво не впливало на обчислення, то кожне наступне значення перевірочного модуля було більше на один розряд.

У таблиці 2.8 розрахований загальний та робочий діапазони, а також кількість комбінацій з помилками в трьох символах для даних наборів модулів. Оскільки кількість та значення не змінювалося, то незалежно від значення перевірочного модуля робочий діапазон не змінювався і становив 18181979, а кількість комбінацій залишалася рівною 17975296.

Таблиця 2.8

Обчислення кількісного значення невиявлених помилок

p_3	P	Кількість невиявлених помилок	Кількість невиявлених помилок, %
271	4927316309	66327	0, 36899
509	9254627311	35315	0, 19646
1021	18563800559	17606	0, 09795
2039	37073055181	8817	0, 04905
4093	74418840047	4393	0, 02444
8191	148928589989	2195	0, 01221
16273	295875344267	1105	0, 00615
32742	595441630271	549	0, 00305
65521	1191301446059	274	0, 00152

Графічне відображення залежності відсотка невиявлених помилок від перевірочного модуля наведено на рисунку 2.3.

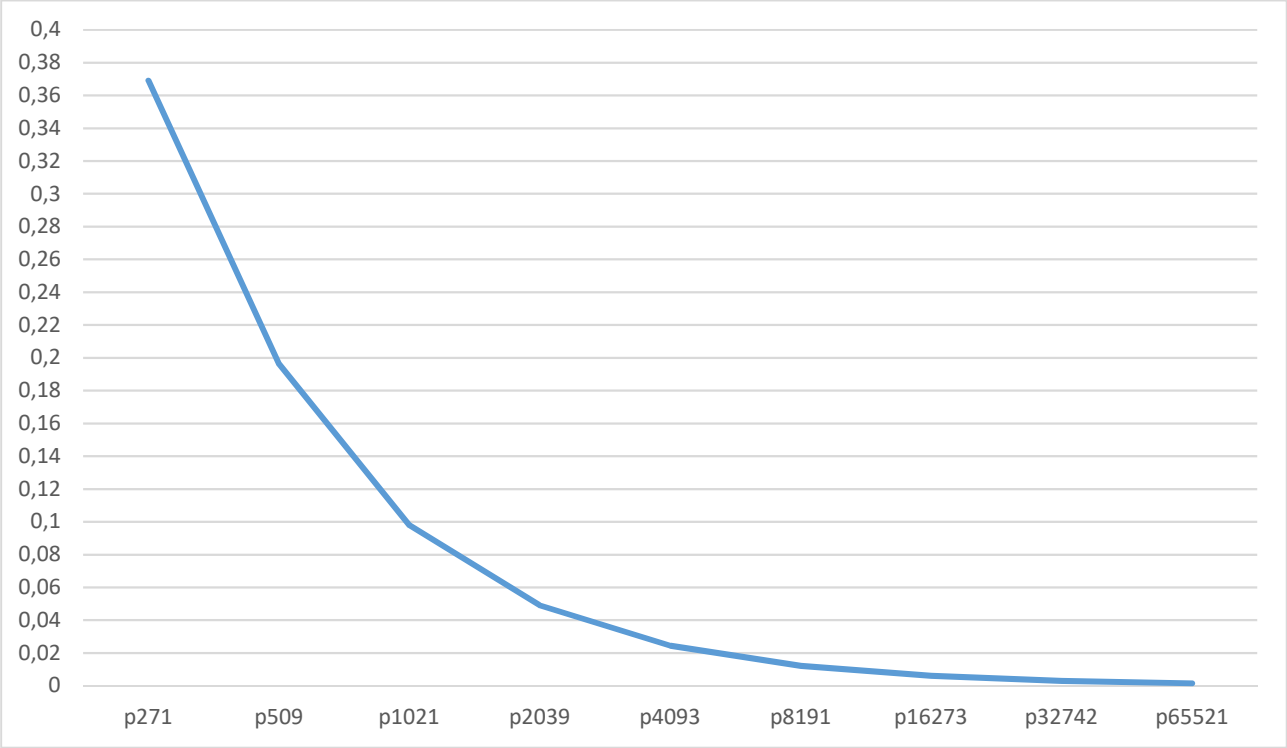

Рисунок 2.3 – Залежність кількості (%) невиявлених помилок від перевірочного модуля

За результатами проведеного експериментального дослідження можна зробити висновок, що при збільшенні значення перевірочного модуля на один розряд кількість невиявлених помилок в трьох символах у заданому діапазоні зменшується в 1,88-2,02 рази. Також з таблиці 2.8 видно, що збільшення перевірочного модуля з 271 до 65521 дозволяє зменшити кількість невиявлених помилок приблизно у 240 разів.

Зазвичай при зростанні розрядності повідомлення зростає ймовірність спотворення більшої кількості символів [97]. Для обчислення цього значення було проведено експериментальне дослідження залежності кількості невиявлених помилок. Результати розрахунків при спотворенні усіх інформаційних символів та при збільшенні їх кількості наведені в таблиці 2.9.

Таблиця 2.9

Обчислення кількісного значення виявлених помилок у всіх модулях, окрім перевірочного

Кількість модулів, (n, k)*	Загальна кількість комбінацій	Кількість невиявлених помилок	Кількість невиявлених помилок, %
(3, 2)	67072	251	0,374
(4, 3)	17975296	66327	0,369
(5, 4)	4853329920	17521051	0,361
(6, 5)	1339519057920	4755290656	0,355

Графічне відображення залежності невиявлених помилок у відсотках від кількості модулів наведено на рисунку 2.4.

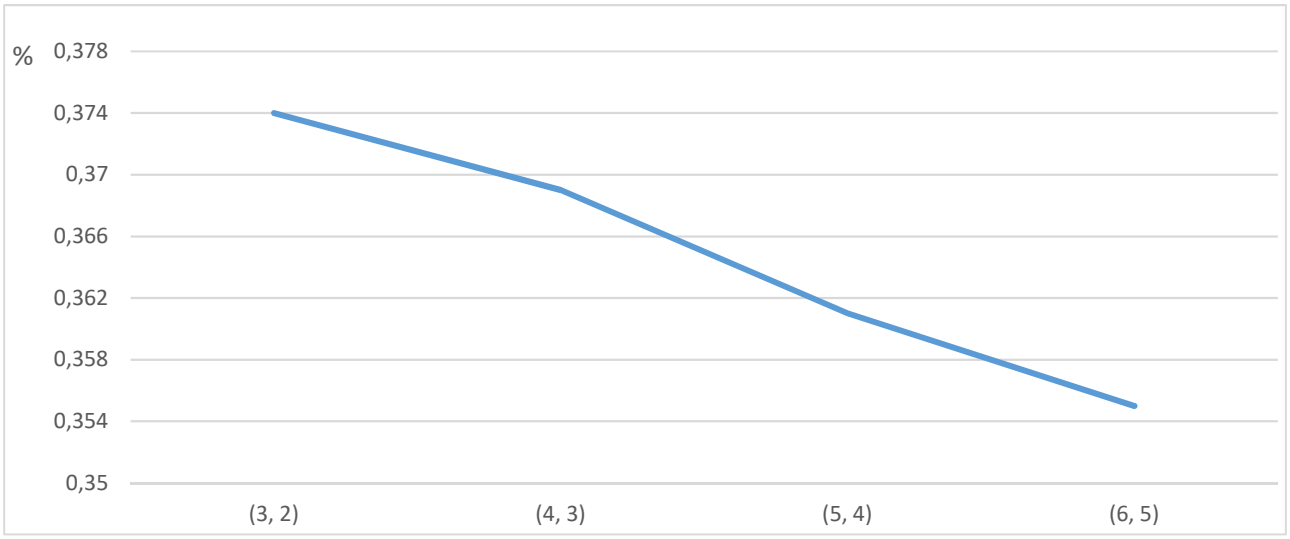

Рисунок 2.4 – Залежність (%) невиявлених помилок від кількості модулів

Як видно з рисунку 2.4, зі збільшенням числа інформаційних модулів та комбінацій спотворених символів кількість невиявлених помилок у відсотковому поданні практично залишається без змін.

На основі результатів дослідження можна зробити висновок, що збільшення кількості помилок при зміні кількості інформаційних модулів прямо пропорційне збільшенню робочого діапазону.

2.3.2. Аналіз методів зворотного перетворення системи залишкових класів

Окрім згаданого раніше методу зворотного перетворення (формула 2.5) згідно Китайської теореми про залишки (КТЗ) поширені й інші методи переведення із СЗК в позиційну систему числення [99]. Одним із таких методів зворотного перетворення є використання нової китайської теореми про залишки I (КТЗ I).

У цьому випадку відновлення початкової інформації відбувається за наступною формулою [100]:

$$X = (\mathrm{x}_0 + w_0 p_0(\mathrm{x}_1 - \mathrm{x}_0) + w_1 p_0 p_1(\mathrm{x}_2 - \mathrm{x}_1) + \cdots \\ \cdots + w_{n-1} p_0 p_1 * \cdots * p_{n-1}(\mathrm{x}_n - \mathrm{x}_{n-1}))\ mod\ P \qquad (2.13)$$

де w_i – набір коефіцієнтів, які забезпечують ортогональність перетворень та задовольняють умову:

$$w_i = (\prod_1^j p_i)^{-1} mod\ (\prod_{j+1}^n p_i). \qquad (2.14)$$

При аналізі існуючих методів відновлення також було розглянуто нову китайську теорему про залишки 2 (КТЗ II) [100].

Порівняння складності відновлення інформації було проведено для системи з 4-х модулів. Згідно поставленої задачі, відновлення даних відбувається за формулою [100]:

$$X = N_2 + [k_2(N_1 - N_2)] mod\ P, \qquad (2.15)$$

де N_1 та N_2 обчислюються за формулами:

$$N_1 = x_1 + [w_0(x_0 - x_1)] mod\ p_0 p_1; \qquad (2.16)$$

$$N_2 = x_3 + [k w_1(x_2 - x_3)] mod\ p_3. \qquad (2.17)$$

У вище згаданих формулах (2.16-2.17) використовується набір коефіцієнтів w_i, які мають задовольняти наступні рівності:

$$w_0 p_1 \equiv 1 \bmod p_0; \quad (2.18)$$

$$w_1 p_3 \equiv 1 \bmod p_2; \quad (2.19)$$

$$w_3 p_2 p_3 \equiv 1 \bmod (p_0 p_1). \quad (2.20)$$

У результаті обчислення кожного із вище згаданих методів відновлення даних ми отримаємо початкове значення *X*.

Приклад. Розглянемо приклади реалізації кожного із згадуваних вище методів зворотного перетворення.

Візьмемо систему модулів з мінімальними значеннями p_i={3, 5, 7, 11}, *n*=4.

Загальний діапазон системи модулів становить $P = 3 \cdot 5 \cdot 7 \cdot 11 = 1155$M $= 3*5*7*11 = 1155$, а робочий H $= 3 \cdot 5 \cdot 7 = 105$.

Для прикладу розглянемо розділення та відновлення інформаційного повідомлення X=80, що подане у десятковій системі числення.

Зазначене число у системі залишків згідно формули (2.1) по відповідних модулях матиме вигляд: X_i={2, 0, 3, 3}.

Використовуючи кожен із методів, виконаємо зворотне перетворення і отримаємо початкове значення *X*.

Метод 1. Для відновлення числа згідно КТЗ обчислимо базисні значення згідно формули 2.3:

b_i={385, 231, 330, 210}.

При підстановці відповідних значень b_i та x_i ми отримаємо (формула 2.2):

$$X_j=(x_0 b_0+x_1 b_1+x_2 b_2+x_3 b_3) \bmod P=(2 \cdot 385 + 0 \cdot 231 + 3 \cdot 330 +$$

$$+3 \cdot 210) \bmod 1155 = 2390 \bmod 1155 = 80.$$

Метод 2. Для відновлення числа згідно КТЗ I (формула 2.13) знаходимо набір коефіцієнтів t_i відповідно до формули (2.14):

$$w_0=(p_0)^{-1} mod\ (p_1 p_2 p_3) = (3)^{-1} mod\ (385) = 257;$$

$w_1=\left(\mathrm{p}_0\mathrm{p}_1\right)^{-1} mod\ (\mathrm{p}_2\mathrm{p}_3) = (15)^{-1} mod\ (77) = 36;$

$w_2=\left(\mathrm{p}_0\mathrm{p}_1\mathrm{p}_2\right)^{-1} mod\ \mathrm{p}_3 = (105)^{-1} mod\ (11) = 2.$

При підстановці залишків $\mathrm{x_i}$ та коефіцієнтів w_i у формулу (2.13) отримуємо шукане значення *X:*

$\mathrm{X}_j = (\mathrm{x}_0 + w_0 p_0(\mathrm{x}_1 - \mathrm{x}_0) + w_1 p_0 p_1(\mathrm{x}_2 - \mathrm{x}_1) + w_2 p_0 p_1 p_2(\mathrm{x}_3 - \mathrm{x}_2)\ mod\ P = (2 + 257 \cdot 3 \cdot (0-2) + 36 \cdot 3 \cdot 5 \cdot (3-0) + 2 \cdot 3 \cdot 5 \cdot 7 \cdot (3-3)) mod\ 1155 = (2 + (-1542)\ + 1620 + 0) mod\ 1155 = 80.$

Метод 3. При використанні КТЗ ІІ обчислюємо значення згідно рівностей (2.18-2.20):

$w_0 p_1 \equiv 1 \bmod p_0 =\ 5 w_1 \equiv 1 \bmod 3, w_0 = 2;$

$w_1 p_3 \equiv 1 \bmod p_2 = 11 w_1 \equiv 1 \bmod 7, w_1 = 2;$

$w_2 p_2 p_3 \equiv 1 \bmod (p_0 p_1) = 77 w_3 \equiv 1 \bmod 15, w_2 = 8.$

Після обчислення коефіцієнтів w_i, наступним кроком необхідно знайти значення N_1 та N_2 (формули 2.16 і 2.17):

$N_1 = x_1 + ([w_0(x_0 - x_1)] mod\ p_0) p_1 = 0 + [2(2-0)] mod\ 3 \cdot 5 = 5;$

$N_2 = x_3 + [w_1(x_2 - x_3)] mod\ p_3 = 3 + [2 \cdot (3-3)] mod\ 11 = 3.$

За результатами розрахунків на основі формули (2.15) отримаємо відновлене значення *X*:

$X = N_2 + [k_2(N_1 - N_2)] mod\ P = 3 + ([8 \cdot (5-3)] mod\ 15) \cdot 77 = 80.$

Слід відмітити, що кожен із вищезгаданих методів має свою послідовність виконання операцій. Деякі змінні, такі як базисні числа b_i та коефіцієнти w_i, для багаторазового зворотного перетворення не потрібно обчислювати кожного разу і можна обчислити заздалегідь та зберегти для подальшого використання. Це дозволяє зменшити кількість кроків та відповідно збільшити швидкодію роботи системи в цілому. При реалізації у розподілених системах КТЗ ІІ має значні переваги, оскільки всі кроки, окрім останнього, можуть виконуватися паралельно, а останній має меншу складність обчислення, ніж аналоги у КТЗ та КТЗ І [101, 102].

2.3.3. Аналітичні вирази оцінки складностей методів зворотного перетворення системи залишкових класів

Для проведення досліджень часової складності КТЗ необхідно визначити набір базових операцій, до яких відносяться: додавання, пошук залишку, добуток k-розрядних чисел, ділення [103].

У таблиці 2.10 представлено залежність часової складності базових операцій від розрядності вхідних даних.

Таблиця 2.10

Часова складність базових операцій КТЗ

№з/п	Базова операція	Часова складність
1.	$x_i = X(mod\ p_i)$	$O_1(k+1)^2 \approx O_1(k^2+2\cdot k)$
2.	$P = \prod_{1}^{n} p_i$	$O_2(n\cdot k^2)$
3.	$b_i = \frac{P}{p_i}\cdot w_i \equiv 1 mod\ p_i$	$O_3(2\cdot k^2)$
4.	$X = \left(\sum_{i=1}^{n} x_i b_i\right)\ mod\ P$	$O_4(n\cdot(k^2+(k+1)^2)) \approx$ $\approx O_4(2\cdot n\cdot k^2+2\cdot n\cdot k+n)$

При проведенні розрахунків також необхідно враховувати, що частиною операцій, які є на порядок простішими, можна нехтувати, оскільки вони не впливають на загальну складність (наприклад, у формулах №1 та №4 в таблиці 2.10).

Згідно формул, представлених в таблиці 2.10, загальна складність КТЗ буде обчислюватись за формулою:

$$O_{КТЗ}(n\cdot((k+1)^2+(n\cdot k^2)+(2\cdot k^2))+(2\cdot n\cdot k^2)) = O_{КТЗ}((n^2\cdot k^2)+(5\cdot n\cdot k^2)) \approx$$
$$\approx O_{КТЗ}(n^2\cdot k^2) \qquad (2.20)$$

Часова складність базових операцій КТЗ І наведена в таблиці 2.11.

Таблиця 2.11

Часова складність базових операцій КТЗ І

№з/п	Базова операція	Часова складність
1.	$X_j = (x_0 + t_0 p_0 (x_1 - x_0) + t_1 \cdot \cdot\ p_0 p_1 (x_2 x_1) \cdots t_{n-1} p_0 p_1 \cdot \cdot \cdots \cdot p_{n-1} \cdot (x_n - x_{n-1}))\ mod\ P$	$O_5((n \cdot k)^2 + (k+1)^2) = = O_5((n+1) \cdot k^2 + 2 \cdot k + 1)$
2.	$t_{n-1} = (p_1 \cdot p_2 \cdot p_3 \cdot \ldots p_{n-1})^{-1} \bmod p_n$	$O_6(35 \cdot k^2)$

Згідно формул, представлених в таблиці 2.11, загальна складність КТЗ І становитиме:

$$O_{КТЗ\_I}((n+1) \cdot k^2 + 2 \cdot k + 1) + 35 \cdot k^2) = O_{КТЗ\_I}((n+1) \cdot k^2 + 37 \cdot k + 1) \approx$$
$$\approx O_{КТЗ\_I}((n+1) \cdot k^2 + 37 \cdot k) \qquad (2.21)$$

Визначимо часову складність базових операцій КТЗ ІІ (таблиця 2.12).

Таблиця 2.12

Часова складність базових операцій КТЗ ІІ

№з/п	Базова операція	Часова складність
1.	$N_1 = x_2 + [k_1 \cdot (x_1 - x_2)] \bmod (p_1 \cdot p_2)$ $N_2 = x_4 + [k_2 \cdot (x_3 - x_4)] \bmod p_4$	$O_7((k+1)^2 + k^2 + 2 \cdot k^2) \approx O_7(4 \cdot k^2)$
2.	$X = N_2 + [k_3 \cdot (N_1 - N_2)] \bmod P$	$O_8((k+1)^2 + k^2 + 2 \cdot k^2) \approx O_8(4 \cdot k^2)$
3.	$k_1 \cdot p_2 \equiv 1 \bmod p_1$ $k_2 \cdot p_4 \equiv 1 \bmod p_3$ $k_3 \cdot p_3 \cdot p_4 \equiv 1 \bmod (p_1 \cdot p_1)$	$O_9(k^2)$

Згідно формул, представлених в таблиці 2.12, загальна складність КТЗ ІІ для 4 модулів буде оцінюватись як:

$$O_{КТЗ\_II}(9 \cdot k^2). \quad (2.22)$$

Для обчислення залежності часової складності від розрядності модулів було проведено порівняння систем із чотирьох модулів різної розрядності. На основі формул (2.20-2.22) отримаємо наступні значення оцінок для кожного із методів:

$$O_{КТЗ}(4^2 \cdot k^2) = O_{КТЗ}(16 \cdot k^2);$$

$$O_{КТЗ\_I}(2 \cdot 4 \cdot k^2) = O_{КТЗ\_I}(8 \cdot k^2);$$

$$O_{КТЗ\_II}(5 \cdot k^2).$$

Залежність часової складності методів зворотного перетворення графічно зображено на рисунку 2.5.

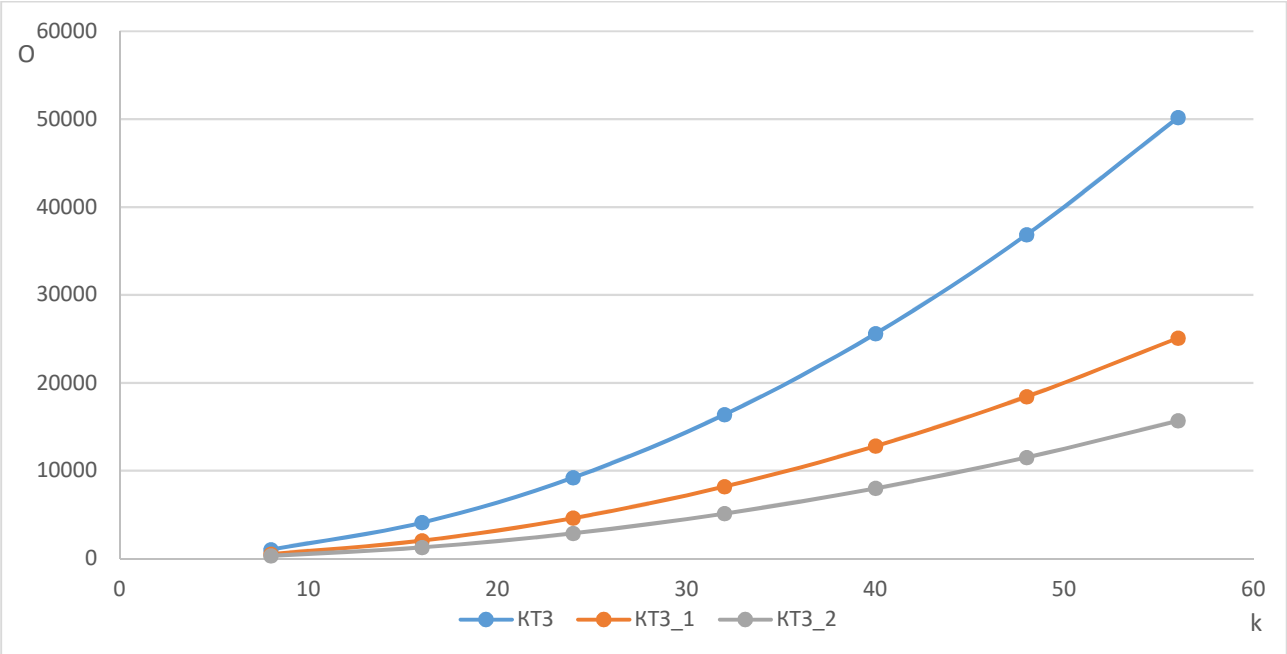

Рисунок 2.5 – Залежність часової складності КТЗ, КТЗ I та КТЗ II від розрядності k

Як видно з рисунку 2.5, часова складність обчислень КТЗ I менша в 2 рази порівняно з КТЗ, а КТЗ II – в 3,2 рази порівняно з КТЗ.

Результати чисельних експериментів вказують на переваги зворотного перетворення на основі КТЗ I і КТЗ II, які доцільно використовувати для підвищення швидкодії перетворення.

Проте, реалізація відновлення позиційного значення згідно методу КТЗ II потребує обчислень значення змінних N_1 та N_2 (формула 2.15, 2.16) кожного разу при обчисленні початкового значення X, що є неефективним у випадку роботи із масивами даних.

Висновки до розділу 2

У роботі для виявлення та виправлення помилок обрано використання коригуючих кодів на основі системи залишкових класів. Розглянуто методи переведення даних із десяткової системи в систему залишкових класів та навпаки.

Запропоновано метод захищеного зберігання даних на основі надлишкової системи залишкових класів та геш-функції. Визначені основні кроки та їх послідовність при кодуванні та декодуванні даних. Проведено розрахунок надлишковості для запропонованого методу та його порівняння з іншими методами виявлення та виправлення помилок.

Розраховано ймовірності не виявлення помилок у інформаційних символах при використанні одного перевірочного символа різної розрядності.

Проведено дослідження виявлення помилок у послідовностях інформаційних символів. Проведено порівняння часової складності методів зворотного перетворення системи залишкових класів на основі Китайської теореми про залишки, результати представлено графічно.

РОЗДІЛ 3. МЕТОД ШИФРУВАННЯ ДАНИХ НА ОСНОВІ НАДЛИШКОВОЇ СИСТЕМИ ЗАЛИШКОВИХ КЛАСІВ ТА ПСЕВДОВИПАДКОВОЇ ПОСЛІДОВНОСТІ

3.1. Розрахунок оптимального набору модулів для реалізації систем захищеного зберігання даних

Вибір оптимального набору модулів для кожної конкретної задачі є важливим завданням при проектуванні СЗК. Як згадувалося в розділі 2, основою проведення розрахунків у СЗК є обчислення діапазонів значень загального та робочого діапазонів (формули 2.1 і 2.4), від яких залежить максимальна розрядність інформаційного повідомлення, яке зможе опрацювати розрахована СЗК за один цикл роботи.

Відповідно, на відміну від теоретичних досліджень, де ми не прив’язані до будь-яких умов при виборі модулів, при практичній реалізації виникає кілька умов, які необхідно задовільнити [104].

Першою умовою є те, що найменшою адресованою коміркою пам’яті є байт. Відповідно, значення модулів потрібно обирати так, щоб розрядність отриманого робочого діапазону була кратна восьми бітам, тобто одному байту. А саме значення максимально наближалося до умови:

$$H \geq 2^{8 \cdot c} + 1, \tag{3.1}$$

де c – ціле число, яке визначає розмір блоку даних, що зчитуватиметься з вхідного файлу.

Також необхідно враховувати, що для модулів СЗК використовуються взаємно прості числа, значення яких впливає на величину робочого діапазону [105].

Наприклад, при $c = 2$ значення робочого діапазону має задовольняти умову $H \geq 65537$. Для виконання даної умови одним із варіантів є набір модулів {41, 43,

47} зі значенням $H = 82861$, а іншим: {3, 5, 7, 11, 13, 17}, $H = 255255$.

Як бачимо, обидва значення задовільнять умову, проте в другому випадку кількість модулів у 2 рази більша і, відповідно, кількість комірок пам'яті для зберігання залишків також потрібна вдвічі більша.

З такого порівняння можна вивести умови вибору оптимального набору модулів, а саме залежність від їх кількості та розрядності.

Також необхідно врахувати те, що кількість модулів не можна збільшувати без обмежень, оскільки це ускладнює операції запису та зчитування даних з пам'яті. Окрім цього, швидкість виконання арифметичних операцій із модулями однакової розрядності займає однаковий час, тому вибір модулів різної розрядності є не доречним [102].

При порівнянні наборів модулів {41, 43, 47} і {3, 5, 7, 11, 13, 17}, які мають однакову розрядність – один байт та робочий діапазон два байти, перший набір ефективніший в 2 рази, оскільки залишки при обчисленні займатимуть три байти проти шести в іншого.

Іншою умовою, яку рекомендовано враховувати при практичній реалізації, є взаємна сумісність обладнання. Для забезпечення цієї умови модулі необхідно підбирати так, щоб вони мали однакову розрядність, що дозволить використовувати накопичувачі одного розміру та типу, а також дозволить застосовувати один резервний накопичувач для заміни при виході з ладу вже підключеного пристрою. Також при розрахунку системи на основі надлишкової СЗК постає задача обґрунтування вибору кількості перевірочних модулів.

Згідно звіту Backblaze Storage Cloud за 2022 рік [106], відсоток виходу з ладу пристроїв зберігання даних (ПЗД) в незначній мірі залежить від об'єму самого пристрою та значною мірою від часу експлуатації. Для пристроїв, які безперебійно працюють більше 8 років, ймовірність виходу з ладу не перевищує 3,73% для малих накопичувачів (до 10 Тб), а для дисків розміром 12-16 Тб в середньому становить 1,07%.

Це підтверджує високу надійність сучасних пристроїв зберігання даних та дозволяє при розрахунку кількості перевірочних модулів СЗК, необхідних для

виявлення та виправлення помилок, розглядати використання одного перевірочного модуля.

Щоб визначати ефективність кожного із наборів модулів, необхідно оцінити кількість пам’яті, яку обраний набір модулів зможе опрацювати та яку займатиме кожна із реалізацій на носіях. Найменша допустима кількість модулів в надлишковій СЗК повинна містити два інформаційних та один перевірочний модулі, тобто $n = 3$.

Згідно формули 3.1 дані можна опрацьовувати блоками розміром $8*c$ біт або c байт. Відповідно для оцінки пам’яті, затраченої на зберігання модулів, необхідно обчислити, розмір пам’яті, який займуть модулі системи на носіях в байтовому представленні:

$$P_{r\_b} = \sum_{1}^{n} \lceil \log_{255} p_i \rceil . \qquad (3.2)$$

Відповідно ефективність кожного із наборів модулів можна заздалегідь порахувати, обчисливши розмір пам’яті у байтах, який система може опрацювати H_b та який займуть модулі на носіях P_{r_b}. Визначивши такі параметри, доступним стає обчислення надлишковості системи в байтовому N, та у відсотковому значенні R.

В таблиці 3.1 представлено характеристики системи для наборів модулів, при n=3, які відповідають зазначеним вище вимогам та відрізняються розрядністю.

Як видно із таблиці 3.1, використання набору модулів з двома інформаційними є неефективним, оскільки $R \geq 80$ % у розглянутих прикладах, тобто подвоює необхідний простір для зберігання інформації.

Набір модулів {16779979, 16779989, 16779991} та наступні після нього займають чотири байти і більше для кожного із модулів, що значно ускладнює розрахунки, тому їх використання є неефективним.

Таблиця 3.1

Характеристики системи для наборів модулів, при n=3

№ з/п	p_0	p_1	$p_{пер}$	H	P_{r_b}	H_b	N	R, %
1.	17	19	23	323	3	1	2	200
2.	257	263	269	$6{,}76*10^4$	6	2	4	200
3.	4099	4111	4127	$1{,}69*10^7$	6	3	3	100
4.	65 537	65 539	65 543	$4{,}30*10^9$	9	4	5	125
5.	1 049 963	1 049 977	1 049 999	$1{,}10*10^{12}$	9	5	4	80
6.	16 777 259	16 777 289	16 777 291	$2{,}81*10^{14}$	12	6	6	100

Наступним набором модулів, який можна використати для реалізації СЗК, є три інформаційних та один перевірочний (таблиця 3.2).

Таблиця 3.2

Характеристики системи для наборів модулів, при n=4

№ з/п	p_0	p_1	p_2	p_{per}	H	P_{r_b}	H_b	N	R
1.	41	43	47	53	$8{,}28*10^4$	4	2	2	100
2.	257	263	269	271	$1{,}82*10^7$	8	3	5	167
3.	1621	1627	1637	1657	$4{,}32*10^9$	8	4	4	100
4.	10313	10321	10331	10333	$1{,}10*10^{12}$	8	5	3	60
5.	65 537	65 539	65 543	65 551	$2{,}82*10^{14}$	12	6	6	100
6.	416 107	416 147	416 149	416 153	$7{,}21*10^{16}$	12	7	5	71
7.	2 642 323	2 642 329	2 642 333	2 642 351	$1{,}84*10^{19}$	12	8	4	50
8.	16 777 259	16 777 289	16 777 291	16 777 331	$4{,}72*10^{21}$	16	9	7	78

Аналізуючи набір модулів №1 із таблиці 3.2 можна зазначити, що використання однобайтних модулів призводить до надлишку 100%. Використання наборів №2 та №3 також приводить до надлишковості 100% і більше, що є неефективним. Окрім цього, варто зазначити, що використання наборів модулів, починаючи з {16777259, 16777289, 16777291, 16777331}, які

займають чотири байти і більше, є також неефективним.

Наступним набором модулів, які можна використати для побудови СЗК, є послідовність із чотирьох інформаційних та одного перевірочного. Варіанти наборів таких модулів представлено в таблиці 3.3.

Таблиця 3.3

Характеристики системи для наборів модулів, при n=5

№ з/п	p_0	p_1	p_2	p_3	p_{per}	H	P_{r_b}	H_b	N	R
1.	61	67	71	73	79	$2{,}12*10^7$	5	3	2	67
2.	257	263	269	271	277	$4{,}93*10^9$	10	4	6	150
3.	1031	1033	1039	1049	1051	$1{,}16*10^{12}$	10	5	5	100
4.	4099	4111	4127	4129	4133	$2{,}87*10^{14}$	10	6	4	67
5.	16 369	16 381	16 411	16 417	16 421	$7{,}22*10^{16}$	10	7	3	43
6.	65 537	65 539	65 543	65 551	65 557	$1{,}85*10^{19}$	15	8	7	88
7.	262 147	262 151	262 153	262 187	262 193	$4{,}72*10^{21}$	15	8	6	67
8.	1 048 583	1 048 589	1 048 601	1 048 609	1 048 613	$1{,}21*10^{24}$	15	10	5	50
9.	16 777 259	16 777 289	16 777 291	16 777 331	16 777 333	$7{,}92*10^{28}$	20	12	8	67

З набору №1 видно, що використання однобайтних модулів уже є ефективним, оскільки надлишок становить 67%, що менше, ніж дублювання інформації при звичайному резервному копіюванні даних, а набори модулів №2 та №3 є неефективними, оскільки надлишок при їх використанні становить 150% та 100% відповідно.

Також варто зазначити, що залишки у наборі модулів №9 займають чотири байти, а, отже, як зазначалося раніше, їх та всі наступні використовувати неефективно.

Наступний набір модулів, який доцільно розглянути, це послідовність із п’яти інформаційних (k=5) та одного перевірочного (r=1) (таблиця 3.4).

Таблиця 3.4

Характеристики системи для наборів модулів, при n=6

№ з/п	p_0	p_1	p_2	p_3	p_4	p_{per}	H	P_{r_b}	H_b	N	R
1.	83	89	97	101	103	107	$7{,}45*10^{9}$	6	4	2	50
2.	257	263	269	271	277	281	$1{,}36*10^{12}$	12	5	7	140
3.	787	797	809	811	821	823	$3{,}38*10^{14}$	12	6	6	100
4.	2351	2357	2371	2377	2381	2383	$7{,}44*10^{16}$	12	7	5	71
5.	7151	7159	7177	7187	7193	7207	$1{,}90*10^{19}$	12	8	4	50
6.	21 617	21 647	21 649	21 661	21 673	21 683	$4{,}76*10^{21}$	12	9	3	33
7.	65 537	65 539	65 543	65 551	65 557	65 563	$1{,}21*10^{24}$	18	10	8	80
8.	198 673	198 689	198 701	198 719	198 733	198 761	$3{,}10*10^{26}$	18	11	7	64
9.	5 534 411	5 534 413	5 534 429	5 534 483	5 534 489	5 534 519	$5{,}19*10^{33}$	18	14	4	29
10.	16 777 259	16 777 289	16 777 291	16 777 331	16 777 333	16 777 337	$1{,}33*10^{36}$	24	15	9	60

На основі набору модулів №1 можна відзначити, що їх доцільно використовувати при розрядності робочого діапазону не менше чотирьох байт, оскільки саме таку розрядність дає використання однобайтних модулів. У свою чергу, набори №2 та №3 мають надлишковість більше 100%, а набір модулів №10 та наступні після нього складаються із чотирьох байтних модулів і є неефективними у використанні.

Наступний набір модулів, який доцільно розглянути, це послідовність із шести інформаційних та одного перевірочного (таблиця 3.5).

Таблиця 3.5

Характеристики системи для наборів модулів, при n=7

№ з/п	p_0	p_1	p_2	p_3	p_4	p_5	p_{per}	H	P_{r_b}	H_b	N	R
1.	101	103	107	109	113	127	131	$1{,}74*10^{12}$	7	5	2	40
2.	257	263	269	271	277	281	283	$3{,}84*10^{14}$	12	6	6	100
3.	643	647	653	659	661	673	677	$7{,}96*10^{16}$	14	7	7	100
4.	1621	1627	1637	1657	1663	1667	1669	$1{,}98*10^{19}$	14	8	6	75
5.	4099	4111	4127	4129	4133	4139	4153	$4{,}91*10^{21}$	14	9	5	56
6.	10313	10321	10331	10333	10337	10343	10357	$1{,}21*10^{24}$	14	10	4	40
7.	26 003	26 017	26 021	26 029	26 041	26 053	26 083	$3{,}11*10^{26}$	14	11	3	27
8.	65 537	65 539	65 543	65 551	65 557	65 563	65 579	$7{,}93*10^{28}$	21	12	9	75
9.	165 133	165 161	165 173	165 181	165 203	165 211	165 229	$2{,}03*10^{31}$	21	13	8	62
10.	416 107	416 147	416 149	416 153	416 159	416 167	416 201	$5{,}19*10^{33}$	21	14	7	50
11.	1 048 571	1 048 573	1 048 583	1 048 589	1 048 601	1 048 609	1 048 613	$1{,}33*10^{36}$	21	15	6	40
12.	2 642 231	2 642 239	2 642 257	2 642 287	2 642 291	2 642 323	2 642 329	$3{,}40*10^{38}$	21	16	5	31
13.	6 658 013	6 658 027	6 658 049	6 658 051	6 658 079	6 658 097	6 658 103	$8{,}71*10^{40}$	21	17	4	24
14.	16 777 259	16 777 289	16 777 291	16 777 331	16 777 333	16 777 337	16 777 381	$2{,}23*10^{43}$	28	18	10	56

Як видно з таблиці 3.5, набір модулів при n=7 доцільно використовувати при розрядності робочого діапазону не менше п’яти байт, оскільки саме таку розрядність дає використання модулів з розрядністю один байт.

Як і в таблиці 3.4, набори модулів №2 та №3 із таблиці 3.5 з надлишковістю 100% є неефективними у використанні, а набір №14 складається з модулів розрядністю чотири байти та є недоцільним у застосуванні.

Наступний набір модулів, який можна розглянути, це послідовність із семи інформаційних та одного перевірочного (таблиця 3.6).

Таблиця 3.6

Характеристики системи для наборів модулів, при n=8

№	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_{per}	H	P_{r_b}	H_b	N	R
1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*
1.	103	107	109	113	127	131	137	139	$3{,}09 * 10^{14}$	8	6	2	33
2.	257	263	269	271	277	281	283	293	$1{,}09 * 10^{17}$	16	7	9	129
3.	569	571	577	587	593	599	601	607	$2{,}35 * 10^{19}$	16	8	8	100
4.	1237	1249	1259	1277	1279	1283	1289	1291	$5{,}25 * 10^{21}$	16	9	7	78
5.	2753	2767	2777	2789	2791	2797	2801	2803	$1{,}29 * 10^{24}$	16	10	6	60
6.	6079	6089	6091	6101	6113	6121	6131	6133	$3{,}16 * 10^{26}$	16	11	5	45
7.	13421	13441	13451	13457	13463	13469	13477	13487	$7{,}98 * 10^{28}$	16	12	4	33
8.	29641	29663	29669	29671	29683	29717	29723	29741	$2{,}03 * 10^{31}$	16	13	3	23

Прод. таблиці 3.6

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*
9.	65 537	65 539	65 543	65 551	65 557	65 563	65 579	65 581	$5{,}20 * 10^{33}$	24	14	10	71
10.	144 719	144 731	144 737	144 751	144 757	144 763	144 773	144 779	$1{,}33 * 10^{36}$	24	15	9	60
11.	319 547	319 567	319 577	319 589	319 591	319 601	319 607	319 639	$3{,}40 * 10^{38}$	24	16	8	50
12.	705 631	705 643	705 689	705 713	705 737	705 751	705 763	705 769	$8{,}72 * 10^{40}$	24	17	7	41
13.	1 558 129	1 558 177	1 558 189	1 558 201	1 558 213	1 558 217	1 558 223	1 558 243	$2{,}23 * 10^{43}$	24	18	6	33
14.	3 440 719	3 440 771	3 440 807	3 440 819	3 440 839	3 440 849	3 440 861	3 440 863	$5{,}71 * 10^{45}$	24	19	5	26
15.	7 597 763	7 597 769	7 597 823	7 597 829	7 597 847	7 597 859	7 597 873	7 597 901	$1{,}46 * 10^{48}$	24	20	4	20
16.	16 777 259	16 777 289	16 777 291	16 777 331	16 777 333	16 777 337	16 777 381	16 777 421	$3{,}74 * 10^{50}$	32	21	11	52

На основі даних таблиці 3.6 можна відзначити, що набір модулів при n=8 доцільно використовувати при розрядності робочого діапазону не менше шести байт, оскільки саме таку розрядність дає застосування однобайтних модулів.

Також, як і в попередніх таблицях, набори модулів №2 та №3 є надлишковими, тобто $N \geq H_b$, а це означає, що їх використання є недоцільним. Останній набір (№16), який вказаний у таблиці 3.6, також складається із чотирьох байтних модулів та є недоцільним у використанні.

На основі таблиць 3.1-3.6 можна побудувати загальну таблицю наборів модулів для проектування систем зберігання даних на основі СЗК (додаток Б), за допомогою якої у подальшому можна обрати залишки, що забезпечать надлишковість менше 100%, а також зробити такі висновки:

- оскільки при n=3 значення $R \geq 100\%$, у більшості випадків

використання таких наборів модулів є надлишковим;

- ефективні набори модулів, тобто такі, які задовольняють умову $N \leq H_b$, можуть бути побудовані, починаючи із значення H_b=3;
- додавання ще одного модуля такої самої розрядності збільшує H_b та, відповідно, зменшує R;
- збільшення розрядності дозволяє зберегти кількість модулів, проте це не завжди доцільно.

3.2. Дослідження криптографічної стійкості шифрування даних в системі залишкових класів

При класичному шифруванні існує поняття ключа K, за допомогою якого здійснюється шифрування даних. У СЗК роль ключа виконують кількість та значення модулів p_i, перетворюючи відкриту інформацію в шифротекст – набір залишків [107-109].

Завдяки зчитуванні інформації блоками певного розміру та обчисленні залишків по них інформація стає шифротекстом для зловмисника і простого перехоплення всіх файлів залишків недостатньо для розшифрування.

Відповідно, для отримання початкового вмісту файлу зловмиснику потрібно виконати зворотне перетворення СЗК, а для цього необхідно визначити кількість та значення модулів.

3.2.1. Дослідження стійкості шифрування даних на основі асимптотичного розподілу простих чисел

Першим кроком для розшифрування зловмисником даних є визначення модулів, на основі яких побудована система.

Як уже зазначалося раніше, у якості модулів використовують прості числа. Відповідно для їх визначення необхідно перебрати всі можливі варіанти.

Для цього можна використати асимптотичний розподіл простих чисел [110], згідно якого їх кількість на інтервалі від 0 до деякого q наближено визначається наступною формулою:

$$\pi(q) \approx \frac{q}{\ln q - 1,07}. \tag{3.3}$$

При використанні s-розрядних чисел межу q можна представити у вигляді $q=2^s$, а формула 3.3 матиме наступний вигляд:

$$\pi(s) \approx \frac{2^s}{(s \cdot \ln 2) - 1,07}. \tag{3.4}$$

За умови вибору системи із n модулів визначити приблизну кількість варіантів можна за формулою [111]:

$$L(n,s) \approx \prod_{\pi(s)-n}^{\pi(s)} \pi(s). \tag{3.5}$$

При малих значеннях s таке представлення є більш точним, проте за умови використання модулів з розрядністю $s \geq 16$, формулу (3.5) можна представити у вигляді:

$$L(n,s) \approx \left(\frac{2^s}{(s \cdot \ln 2) - 1,07} \right)^n \tag{3.6}$$

Враховуючи складність обчислення КТЗ, яка була розрахована в другому розділі (формула 2.20), та враховуючи складність визначення модулів системи (формула 3.6), криптографічну стійкість даних, представлених у вигляді залишків, можна подати у вигляді наступної формули [102]:

$$O(n,s) = \left(\frac{2^s}{s \cdot \ln 2} \right)^n \cdot n^2 \cdot 2^{2 \cdot s}. \quad (3.7)$$

При обчисленні криптографічної стійкості всі значення степенів доцільно округлювати в меншу сторону, що дозволить підібрати оптимальний набір модулів для забезпечення захищеності даних аналогічної алгоритму AES із довжиною ключа 128 біт [112, 113].

Для оцінки ефективності вибраної розрядності модулів проводиться порівняння криптографічної стійкості *O(n, s)* із криптографічною стійкістю *O(n, (s-8))*, тобто:

$$J = \frac{O(n,s)}{O(n,(s-8))}. \quad (3.8)$$

Значення *J* дозволяє оцінити ефективність збільшення розрядності модулів. Оскільки збільшення модуля на один розряд приводить до зростання обсягу пам'яті для його зберігання на один байт, то у формулі (3.8) крок розрядності залишків становить вісім.

Результати обчислення залежності криптографічної стійкості даних від розрядності модулів при двох інформаційних та одному перевірочному модулях представлено у таблиці 3.7.

Як видно з проведених розрахунків (таблиця 3.7), при збільшенні розрядності модуля значення степеню криптографічної стійкості $\lfloor \log_2 O(n,s) \rfloor$ збільшується в середньому на 24 порядки, що показує пряму арифметичну залежність, а значення *J* в цілому поступово зменшується, оскільки зростання ефективності від збільшення розрядності модулів у відсотковому відношенні падає.

Таблиця 3.7

Залежність криптографічної стійкості шифрування даних від розрядності модулів, при n=3

№ з/п	s	$O(n, s)$	$\lfloor \log_2 O(n,s) \rfloor$	J
1.	8	$5{,}67*10^{7}$	25	---
2.	16	$4{,}75*10^{14}$	48	1,92
3.	24	$5{,}32*10^{21}$	72	1,50
4.	32	$6{,}69*10^{28}$	95	1,32
5.	40	$8{,}98*10^{35}$	119	1,25
6.	48	$1{,}26*10^{43}$	143	1,20
7.	56	$1{,}81*10^{50}$	166	1,16
8.	64	$2{,}65*10^{57}$	190	1,14
9.	72	$3{,}95*10^{64}$	214	1,13
10.	80	$5{,}97*10^{71}$	238	1,11
11.	88	$9{,}10*10^{78}$	262	1,10
12.	96	$1{,}40*10^{86}$	286	1,09
13.	104	$2{,}17*10^{93}$	310	1,08
14.	112	$3{,}38*10^{100}$	333	1,07
15.	120	$5{,}29*10^{107}$	357	1,07
16.	128	$8{,}32*10^{114}$	381	1,07

У таблиці 3.8 представлено результати обчислення залежності криптографічної стійкості шифрування даних від розрядності при n=4.

Таблиця 3.8

Залежність криптографічної стійкості шифрування даних від розрядності, при n=4

№ з/п	s	$O(n, s)$	$\lfloor \log_2 O(n,s) \rfloor$	J
1.	8	$4{,}65*10^{9}$	32	---
2.	16	$4{,}99*10^{18}$	62	1,94
3.	24	$9{,}53*10^{27}$	92	1,48
4.	32	$2{,}30*10^{37}$	124	1,35
5.	40	$6{,}33*10^{46}$	155	1,25
6.	48	$1{,}89*10^{56}$	186	1,20
7.	56	$5{,}96*10^{65}$	218	1,17
8.	64	$1{,}96*10^{75}$	250	1,15
9.	72	$6{,}65*10^{84}$	281	1,12
10.	80	$2{,}31*10^{94}$	313	1,11
11.	88	$8{,}21*10^{103}$	345	1,10
12.	96	$2{,}96*10^{113}$	376	1,09
13.	104	$1{,}08*10^{123}$	408	1,09
14.	112	$4{,}02*10^{132}$	440	1,08
15.	120	$1{,}50*10^{142}$	472	1,07
16.	128	$5{,}67*10^{151}$	504	1,07

Як і у попередньому випадку, система із 4 модулів (таблиця 3.8) при зростанні розрядності модуля на 8 показує пряму арифметичну залежність із збільшенням криптографічної стійкості шифрування даних на 30-32 порядки.

Підставивши у формулу 3.7 кількість модулів n=5, отримаємо залежність криптографічної стійкості шифрування даних від розрядності при чотирьох інформаційних та одному перевірочному модулях (таблиця 3.9).

Таблиця 3.9

Залежність криптографічної стійкості шифрування даних від розрядності, при n=5

№ з/п	s	$O(n, s)$	$\lfloor \log_2 O(n,s) \rfloor$	J
1.	8	$3{,}36*10^{11}$	38	---
2.	16	$4{,}61*10^{22}$	75	1,97
3.	24	$1{,}50*10^{34}$	113	1,51
4.	32	$6{,}97*10^{45}$	152	1,35
5.	40	$3{,}92*10^{57}$	191	1,26
6.	48	$2{,}50*10^{69}$	230	1,20
7.	56	$1{,}73*10^{81}$	269	1,17
8.	64	$1{,}27*10^{93}$	309	1,15
9.	72	$9{,}83*10^{104}$	281	1,13
10.	80	$7{,}88*10^{116}$	313	1,11
11.	88	$6{,}51*10^{128}$	427	1,10
12.	96	$5{,}51*10^{140}$	467	1,09
13.	104	$4{,}77*10^{152}$	507	1,09
14.	112	$4{,}20*10^{164}$	546	1,08
15.	120	$3{,}75*10^{176}$	586	1,07
16.	128	$3{,}40*10^{188}$	626	1,07

Як і у попередньому випадку, система із 5 модулів (таблиця 3.9) при зростанні розрядності модуля на 8 показує пряму арифметичну залежність із збільшенням криптографічної стійкості шифрування даних у середньому на 39 порядків.

Залежність криптографічної стійкості шифрування даних від розрядності при п’яти інформаційних та одному перевірочному модулях згідно формули (3.7) представлено у таблиці 3.10.

Таблиця 3.10

Залежність криптографічної стійкості шифрування даних від розрядності, при n=6

№ з/п	s	$O(n, s)$	$\lfloor \log_2 O(n,s) \rfloor$	J
1.	8	$2{,}23*10^{13}$	44	---
2.	16	$3{,}92*10^{26}$	88	2,00
3.	24	$2{,}18*10^{40}$	134	1,52
4.	32	$1{,}94*10^{54}$	180	1,34
5.	40	$2{,}24*10^{68}$	227	1,26
6.	48	$3{,}04*10^{82}$	274	1,21
7.	56	$4{,}62*10^{96}$	321	1,17
8.	64	$7{,}62*10^{110}$	368	1,15
9.	72	$1{,}34*10^{125}$	415	1,13
10.	80	$2{,}47*10^{139}$	463	1,12
11.	88	$4{,}76*10^{153}$	510	1,10
12.	96	$9{,}45*10^{167}$	558	1,09
13.	104	$1{,}93*10^{182}$	605	1,08
14.	112	$4{,}04*10^{196}$	653	1,08
15.	120	$8{,}63*10^{210}$	700	1,07
16.	128	$1{,}88*10^{225}$	748	1,07

Як видно з таблиці 3.10, у системі із 6 модулів зростання степені криптографічної стійкості шифрування даних при зростанні розрядності модуля на 8 збільшується в середньому на 47 порядків.

Залежність криптографічної стійкості шифрування даних від розрядності при шести інформаційних та одному перевірочному модулях згідно формули (3.6) представлено у таблиці 3.11.

Таблиця 3.11

Залежність криптографічної стійкості шифрування даних від розрядності, при n=7

№ з/п	s	$O(n, s)$	$\lfloor \log_2 O(n,s) \rfloor$	J
1.	8	$1{,}40*10^{15}$	50	---
2.	16	$3{,}16*10^{30}$	101	2,02
3.	24	$3{,}00*10^{46}$	154	1,52
4.	32	$5{,}12*10^{62}$	208	1,35
5.	40	$1{,}21*10^{79}$	262	1,26
6.	48	$3{,}50*10^{95}$	317	1,21
7.	56	$1{,}17*10^{112}$	372	1,17
8.	64	$4{,}31*10^{128}$	427	1,15
9.	72	$1{,}73*10^{145}$	482	1,13
10.	80	$7{,}34*10^{161}$	537	1,11
11.	88	$3{,}28*10^{178}$	589	1,10
12.	96	$1{,}53*10^{195}$	648	1,10
13.	104	$7{,}40*10^{211}$	703	1,08
14.	112	$3{,}68*10^{228}$	759	1,08
15.	120	$1{,}88*10^{245}$	814	1,07
16.	128	$9{,}80*10^{261}$	870	1,07

При оцінці системи із 7 модулів можна відзначити, що середній приріст криптографічної стійкості шифрування даних при збільшенні модуля на 8 розрядів становить 55 порядків. Виняток становить лише використання початок діапазону, 8 і 16 розрядні модулі, при яких збільшення степеню становить 50 порядків.

Залежність криптографічної стійкості шифрування даних від розрядності при семи інформаційних та одному перевірочному модулях згідно формули (3.7) представлено у таблиці 3.12.

Таблиця 3.12

Залежність криптографічної стійкості шифрування даних від розрядності, при n=8

№ з/п	s	$O(n, s)$	$\lfloor \log_2 O(n,s) \rfloor$	J
1.	8	$8{,}45*10^{16}$	56	---
2.	16	$2{,}44*10^{34}$	114	2,04
3.	24	$3{,}95*10^{52}$	174	1,53
4.	32	$1{,}30*10^{71}$	236	1,36
5.	40	$6{,}26*10^{89}$	298	1,26
6.	48	$3{,}87*10^{108}$	360	1,21
7.	56	$2{,}83*10^{127}$	423	1,18
8.	64	$2{,}34*10^{146}$	486	1,15
9.	72	$2{,}13*10^{165}$	549	1,13
10.	80	$2{,}09*10^{184}$	612	1,11
11.	88	$2{,}18*10^{203}$	675	1,10
12.	96	$2{,}38*10^{222}$	738	1,09
13.	104	$2{,}72*10^{241}$	802	1,09
14.	112	$3{,}21*10^{260}$	865	1,08
15.	120	$3{,}92*10^{279}$	928	1,07
16.	128	$4{,}91*10^{298}$	992	1,07

Як і в попередньому випадку, початок діапазону, 8 і 16 розрядні модулі, дає нижчий приріст криптографічної стійкості шифрування даних $O(n, s)$ ніж середній в цілому, який становить 62-63 порядки.

Порівняння залежності криптографічної стійкості шифрування даних від розрядності для різної кількості модулів наведено на рисунку 3.1.

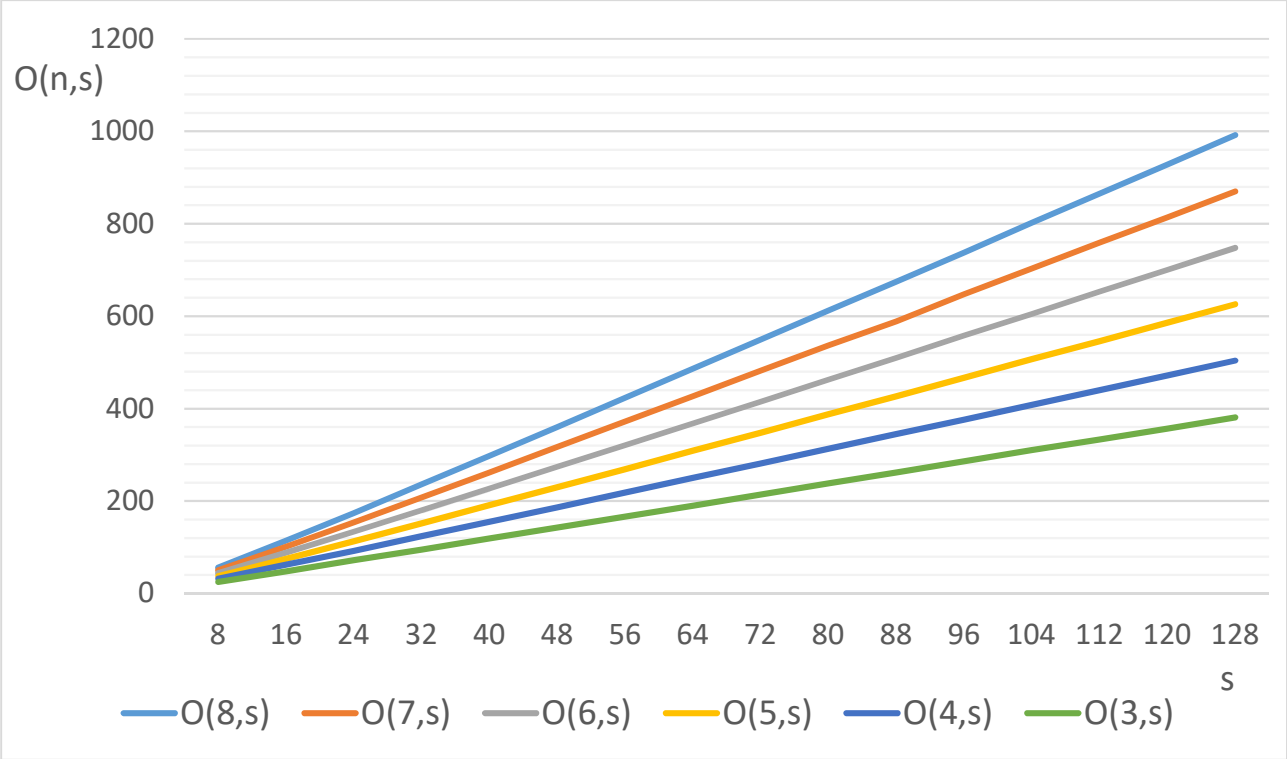

Рисунок 3.1 – Порівняння криптографічної стійкості шифрування даних для різної кількості модулів

За даними рисунка 3.1 видно, що криптографічна стійкість шифрування даних *O(n, s)* зберігає лінійність при збільшенні розрядності залишків. У цілому збільшення розрядності модулів підвищує криптографічну стійкість шифрування даних приблизно в 2 рази, при збільшенні розрядів з 8 до 16, з подальшим зменшенням ефективності кожного наступного до 1,07 раза при 120-128 розрядних модулях.

3.2.2. Оцінки криптографічної стійкості шифрування даних в системі залишкових класів

У криптографії одним із методів, на яких ґрунтується оцінка криптостійкості шифрів, є обчислювальна складність повного перебору всіх варіантів, а такий метод називають атакою грубої сили [114, 115].

На даний час одним із найбільш поширених алгоритмів шифрування інформації є симетричний алгоритм шифрування AES із довжиною ключа 128 біт [116, 117].

На основі результатів арифметичних розрахунків, поданих у таблицях 3.7-3.12, визначено розрядність модулів, необхідну для досягнення рівня захисту аналогічного алгоритму AES із довжиною ключа 128 біт (таблиця 3.13).

Таблиця 3.13

Розрядність модулів з ключем більше 128 біт

n	3	4	5	6	7	8
s	48	40	32	24	24	24
$\lfloor \log_2 O(n,s) \rfloor$	143	155	152	134	154	174

Для оцінки криптографічної стійкості шифрування даних, що передаються до віддаленого пристрою зберігання, запропоновано враховувати не тільки загальну криптографічну стійкість на основі КТЗ (формула 3.7), а й розмір файлів залишків, оскільки при перехопленні повідомлення зловмиснику невідомо, якої розрядності були обрані модулі. Тому для розшифрування повідомлення необхідно перебрати всі прості числа, які можуть використовуватись в якості модулів при шифруванні [118].

Згідно асимптотичного розподілу простих чисел (формула 3.3), кількість простих чисел у файлі залишків визначається за формулою:

$$S(f) = f \cdot \sum_{i=3}^{f} \frac{1}{i}, \tag{3.9}$$

де f – розрядність файлу залишку.

Оскільки розробка системи з надлишковістю більше 100% є недоцільною, то сума розмірів всіх файлів залишків не повинна перевищувати подвійний розмір початкового файлу. Відповідно розмір файлів залишків обчислюється за формулою:

$$f = \frac{2 \cdot F}{n}, \tag{3.10}$$

де F – розмір початкового файлу.

На основі формули 3.9 виведено залежність зростання криптографічної стійкості шифрування даних від розміру повідомлення, яка представлена у таблиці 3.14.

Таблиця 3.14

Залежність криптографічної стійкості системи від розміру повідомлення

Розмір файлу залишку	1 байт	1 Кб	1 Мб	10 Мб	50 Мб
S_f	9	$6{,}63*10^4$	$1{,}26*10^8$	$1{,}45*10^8$	$8{,}91*10^{10}$
s	3	16	26	27	36

Відповідно для знаходження значення модулів при перехопленні повідомлення буде враховуватися як криптографічна стійкість шифрування даних (формула 3.7), так і залежність криптографічної стійкості від розміру файлів залишків (формула 3.9).

При цьому, складність дешифрування повідомлень обчислюється за формулою:

$$O(n,s,f)=\left(\left(\frac{2^s}{s\ln 2}\right)^n\cdot n^2\cdot 2^{2\cdot s}\right)\cdot\left(f\cdot\sum_{i=3}^{f}\frac{1}{i}\right)=O\left(\sum_{i=3}^{f}\frac{1}{i}\cdot\left(\frac{2^s}{s\ln 2}\right)^n\cdot f\cdot n^2\cdot 2^{2\cdot s}\right). \qquad (3.11)$$

Відповідно, на основі формули 3.11 розраховано необхідну розрядність модулів для забезпечення криптографічної стійкості перехоплених повідомлень рівної AES із довжиною ключа 128 біт (табл. 3.15).

Таблиця 3.15

Оптимальна розрядність модулів з ключем більше 128 біт

n	3		4		5		6		7		8	
f_{min}	1 Кб	1 Мб	1 Кб	1 Мб	1 Кб	1 Мб	1 Кб	1 Мб	1 Кб	1 Мб	1 Кб	1 Мб
s	40	40	32	32	24	24	24	24	24	24	16	16
$\lfloor \log_2 O(n,s,f) \rfloor$	135	145	140	150	129	139	150	160	170	180	130	140

де f_{min} – мінімальна довжина повідомлення.

На основі таблиці 3.15 можна зробити висновок, що врахування мінімальної довжини файлів залишків 1 Кб при розрахунку криптографічної стійкості перехоплених повідомлень дозволяє зменшити розрядність модулів на 8 біт при забезпеченні необхідного рівня захисту.

3.3. Схема та криптостійкість методу шифрування даних в надлишковій системі залишкових класів

У роботі [118] досліджено криптографічну стійкість алгоритму шифрування даних в СЗК. Авторами запропоновано обчислювати криптографічну стійкість алгоритму шифрування на основі СЗК за формулою:

$$O(n,s) = \left(\frac{2^{s+1}}{n} \right)^n \cdot s^2 . \qquad (3.12)$$

Отримані авторами результати, необхідні для забезпечення криптографічної стійкості даних аналогічно алгоритму AES із довжиною ключа 128 біт, у залежності від кількості модулів, становлять від 39 біт при трьох модулях і до 14 біт при восьми модулях [120]. Проте, ці значення не враховують принципу побайтного зберігання даних на носіях. При прив’язці до байтного представлення згадані вище значення становитимуть 40 та 16 біт відповідно.

3.3.1. Метод шифрування даних в надлишковій системі залишкових класів з використанням псевдовипадкових послідовностей

Для зменшення розрядності модулів при заданій криптографічній стійкості алгоритму запропоновано удосконалений метод шифрування даних, суть якого полягає в заміні позицій залишків x_i згідно псевдовипадкової послідовності (ПВП). У якості ПВП вибирається М-послідовність, що забезпечує високу швидкість генерування бітів та необхідну довжину ключа [121, 122].

На рисунку 3.2 представлено схему шифрування на основі НСЗК, де Data – дані що шифруються; p_i – набір модулів; Cipher – блок шифрування; Decipher – блок дешифрування; S.key – М-послідовність, яка відповідає за тасування залишків. Оскільки було обґрунтовано, що при практичній реалізації використання модулів, менших 8 біт та не кратних 8 біт не доцільно, то операцію циклічного зсуву за ключем доцільно проводити на залишках, а не побітно.

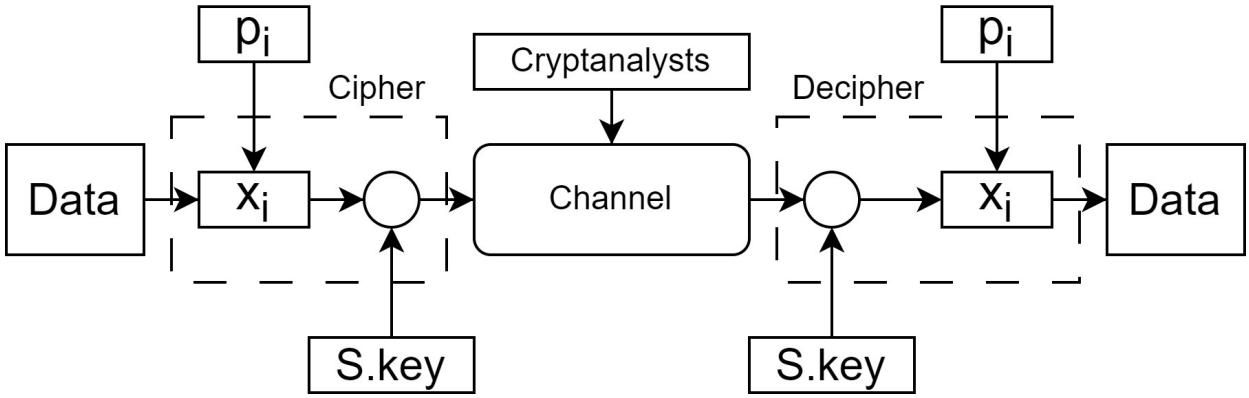

Рисунок 3.2 – Схема запропонованого методу шифрування на основі СЗК

Операція циклічного зсуву залишків виконується у залежності від значення псевдовипадкової послідовності, яка представлена у вигляді *‘0’* та *‘1’*. Запропоновано при сигналі *‘0’* обчислений залишок записувати у файли залишків без змін. У випадку, коли значення послідовності рівне *‘1’*, то залишки записуються у наступний за порядком файл залишків, а останній – в перший (рисунок 3.3).

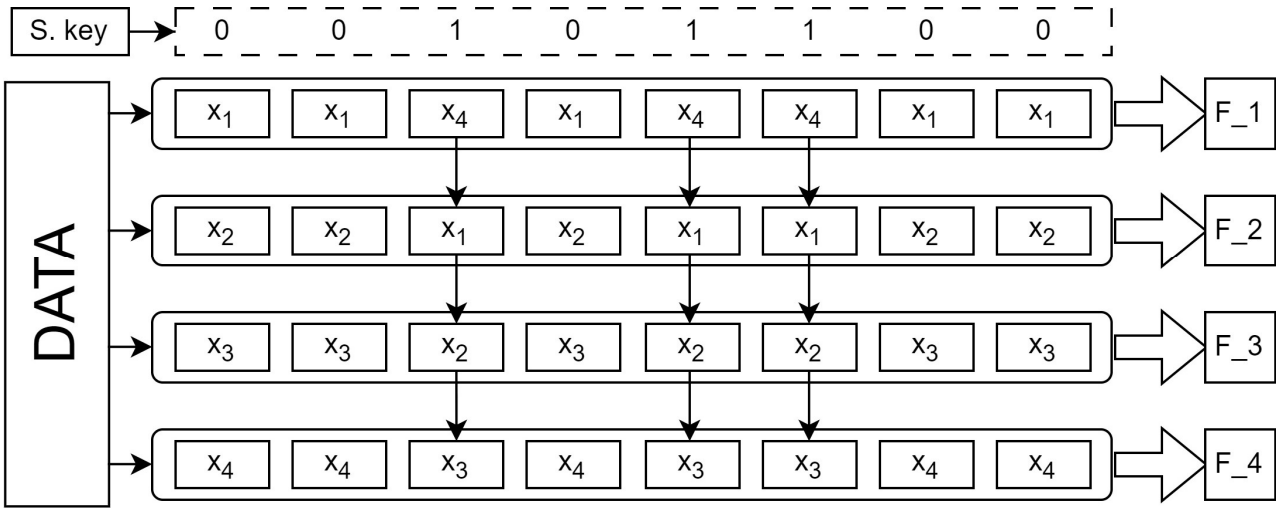

Рисунок 3.3 – Перетворення даних із застосуванням методу циклічного зсуву

При дешифруванні виконується обернена послідовність операцій, тобто спочатку вміст файлів залишків (F_1 - F_4) записується у відповідні масиви залишків, до яких згодом застосовується операція циклічного зсуву за ключем.

Враховуючи, що СЗК є не позиційною системою числення, циклічний зсув будь-якого із залишків на одну позицію ускладнює відновлення даних.

3.3.2. Криптографічна стійкість методу шифрування даних в надлишковій системі залишкових класів

У випадку, якщо набір модулів, що використовувався для розробки системи, невідомий зловмиснику, то він не зможе отримати вихідний файл навіть після отримання доступу до всіх файлів залишків.

Для відновлення правильної послідовності залишків йому необхідно перебрати кожен із можливих залишків в усіх файлах. Отримати правильний файл залишку можливо, тільки підібравши вірну послідовність для всіх файлів. Кількість залишків залежить від розрядності модулів, а складність перебору варіантів з урахуванням зсуву обчислюється за формулою:

$$O_{ch}(n,s,f)=\sum_{i=1}^{\frac{s}{8}} n!^{\frac{f}{8\cdot i}} \tag{3.13}$$

Враховуючи те, що раніше було запропоновано використовувати мінімальний розмір файлу залишку не менше 1 Кб, то підставивши відповідні значення в формулу 3.13 та враховуючи дані таблиці 3.15 визначаємо складність перебору залишків у файлі залишку (таблиця 3.16).

Таблиця 3.16

Складність перебору залишків зловмисником з невідомими модулями

n	3	4	5	6	7	8
s	40	32	24	24	24	16
$\lfloor \log_2 O_{ch}(n,s,f) \rfloor$	582	1064	1638	2106	2808	3010

Як ми бачимо з таблиці 3.16, складність перебору залишків $O_{ch}(n,s,f)$ з урахуванням зсуву досить висока, а її врахування дозволить збільшити криптостійкість проектованої системи та зменшити розрядність модулів.

Відповідно, враховуючи складність знаходження залишків (формула 3.7), криптографічну стійкість системи (таблиця 3.14), складність перебору залишків (таблиця 3.16) та надсилання повідомлень, залишки яких будуть не менше 1 Кб, можна розрахувати складність взламу системи при використанні різної кількості модулів з розрядністю менше 16 біт (таблиця 3.17).

Таблиця 3.17

Криптографічна стійкість системи з розрядністю модулів до 16 біт

n	3	4	5	6	7	8
$\lfloor \log_2 O_{sys}(n,s,f) \rfloor$	229	371	533	711	904	1109

Згідно таблиці 3.17, для забезпечення криптографічної стійкості аналогічної алгоритму AES із довжиною ключа 128 біт достатньо набору з двох інформаційних та одного перевірочного модулів. У таблиці 3.18 представлено залежність криптографічної стійкості системи від розрядності залишків при n=3.

Таблиця 3.18

Залежність криптографічної стійкості системи від розрядності модулів, при n=3

№ з/п	$2s$	$\lfloor \log_2 O_{sys}(n,s,f) \rfloor$	J
1.	8	394	---
2.	16	559	1,42
3.	24	667	1,19
4.	32	749	1,12
5.	40	813	1,09
6.	48	867	1,07
7.	56	914	1,05
8.	64	954	1,04
9.	72	990	1,04
10.	80	1021	1,03
11.	88	1049	1,03
12.	96	1075	1,02
13.	104	1098	1,02
14.	112	1121	1,02
15.	120	1141	1,02
16.	128	1161	1,02

Як видно з проведених розрахунків (таблиця 3.18) при збільшенні розрядності модуля на 8, розрядність криптографічної стійкості системи $\lfloor \log_2 O_{sys}(n,s,f) \rfloor$ збільшується не лінійно, проте ефективність J значно зменшується при збільшенні розрядності модулів до 24 біт та вище.

3.3.3. Оцінка криптографічної стійкості пропонованого методу

Порівняння криптографічної стійкості шифрування даних в НСЗК $\lfloor \log_2 O(n,s) \rfloor$ (таблиця 3.7) та розробленого методу шифрування даних $\lfloor \log_2 O_{sys}(n,s,f) \rfloor$ (таблиця 3.18) при n=3 приведено на рисунку 3.4.

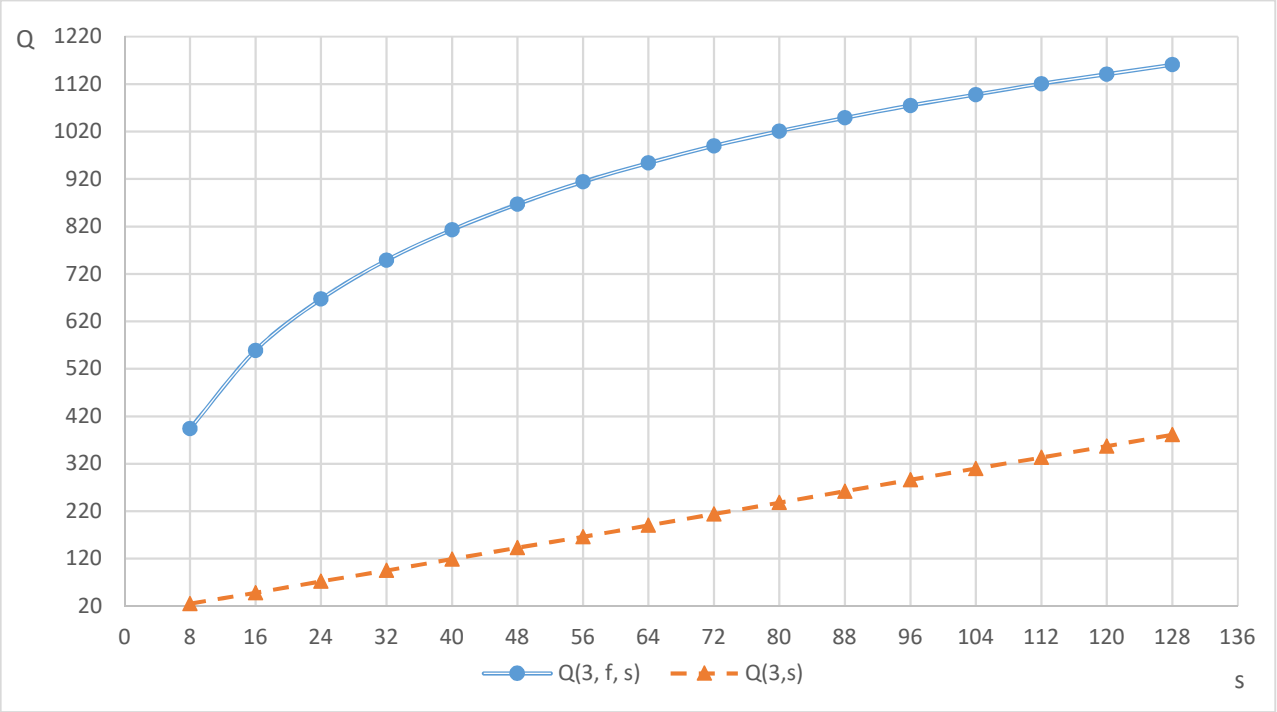

Рисунок 3.4 – Порівняння криптографічної стійкості СЗК *O(n, s)* та розробленого методу *Q (n, f, s)* від розрядності при n=3

Запропонований метод забезпечує в 11-15 раз вищу криптографічну стійкість із 8-16 бітними модулями, ніж класичний, і в середньому в 3 рази з 96-128-бітними модулями.

Оскільки для розрахунків було прийнято, що файли залишків повинні бути не менше 1 Кб, а використання модулів розрядністю 1 байт не доцільно, у зв'язку з зростанням часу операції зчитування/запису, то для здійснення циклічного зсуву даних достатньо ключа довжиною 512 біт.

Висновки до розділу 3

У роботі детально розглянуто умови, які виникають при розробці систем зберігання даних на основі НСЗК. Основою проведення розрахунків у НСЗК є обчислення діапазонів значень загального та робочого діапазонів, від яких залежить максимальна розрядність інформаційного повідомлення, яке зможе опрацювати розрахована СЗК за один цикл роботи.

Для визначення оптимального набору модулів проведено аналіз їх надлишковості при зберіганні в залежності від кількості.

Сформовано оптимальні набори модулів різної розрядності для використання у системах зберігання даних, які забезпечують надлишковість менше 100% (додаток А).

Досліджено криптографічну стійкість шифрування даних на основі асимптотичного розподілу простих чисел. Обчислено залежність криптографічної стійкості від розрядності модулів при різній їх кількості.

Для оцінки криптографічної стійкості шифрування даних, що передаються до віддаленого пристрою зберігання, запропоновано враховувати не тільки криптографічну стійкість шифрування даних на основі СЗК, а й розмір файлів залишків.

Для підвищення рівня криптографічної стійкості шифрування даних на основі НСЗК запропоновано удосконалення методу шифрування шляхом зміни позицій залишків з використанням ПВП. Розроблено схему шифрування на основі НСЗК та циклічного зсуву залишків, відповідно до ПВП.

Проведено дослідження криптографічної стійкості шифрування даних в НСЗК запропонованого методу та оцінено складність перебору залишків при невідомих модулях. Проведено порівняння криптографічної стійкості шифрування даних в НСЗК та пропонованого методу.

РОЗД ІЛ 4. АЛГОРИТМИ ЗАХИШЕНОГО ЗБЕРІГАННЯ ДАНИХ НА ОСНОВІ НАДЛИШКОВОЇ СИСТЕМИ ЗАЛИШКОВИХ КЛАСІВ

Розроблений у пункті 3.3 метод шифрування даних в НСЗК з використанням ПВП забезпечує конфіденційність даних при їх передаванні та зберіганні на розподілених носіях. Для реалізації вказаного методу необхідно розробити наступні алгоритми: шифрування даних на основі НСЗК та ПВП, перевірки цілісності файлів, дешифрування даних, зворотного перетворення та відновлення пошкоджених блоків згідно удосконаленого методу проекцій.

4.1. Шифрування даних на основі надлишкової системи залишкових класів та псевдовипадкової послідовності

Шифрування даних на основі НСЗК та ПВП полягає в розділенні початкового файлу на залишки та зміна позицій залишків відповідно до ключа шифрування, в якості якого використовуються М-послідовність. Для цього необхідно спочатку зчитати початковий файл, потім розділити його на блоки, з яких обчислити залишки і в подальшому записати отриманий результат у відповідні файли залишків.

4.1.1. Алгоритми переведення початкового файлу в масив блоків

Перший крок для здійснення шифрування даних в системі розподіленого захищеного зберігання даних (СРЗЗД) на основі НСЗК та ПВП полягає у зчитуванні файлу, тобто в перетворенні вхідного файлу у масив залишків (рисунок 4.1).

При проектуванні системи заздалегідь обчислюються оптимальні набори модулів, які відповідають заданим параметрам (кількість пристроїв зберігання, кількість надлишкових пристроїв та інші). Розмір блоку j, з яким система працюватиме надалі, залежить від робочого діапазону H.

Рисунок 4.1 – Алгоритм переведення початкового файлу в масив блоків

Інформація про параметри системи зберігається у налаштуваннях і першим кроком роботи є її зчитування (блок 1). Окрім цього, система передає інформацію про сам файл, який необхідно опрацювати. Враховуючи, що зберігання даних здійснюється на розподілених носіях, важливо забезпечити перевірку цілісності зчитаних файлів при їх дешифруванні та відновленні. Для забезпечення

цілісності файлу на етапі відновлення у розробленому методі (пункт 2.2) запропоновано обчислювати геш-значення початкового файлу (блок 2).

Для розбиття початкового файлу на блоки, в першу чергу, необхідно визначити довжину файлу (блок 3).

Наступний крок полягає в розбитті файлу на блоки, для чого виконується порівняння поточної позиції та довжини файлу *L* (блок 4). Якщо кінець файлу не досягнутий, то відбувається послідовне зчитування блоків розміром *j* байт та запис їх у тимчасову змінну *byte* (блок 5).

У випадку, коли блок, що залишився, менший за значення *j* (умова «НІ» блоку 4), то такий залишок також буде записаний у змінну *byte*, проте, окрім цього, розмір такого блоку буде окремо записаний у змінній *rewta* (блок 6).

На наступному кроці (блок 7) до масиву *a[ ]* буде додано значення змінної *byte*, а позиція зчитування інформації з файлу (лічильник *i*) буде зміщена на *j* байт. Ця послідовність дій буде повторюватися до тих пір, поки алгоритм не дійде до кінця початкового файлу (блок 8).

У результаті роботи файл розбивається на блоки заданої довжини *j* та записується у масив *a[ ]*. У системну інформацію записується змінна *rewta*, що містить розмір останнього блоку, і значення геш-функції від початкового файлу (блок 9).

4.1.2. Алгоритм шифрування залишків на основі М-послідовності

Наступним кроком роботи системи буде обчислення залишків з блоків згідно модулів, обраних для реалізації системи, та їх шифрування згідно з ПВП (рисунок 4.2).

У блоці 1 відбувається зчитування параметрів системи, а саме: вибрані модулі, лічильники, М-послідовність, інформація, отримана з попередніх алгоритмів, масиви, наприклад масиву даних *a[ ]*, з яким надалі працюватиме система. Подальша робота з вхідним масивом буде полягати в обчисленні довжини масиву (блок 2) для подальшого розбиття на блоки.

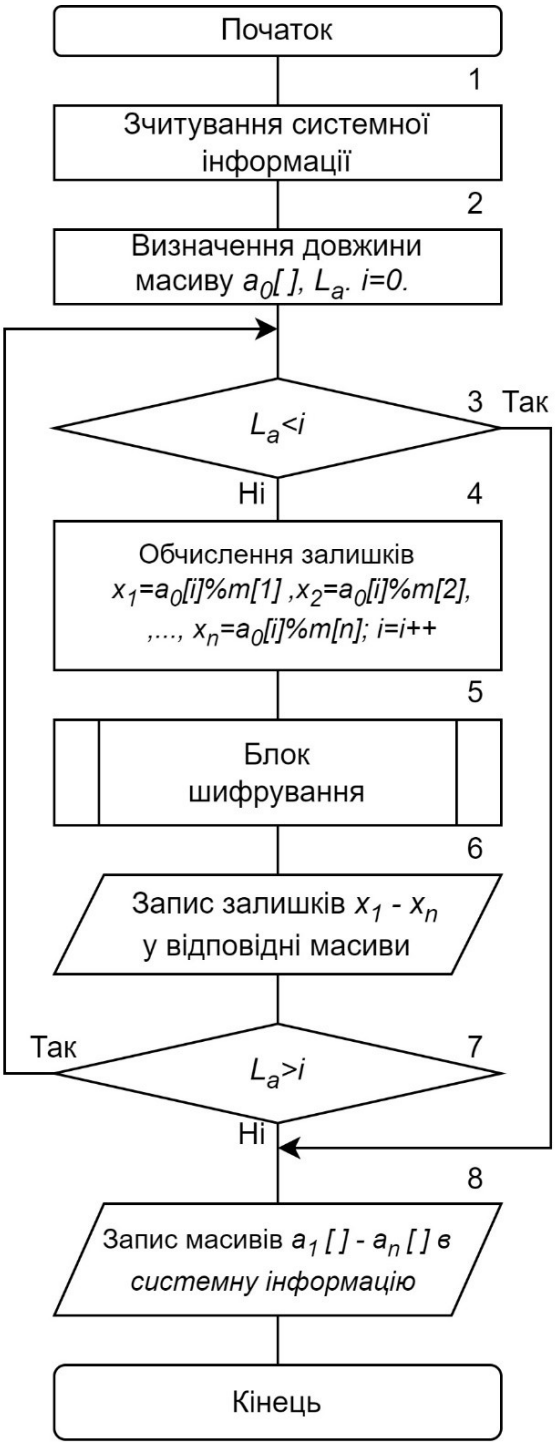

Рисунок 4.2 – Алгоритм шифрування залишків на основі М-послідовності

Для обчислення залишків, починаючи із першого елемента масиву *a[ ]* (блок 3) і до його закінчення, необхідно ділити кожен із елементів масиву на значення модулів (блок 4), які були обрані для поточної системи.

Наступним кроком роботи системи є виконання шифрування (блок 5) на основі методу, розробленого у пункті 3.3, який базується на циклічному зсуві залишків згідно М-послідовності.

В результаті таких дій ми отримаємо масиви залишків по кожному з модулів (блок 6). Для переходу між елементами масиву *a[ ]* лічильник *i* збільшується на одиницю (блок 4), що зміщує позицію зчитування. Після обробки всього вмісту масиву *a[ ]* і досягнення його кінця (умова виходу «НІ» з блоку 7), отримані масиви залишків зберігаються у системній інформації для подальшого їх запису у відповідні файли залишків (блок 8).

Результатом роботи приведеного алгоритму є розділення початкового масиву даних на *n* масивів зашифрованих залишків.

4.1.3. Алгоритм запису файлів залишків та додаткових параметрів

Наступним кроком у шифруванні буде запис отриманих залишків у відповідні файли залишків, а в назву для забезпечення цілісності даних записуються обчислені значення геш-функції та циклічного коду *CRC-32*. Алгоритм дій приведено на рисунку 4.3.

На першому кроці алгоритму (блок 1) відбувається зчитування системної інформації, яка необхідна для запису масивів залишків у відповідні файли та отриманні початкової назви файлу, оскільки вона потрібна для генерації назви файлів залишків.

Як вже зазначалося раніше, кожен залишок в залежності від розрядності займатиме певну кількість байт і алгоритм запису файлів залишків відповідає за вірний їх запис.

Від кількості модулів *n* залежить кількість файлів залишків (блок 2). Для кожного з них створюється окремий файл (блок 3). Підготовленні на етапі розділення масиви залишків записуються у відповідні файли (блок 4).

Згідно з методом, описаним у другому розділі, для забезпечення цілісності файлів обчислюються значення геш-функції від файлів (блок 5) та його запис у назву файлу (блок 6). Додатково для забезпечення цілісності при пересиланні файлів залишків обчислюється циклічний надлишковий код *CRC-32* (блок 7) та здійснюється його запис у назву файлу (блок 8).

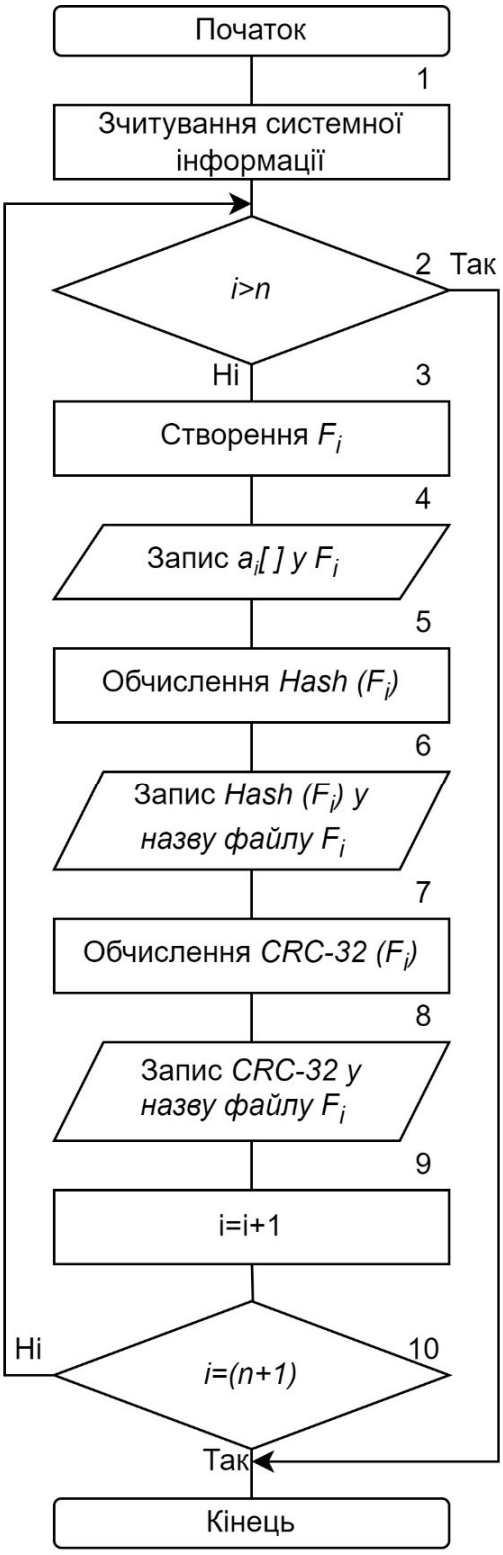

Рисунок 4.3 – Алгоритм запису файлів залишків та додаткових параметрів

Для переходу до наступного файлу залишку здійснюється збільшення лічильника *i* (блок 9) та перевірка, чи не виконувалось заповнення останнього файлу залишку (блок 10).

У результаті роботи алгоритму масиви залишків $a_i[\,] - a_n[\,]$ записуються у відповідні файли залишків, а в їх назву добавляється геш-значення та циклічний надлишковий код *CRC-32*.

4.2. Перевірка цілісності файлів залишків

При проектуванні розподілених систем зберігання даних важливо забезпечити перевірку цілісності даних при їх передачі. Для цього використовують різні методи, зокрема геш-функції та алгоритми обчислення контрольної суми. Як вже зазначалося раніше, для перевірки цілісності та достовірності файлів залишків у роботі запропоновано використання обчислення значення геш-функції та циклічного коду *CRC-32*.

4.2.1. Алгоритм зчитування файлів залишків та обчислення геш-функції

Першим етапом для перевірки цілісності файлів залишків є їх пошук та обчислення значень геш-функції та *CRC-32* для кожного із них (рисунок 4.4).

Для цього необхідно завантажити відповідну системну інформацію, наприклад, назву початкового файлу, набори модулів та інші дані (блок 1).

З системної інформації отримується кількість файлів залишків n, яка забезпечує пошук та обчислення необхідних параметрів, а також встановлює переходи між файлами (блок 2).

Для кожного файлу залишків необхідно знайти його у відповідній папці чи диску (блок 3). Такий пошук виконується на основі початкової назви файлу, а після його знаходження з назви файлу отримуються значення *CRC-32* (блок 4) та геш-значення (блок 5), обчислені при створенні файлів залишків.

Наступні кроки полягають в обчисленні значень *CRC-32* (блок 6) та геш-функції (блок 7) від знайденого файлу залишку. Після здійснення цих операцій відбувається перехід до наступного файлу залишку (блок 8).

Як зазначалося раніше, файл F_0 відповідає за початковий вміст файлу, тому перебір файлів залишків проходить з першого елементу до $(n+1)$ (блок 9).

Рисунок 4.4 – Алгоритм зчитування файлів залишків та обчислення геш-функції

Після здійснення всіх вищезазначених операцій інформація про обчислені та отримані з назв файлів залишків *CRC-32* та геш-значення зберігається в системну інформацію (блок 10) для використання на наступному кроці алгоритму перевірки цілісності файлів залишків.

4.2.2. Алгоритм перевірки цілісності файлів залишків

Як було зазначено в пункті 2.1, на основі СЗК для розподіленого зберігання даних можна відновити початкові дані, використовуючи лише інформаційні модулі, а в разі пошкодження одного з них – замінити його на перевірочний. Алгоритм вибору файлів залишків, які будуть використані для зворотного перетворення, показаний на рисунку 4.5.

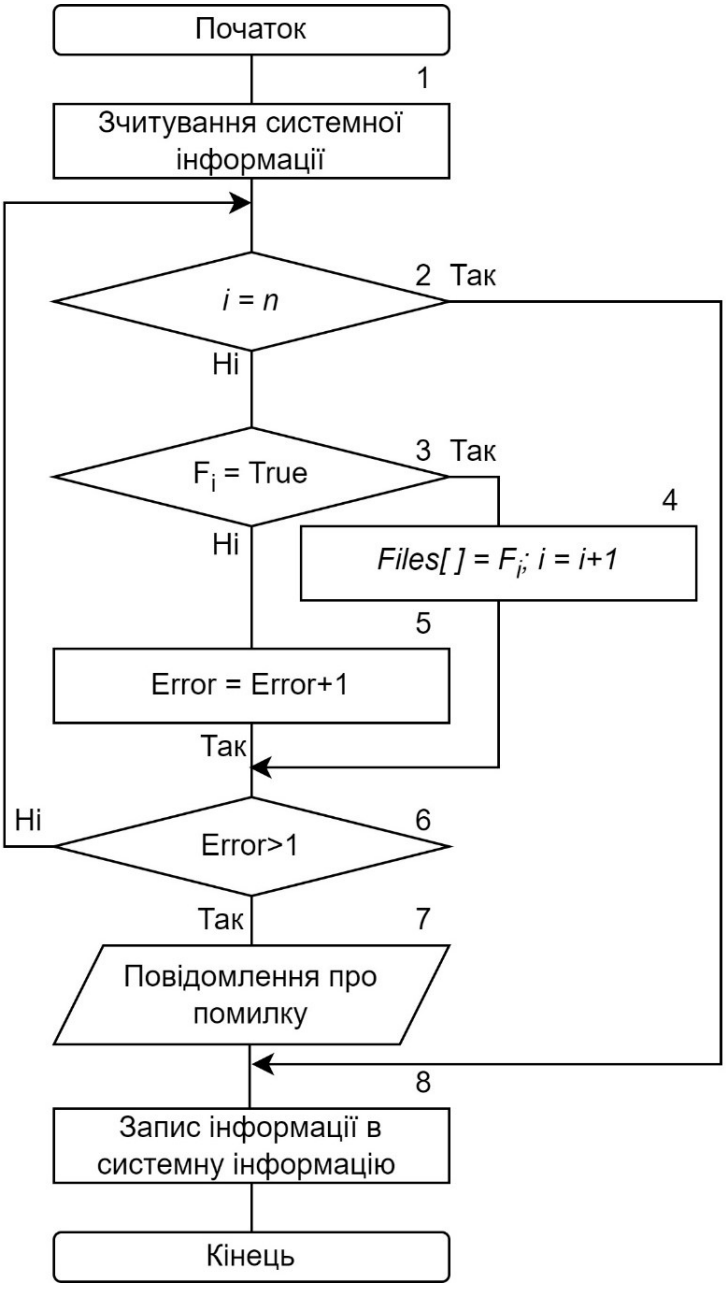

Рисунок 4.5 – Алгоритм перевірки цілісності файлів залишків

У блоці 1 здійснюється зчитування конфігурації системи, а також імена, геш-значення та *CRC-32* файлів залишків.

Для зворотного перетворення у початковий файл нам достатньо підтвердити цілісність файлів, які містять залишки за відповідними модулями. Алгоритм перевіряє доступність і цілісність послідовно, починаючи із першого файлу залишку (блок 2).

На попередньому кроці було обчислено геш-функції та *CRC-32* файлів залишків, а також зчитувалися значення, записані у назви файлів на етапі створення файлів залишків. Порівнявши відповідні значення, ми зможемо підтвердити цілісність файлу залишку (блок 3). У випадку підтвердження назва файлу залишку зберігається у відповідному масиві *Files []* (блок 4).

У разі виконання умови «НІ» блоку 3 змінна *Error*, яка відповідає лічильнику помилок і дорівнює нулю на початку роботи, збільшується на одиницю (блок 5).

У випадку, якщо два файли залишків неможливо прочитати, умова «ТАК» блоку 6, то виконати зворотне перетворення звичайним методом і отримати початковий файл неможливо, про що користувач буде повідомлений (блок 7).

Результатом роботи алгоритму є послідовність цілих та непошкоджених файлів залишків у масиві *Files []*, доступних для відновлення початкового файлу. Запис про виконані дії та їх результат буде збережено у системну інформацію (блок 8).

4.2.3. Алгоритм зчитування файлів залишків для зворотного перетворення

У випадку, якщо досягнута необхідна кількість файлів залишків, тобто кількість елементів у масиві *Files []* рівна кількості інформаційних модулів у системі, то здійснюється формування масивів залишків з відповідних файлів (рисунок 4.6).

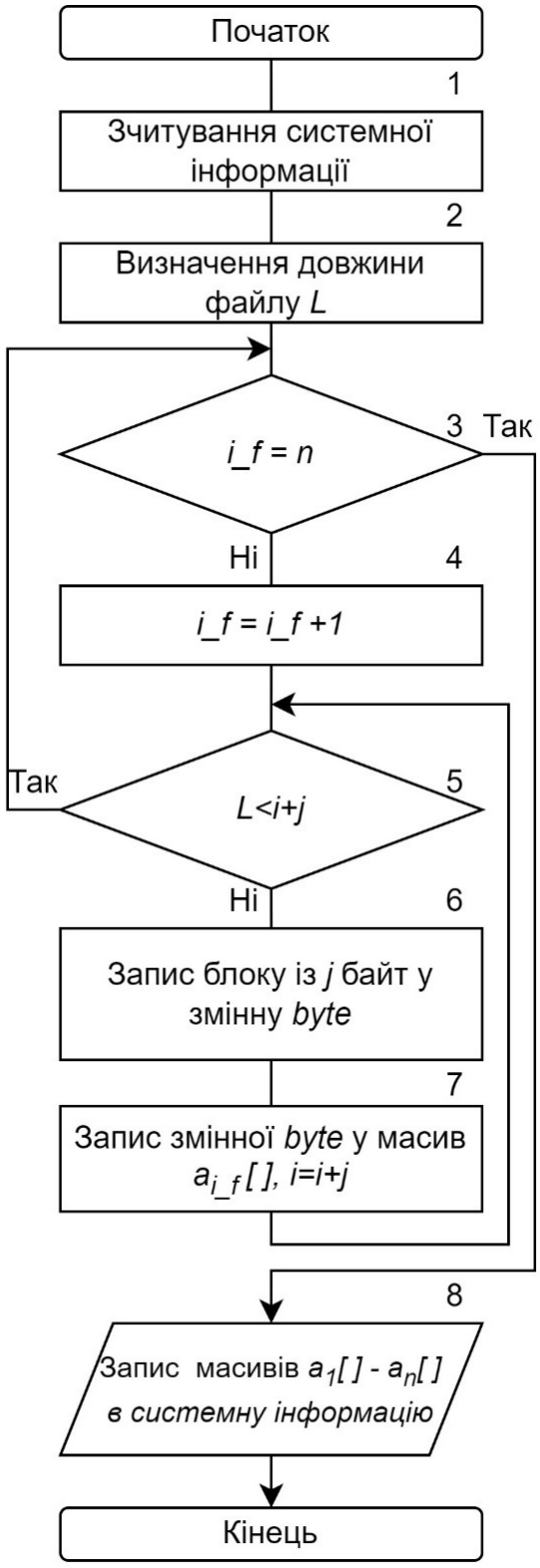

Рисунок 4.6 – Алгоритм зчитування файлів залишків

З системної інформації завантажуються дані про кількість масивів, які необхідно створити (блок 1), а визначивши довжину одного з файлів (блок 2),

можна починати наповнення масивів. Значення *n* відповідає за кількість файлів, які необхідно опрацювати, і є межею для кількості масивів (блок 3).

Оскільки масив $a_0\ [\,]$, як зазначалося раніше, використовується для позначення початкового масиву даних, то заповнення масивів почнеться з $a_1\ [\,]$ завдяки збільшенню значення лічильника на одиницю (блок 4).

Для перевірки досягнення кінця файлу залишків використовується відповідна перевірка (блок 5). Як зазначалося вище, змінна *j* відповідає за розмір блоків, що зчитуватимуться, і залежить від розрядності модулів, тому блоки з таким розміром будуть записуватися у тимчасову змінну byte (блок 6) і, в подальшому, будуть записані у відповідний масив, а лічильник *i* збільшиться на значення *j* (блок 7).

Після створення всіх необхідних масивів, умова «ТАК» блоку 3, створені масиви – $a_1\ [\,]$ - $a_n\ [\,]$, будуть записані в системну інформацію (блок 8), що і є результатом роботи алгоритму.

4.3. Алгоритми дешифрування та перевірки цілісності файлу

4.3.1. Алгоритм дешифрування залишків

Для отримання початкового файлу необхідно виконати зворотне перетворення за формулою (2.5), використовуючи файли залишків, імена яких збережені в масиві *Files [ ]*. Алгоритм дій у цьому випадку представлений на рисунку 4.7.

Як і в попередніх випадках, перш за все необхідно прочитати системну інформацію (блок 1) та визначити довжину масивів залишків (блок 2).

Оскільки в кожному файлі були записані залишки однакової розрядності, то розмір усіх файлів однаковий. Тому неважливо, який із файлів буде оброблятися першим, довжина *L* буде однаковою і визначатиме, скільки елементів масиву потрібно обробити (блок 3).

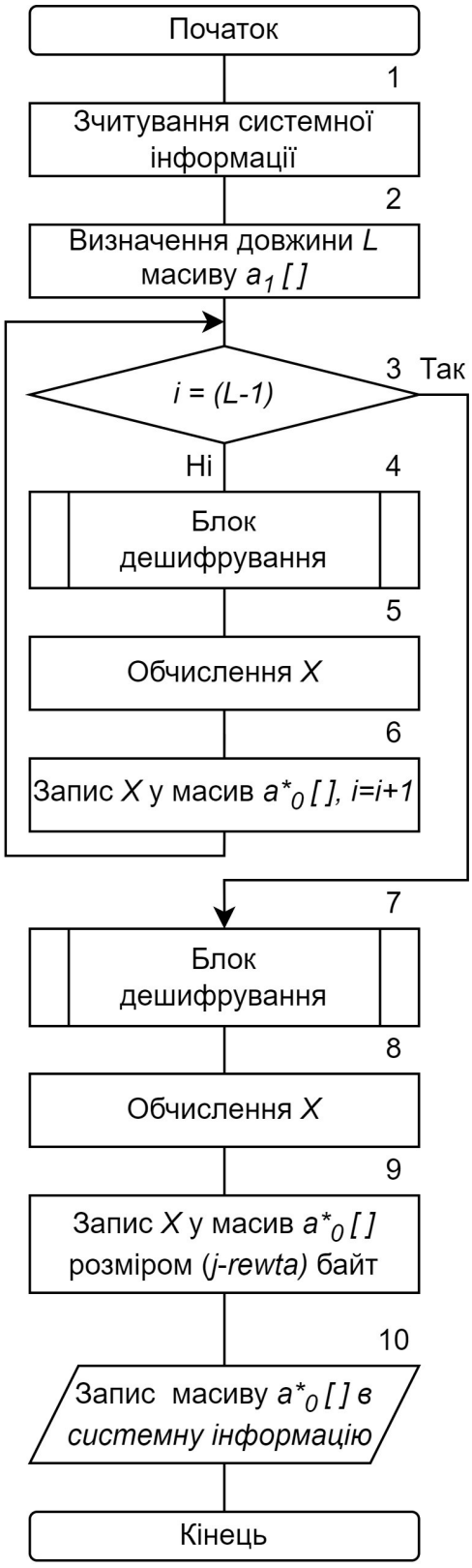

Рисунок 4.7 – Алгоритм дешифрування залишків

У відповідності із запропонованим методом шифрування (пункт 3.3), залишки в масивах цикілічно зсуваються згідно із обраною М-послідовністю, яка

використовується в якості ключа. У блоці дешифрування (блок 4) відбувається зворотне переміщення залишків на позиції, які вони займали до шифрування.

На наступному кроці (блок 5) відбувається обчислення значення X на основі формули зворотного перетворення. Значення X записується у масив $a^*_0[\,]$, а збільшивши лічильник i на одиницю, алгоритм перейде на наступний елемент масиву (блок 6). Така послідовність дій буде виконуватись до досягнення останнього елементу масиву (умова «ТАК» блоку 3).

При досягненні останнього елементу масиву необхідно, як і в попередньому випадку, розшифрувати послідовність залишків, тобто зробити зворотній зсув залишків (блок 7), та обчислити значення останнього обчисленого блоку X*[i]*.

Проте наступний крок (блок 9) матиме свої умови. Як зазначалося раніше, в залежності від розрядності обраних модулів система повинна зчитувати блоки відповідного розміру з файлів залишків.

У випадку, коли модулі мають розрядність один байт, всі блоки будуть однаковими і введення додаткових обчислень або змінних не потрібно.

Однак зчитування блоків розміром один байт є неефективним. Відповідно виникатиме ситуація, коли останній блок не буде кратний значенню змінної j.

При розділенні початкового файлу на блоки така проблема не враховується, оскільки в бітовому представленні додані на початку всіх розрядів нулі не змінять значення блоку у файлі залишків (таблиця 4.1).

Як видно з таблиці 4.1, блоки різної довжини можуть мати однакові значення в десятковому поданні і, як результат, будуть мати однакову кількість байтів у файлі залишків, ніяк при цьому не змінюючи його.

Таблиця 4.1

Залежність десяткового значення блоку від початкового двійкового представлення

Розрядність останнього блоку початкового файлу, байт	Представлення значення в двійковому форматі	Представлення значення в десятковому форматі	Розрядність останнього блоку файлу залишків, байт
1	00000010	2	1
2	00000000 00000010	2	1
3	00000000 00000000 00000010	2	1
4	00000000 00000000 00000000 00000010	2	1
5	00000000 00000000 00000000 00000000 00000010	2	1
6	00000000 00000000 00000000 00000000 00000000 00000010	2	1

Однак, під час зворотного перетворення важливо скільки байтів потрібно записати у файл при відновленні (таблиця 4.2).

Як бачимо з таблиці 4.2, під час зворотного перетворення автоматичне заповнення розрядів нулями змінює вміст файлу. Щоб уникнути такої ситуації здійснюється попередній запис розміру останнього блоку, а при зворотному перетворенні – перевірка його значення. У випадку, коли розмір початкового файлу був кратний *j*, то змінна *rewta* = 0. В іншому випадку значення останнього обчисленого блоку X*[i]* має бути записане не у вигляді *j*-байтового значення, а у вигляді (*j* - *rewta*) байтового значення (блок 9).

Таблиця 4.2

Залежність двійкового представлення значення від кінцевої розрядності блоку

Розрядність блоку у файлі залишків, байт	Значення в десятковому форматі	Розрядність останнього блоку при записі в початковий файл, байт	Значення блоку в двійковому представленні
1	2	1	00000010
1	2	2	00000000 00000010
1	2	3	00000000 00000000 00000010
1	2	4	00000000 00000000 00000000 00000010
1	2	5	00000000 00000000 00000000 00000000 00000010
1	2	6	00000000 00000000 00000000 00000000 00000000 00000010

Отриманий масив $a^*_0[\ ]$ записується у системну інформацію для подальших дій.

4.3.2. Алгоритм перевірки цілісності дешифрованого файлу

Масив $a^*_0[\ ]$, отриманий у результаті виконання попереднього алгоритму, містить відновлені із залишків дані, що повинні бути ідентичними вмісту початкового файлу. Для підтвердження цілісності та ідентичності цього вмісту необхідно створити новий файл, записати в нього масив та перевірити його ідентичність з початковим (рисунок 4.8).

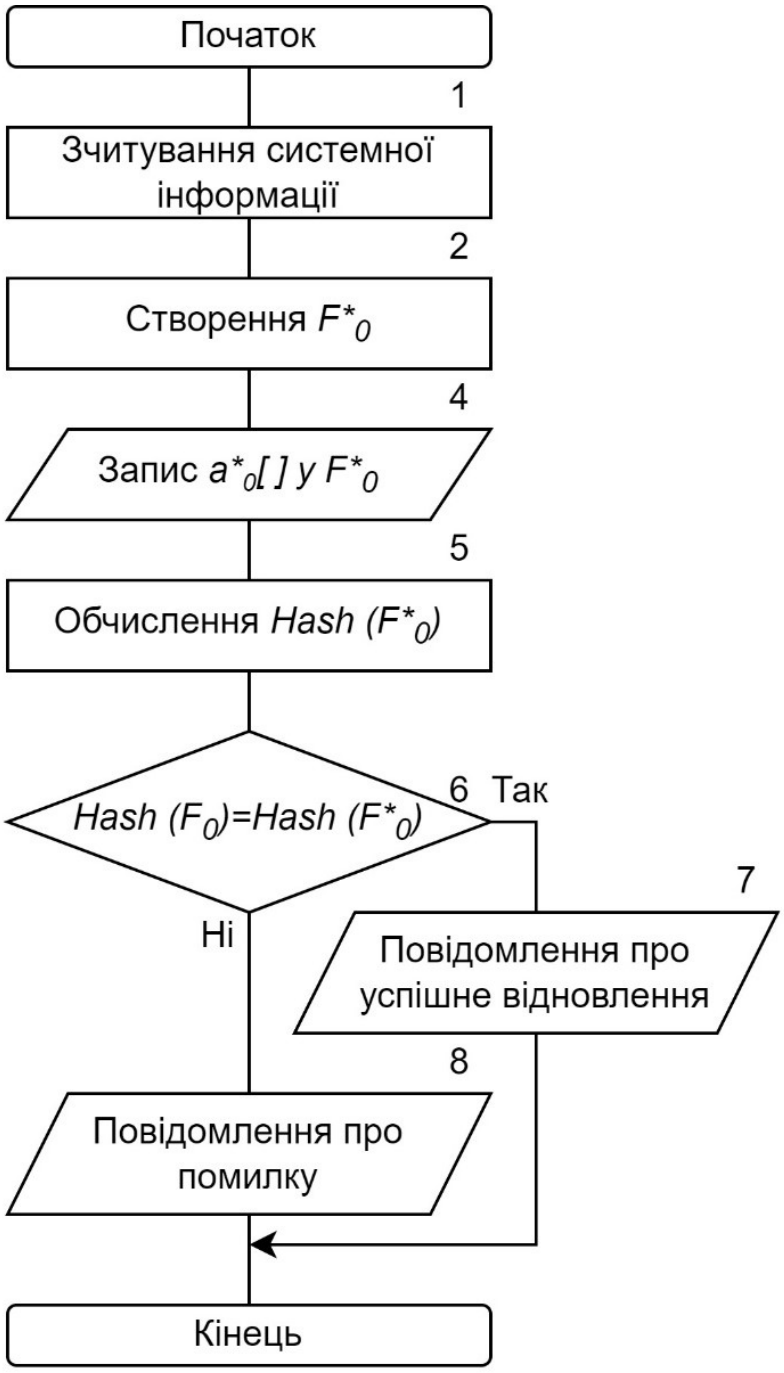

Рисунок 4.8 – Алгоритм перевірки цілісності дешифрованого файлу

Першим кроком для підтвердження цілісності відновленого файлу буде отримання доступу до масиву $a^*_0[\,]$ та геш-значення початкового файлу $Hash\ (F_0)$ (блок 1).

Наступним кроком буде створення нового файлу (блок 2) та запис у нього масиву (блок 3). Значення геш-функції, обчислюване від файлу, не залежить від дати створення файлу та від його назви, тому після створення та заповнення файл

має бути ідентичним початковому. Для підтвердження цього обчислюється геш-функція від новоствореного файлу (блок 5) та її порівняння з геш-значенням, збереженим при розділенні початкового файлу (блок 6).

В залежності від результату порівняння користувач отримає повідомлення про успішне (блок 7) або не успішне відновлення (блок 8).

4.4. Відновлення файлу за методом проекцій

4.4.1. Алгоритм відновлення при втраті одного із блоків даних

У випадку неможливості відновлення початкового файлу із залишків по інформаційних модулях застосовується розроблений підхід на основі методу проекцій та порівняння геш-функцій від файлів. Як відомо, СЗК з одним перевірочним модулем дозволяє відновити значення по (n-1) залишках. Для визначення не пошкоджених файлів використовується порівняння геш-значень.

Суть розробленого методу полягає в тому, що алгоритм відновлення визначає та фіксує позиції спотворених залишків та відновлює їх з використанням методу проекцій та порівняння геш-значення відновленого файлу із геш-значеням початкового.

У результаті даний метод дозволяє відновити файл при спотворенні двох і більше файлів залишків при певній комбінації помилок.

Розглянемо можливість відновлення пошкоджених даних на прикладі системи зберігання даних з трьома інформаційними та одним перевірочним модулем (таблиця 4.3).

Як видно з таблиці 4.3, при загальній кількості залишків $n = 4$ існує $v = 15$ варіантів пошкодження блоків у файлах залишків.

Таблиця 4.3

Варіанти пошкодження залишків у системі із n=4

№ з/п	1-й залишок	2-й залишок	3-й залишок	4-й залишок
1.	*	x_2	x_3	x_4
2.	x_1	*	x_3	x_4
3.	x_1	x_2	*	x_4
4.	x_1	x_2	x_3	*
5.	*	*	x_3	x_4
6.	*	x_2	*	x_4
7.	*	x_2	x_3	*
8.	x_1	*	*	x_4
9.	x_1	*	x_3	*
10.	x_1	x_2	*	*
11.	*	*	*	x_4
12.	*	*	x_3	*
13.	*	x_2	*	*
14.	x_1	*	*	*
15.	*	*	*	*

де x_1 - x_4 – правильний залишок, * – пошкоджений.

Відповідно, коли в масиві виникають дві помилки, то v=15^2, а загальна кількість помилок визначається як:

$$v=15^{er}, \tag{4.1}$$

де *er* – кількість помилок, які виникли в масиві даних.

Як зазначалося раніше (пункт 2.1.3), метод проекцій забезпечує відновлення початкового значення X при пошкодженні одного із залишків за умови використання двох перевірочних модулів.

У даній системі кількість перевірочних модулів $r=1$, а, отже, однозначно визначити, яка із проекцій правильна, використовуючи класичний метод проекцій, неможливо. Але завдяки пошуку всіх варіантів і порівнянню його геш-значення з геш-значенням початкового файлу можна знайти, яка проекція відповідає початковому значенню. Вводячи додаткові умови та відбираючи лише ті випадки, коли файл підлягає відновленню, можна значно скоротити кількість варіантів, які необхідно обчислити.

Запропоновано використовувати порівняння геш-значень файлів залишків, щоб відкинути частину варіантів. За умови, що геш-значення одного із файлів залишків відповідає геш-значенню, записаному у його назві, максимальна кількість пошкоджень з однією помилкою становить v=7 і не залежить від того, які файли залишків пошкоджено. Формула (4.1) набуде вигляду $v=7^{er}$, а варіанти пошкоджень приведені у таблиці 4.4.

Таблиця 4.4

Варіанти пошкодження залишків у системі з $n=4$ при підтвердженні цілісності четвертого залишку

№ з/п	1-й залишок	2-й залишок	3-й залишок	4-й залишок
1.	*	x_2	x_3	x_4
2.	x_1	*	x_3	x_4
3.	x_1	x_2	*	x_4
4.	*	*	x_3	x_4
5.	*	x_2	*	x_4
6.	x_1	*	*	x_4
7.	*	*	*	x_4

При умові, що геш-значення двох файлів залишків не співпадають, тобто пошкоджено два файли залишків з чотирьох, кількість варіантів пошкодження з однією помилкою в масиві даних становить $v=3$, а загальна кількість дорівнює $v=3^{er}$ (таблиця 4.5).

Таблиця 4.5

Варіанти пошкодження файлів залишків у системі із n=4 при підтвердженні цілісності другого та четвертого залишків

№ з/п	1-й залишок	2-й залишок	3-й залишок	4-й залишок
1.	X	x_2	x_3	x_4
2.	x_1	x_2	X	x_4
3.	X	x_2	X	x_4

Якщо на етапі вибору файлів залишків визначено, що пошкоджено більше, ніж один з них, а при відновленні обчислено, що вони відбулися в одному блоці, то виправити таке пошкодження неможливо. Отже, необхідно зазначити, що модуль відновлення можна використовувати при помилках в двох і більше блоках файлів залишків і всі обчислення будуть проводитись, виходячи з цього твердження.

У таблиці 4.6 наведено загальну кількість помилок, які можуть виникнути, та кількість, яку можна виправити запропонованим методом.

Таблиця 4.6

Залежність кількості помилок від кількості цілих файлів залишків

№ з/п	Кількість не пошкоджених файлів залишків	Загальна кількість помилок	Кількість помилок, які можна виправити
1.	0	15^{er}	4^{er}
2.	1	7^{er}	3^{er}
3.	2	3^{er}	2^{er}

Зазвичай при застосуванні методу проекцій шукане значення X знаходиться в діапазоні $0 \le X \le H$, а інші проекції X^* знаходяться в діапазоні $H \le X \le P$.

У випадку, коли використовується тільки один перевірочний модуль, значення проекцій не відповідає цим умовам, але одна з проекцій все одно буде очікуваним результатом. Після перегляду всіх варіантів і порівняння геш-значення вихідного файлу з геш-значенням відновленого, є певна ймовірність отримати початковий файл.

Як згадувалося раніше, алгоритм звертається до процедури відновлення лише тоді, коли пошкоджено більше одного файлу, тому існує ймовірність, що два файли із залишками були пошкоджені в одному блоці, що, в свою чергу, означає, що використання методу проекцій не матиме жодного ефекту. Варто також зауважити, що зі збільшенням кількості помилок у файлах залишків ефективність даного методу знижується. Залежність ефективності відновлення даних від кількості помилок при n=4 та пошкодженні усіх файлів залишків розраховано в таблиці 4.7.

Таблиця 4.7

Залежність ефективності відновлення файлу від кількості помилок при пошкодженні усіх файлів залишків

er	Загальна кількість помилок	Кількість помилок, які можна виправити	Ефективність, %
2	225	16	7,111
3	3375	64	1,896
4	50625	256	0,506
5	759375	1024	0,135
10	$5{,}77*10^{11}$	$1{,}05*10^{6}$	>0,001
20	$3{,}33*10^{23}$	$1{,}10*10^{12}$	>0,001
50	$6{,}38*10^{58}$	$1{,}27*10^{30}$	>0,001
100	$4{,}07*10^{117}$	$1{,}61*10^{60}$	>0,001
200	$1{,}65*10^{235}$	$2{,}58*10^{120}$	>0,001

Як видно із таблиці 4.7, ймовірність виправити чотири помилки при пошкодженні всіх файлів залишків менша 1%, а при 10 і більше – менша 0,001%.

Залежність ефективності модуля відновлення від кількості помилок при n=4 та трьох пошкоджених файлах залишків обчислено в таблиці 4.8.

Таблиця 4.8

Залежність ймовірності виправлення помилок від кількості помилок при трьох пошкоджених файлах залишків

er	Загальна кількість помилок	Кількість помилок, які можна виправити	Ефективність, %
2	49	9	18,367
3	343	27	7,872
4	2401	81	3,374
5	16807	243	1,446
10	$2,82*10^{8}$	59049	0,021
20	$7,98*10^{16}$	$3,49*10^{9}$	>0,001
50	$1,80*10^{42}$	$7,18*10^{23}$	>0,001
100	$3,23*10^{84}$	$5,15*10^{47}$	>0,001
200	$1,05*10^{169}$	$2,66*10^{95}$	>0,001

Як бачимо, при трьох пошкоджених файлах залишків ймовірність виправити дві помилки досить висока і становить 18%. Збільшення кількості помилок приводить до зменшення ефективності – ймовірність виправити 10 помилок становить 0,21%, а 20 і більше – менше 0,001%.

Залежність ефективності відновлення файлу від кількості помилок при n=4 та двох пошкоджених файлах залишків обчислено в таблиці 4.9.

У випадку, коли пошкоджено два з чотирьох файлів залишків (таблиця 4.9), ймовірність виправити дві помилки становить 44%.

Як і в попередніх розглянутих випадках, при збільшенні кількості помилок ймовірність їх виправлення зменшується, при цьому ймовірність виправлення 50 і більше помилок становить менше 0,001%.

Порівняння геш-значення відновленого та початкового файлів у кінці роботи модуля відновлення дозволяє уникнути помилкового повідомлення про успішне відновлення файлу після запису даних у кінцевий файл.

Таблиця 4.9

Залежність ймовірності виправлення помилок від кількості помилок при двох пошкоджених файлах залишків

er	Загальна кількість помилок	Кількість помилок, які можна виправити	Ефективність, %
2	9	4	44,444
3	27	8	29,630
4	81	16	19,753
5	243	32	13,169
10	59049	1024	1,734
20	$3{,}49*10^{9}$	$1{,}05*10^{6}$	0,030
50	$7{,}18*10^{23}$	$1{,}13*10^{15}$	>0,001
100	$5{,}15*10^{47}$	$1{,}27*10^{30}$	>0,001
200	$2{,}66*10^{95}$	$1{,}61*10^{60}$	>0,001

Якщо збережені та обчислені геш-значення збігаються, то можна зробити висновок, що файли залишків були пошкоджені в різних блоках і процес відновлення пройшов успішно. В іншому випадку відновлення даних не відбулося, про що користувач буде сповіщений відповідним повідомленням.

Алгоритм зворотного перетворення із використанням удосконаленого методу проекцій представлено на рисунку 4.9.

Перший крок алгоритму відновлення полягає у зчитуванні системної інформації (блок 1), такої як набір модулів, базові числа, коефіцієнти та імена

файлів залишків. На наступному кроці визначається довжина першого із файлів залишків (блок 2). Інші файли залишків, як вже зазначалося, будуть містити таку саму кількість блоків.

Першим завданням при відновленні даних є визначення кількості пошкоджених блоків. Відомо, що при пошкодженні одного із залишків шукане значення X вийде за межі робочого діапазону, тобто $H \le X \le P$.

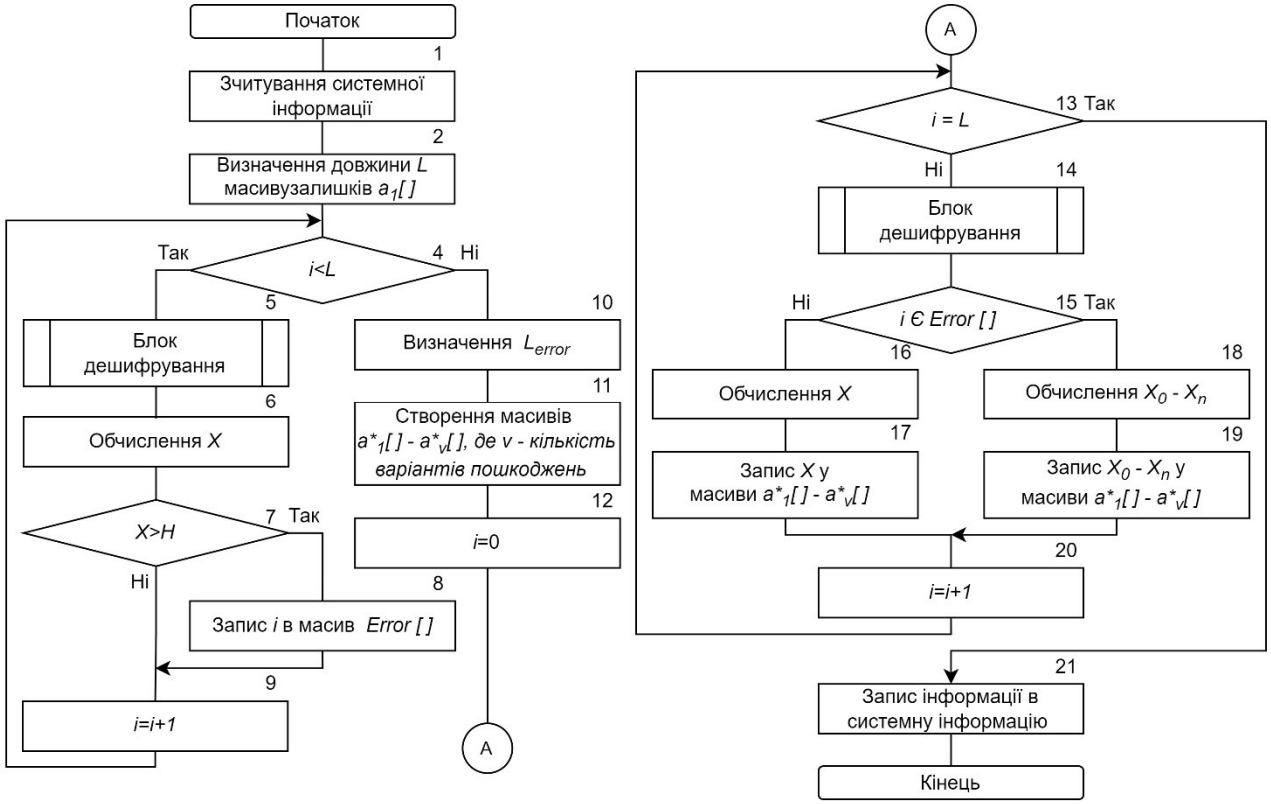

Рисунок 4.9 – Алгоритм відновлення у випадку пошкодження двох і більше файлів залишків

Тому для визначення кількості пошкоджених блоків необхідно для кожного із них виконати дешифрування залишків (блок 5). Далі, обчисливши значення X (блок 6), порівняти його із значенням робочого діапазону H (блок 7). Записавши в окремий масив *Error [ ]* (блок 8) всі номери пошкоджених блоків, отримаємо масив помилок, який у подальшому використовується для їх виправлення. Визначивши довжину масиву *Error [ ]* (блок 10), отримаємо загальну кількість помилок, а використовуючи масив *Files [ ]*, який містить імена непошкоджених файлів залишків та формули з таблиці 4.6, обчислимо кількість

масивів, які потрібно створити (блок 11). Після скидання лічильника *i* (блок 12) переходимо до наступного етапу. Блоки з файлів залишків необхідно прочитати повторно (блок 13) і дешифрувати залишки для кожного з них (блок 14).

У випадку, коли блок непошкоджений (умова «НІ» блоку 15), необхідно виконати операцію зворотного перетворення (блок 16) і записати отримане значення у всі створені масиви (блок 17).

В іншому випадку, коли блок пошкоджений (умова «ТАК» блоку 15), необхідно обчислити значення всіх проекцій (блок 18) і записати їх у певному порядку в новостворені масиви (блок 19). Переходячи до наступного кроку (блок 20), виконуємо задану послідовність дій, поки не дійдемо до останнього блоку (умова «ТАК» блоку 13).

Відповідно, всі отримані значення та дані необхідно зберегти в системній інформації (блок 21).

У результаті проведених операцій буде сформовано *v* масивів, один з яких, за умови відсутності пошкодження двох і більше залишкових файлів в одному блоці, є шуканим.

4.4.2. Алгоритм запису файлу відновленого за методом проекцій

У подальшому необхідно перевірити чи один із отриманих в результаті попереднього кроку масивів відповідає вмісту початкового файлу. Алгоритм запису відновленого файлу згідно методу проекцій, представлено на рисунку 4.10.

Зчитавши інформацію про сформовані масиви та геш-значення початкового файлу (блок 1) необхідно створити файл, у який по черзі записуються масиви (блок 2).

Сформовані масиви починаються з $a_1[\ ]$, тому лічильник збільшується зразу (блок 3). Починаючи з $a_1[\ ]$ і до $a_v[\ ]$ (блок 4) необхідно здійснити почергово запис у файл (блок 5), обчислення геш-функції від нього (блок 6) та порівняння з геш-значенням початкового файлу (блок 7).

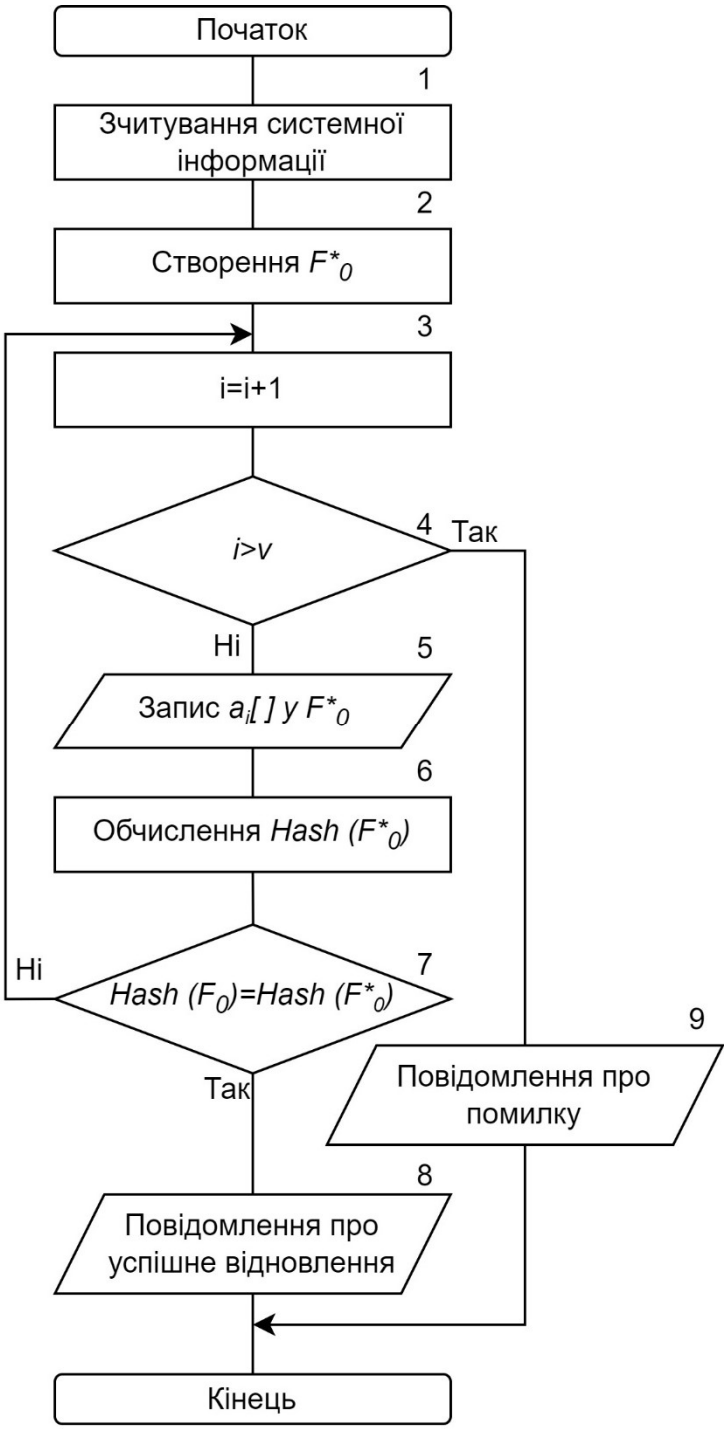

Рисунок 4.10 – Алгоритм запису відновленого файлу за методом проекцій

При умові, що геш-значення однакові (умова «ТАК» блоку 7), то відновлення файлу (блок 8) пройшло успішно.

Якщо ж послідовна перевірка геш-значень для кожного з v масивів не дала результату (умова «НІ» блоку 7) і був перевірений останній масив (умова «ТАК» блоку 4), то користувач отримає повідомлення про помилку (блок 9).

Розглянемо приклад формування таких масивів за умови, що при пошкодженні двох файлів залишків виникає три помилки. Відповідно до таблиці 4.6, загальна кількість можливих варіантів помилок становитиме $3^3=27$, а кількість помилок, які можна виправити – $2^3=8$.

Варто зауважити, що вісім варіантів, які можна виправити, включають обидва випадки, які вже були відкинуті, тобто коли три помилки знаходяться в одному із файлів залишків. Отже, необхідно створити 6 масивів і обчислити значення проекцій для них згідно таблиці 4.10.

Таблиця 4.10

Варіант виникнення трьох помилок у системі із n=4 при підтвердженні цілісності третього та четвертого залишків

№ з/п	1-й залиш.	2-й залиш.	3-й залиш.	4-й залиш.		№ з/п	1-й залиш.	2-й залиш.	3-й залиш.	4-й залиш.
1	***2***	***3***	***4***	***5***		***1***	***2***	***3***	***4***	***5***
1 варіант						4 варіант				
1.	…	…	…	…		1.	…	…	…	…
2.	*	x_2	x_3	x_4		2.	x_1	*	x_3	x_4
3.	…	…	…	…		3.	…	…	…	…
4.	*	x_2	x_3	x_4		4.	x_1	*	x_3	x_4
5.	…	…	…	…		5.	…	…	…	…
6.	x_1	*	x_3	x_4		6.	*	x_2	x_3	x_4
2 варіант						5 варіант				
1.	…	…	…	…		1.	…	…	…	…
2.	*	x_2	x_3	x_4		2.	x_1	*	x_3	x_4
3.	…	…	…	…		3.	…	…	…	…
4.	x_1	*	x_3	x_4		4.	*	x_2	x_3	x_4
5.	…	…	…	…		5.	…	…	…	…
6.	*	x_2	x_3	x_4		6.	x_1	*	x_3	x_4

Прод. таблиці 4.10

1	*2*	*3*	*4*	*5*		*1*	*2*	*3*	*4*	*5*
3 варіант						6 варіант				
1.	…	…	…	…		1.	…	…	…	…
2.	x_1	*	x_3	x_4		2.	*	x_2	x_3	x_4
3.	…	…	…	…		3.	…	…	…	…
4.	*	x_2	x_3	x_4		4.	x_1	*	x_3	x_4
5.	…	…	…	…		5.	…	…	…	…
6.	*	x_2	x_3	x_4		6.	x_1	*	x_3	x_4

У результаті проведених обчислень для наведеного прикладу сформовано 6 масивів, один з яких, за умови відсутності пошкодження двох і більше залишків в одному блоці, є шуканим.

Висновки до розділу 4

На основі приведеного у пункті 3.3 методу шифрування даних в надлишковій СЗК з використанням М-послідовності та методу кодування/декодування, розробленому у пункті (2.2), побудовано алгоритми захищеного кодування даних на основі надлишкової СЗК.

Приведено алгоритми шифрування даних на основі надлишкової СЗК та М-послідовності, що включають наступні кроки:

– переведення початкового файлу в масив блоків;

– обчислення залишків за вибраними наборами модулів та їх шифрування;

– запис масивів залишків у файли залишків та обчислення параметрів, що забезпечують цілісність файлів, а саме геш-значення та контрольної суми з використанням циклічного надлишкового коду *CRC-32.*

Приведено перевірку цілісності файлів залишків, що включає в себе:

– зчитування файлів залишків та обчислення геш-функції;

– перевірку цілісності файлів залишків;

– зчитування файлів залишків для зворотного перетворення.

Для зазначених пунктів побудовано алгоритми та детально описано їх застосування.

Побудовано алгоритми дешифрування та перевірки цілісності файлів залишків та описано необхідні додаткові умови (таблиця 4.1-4.2) зберігання даних на фізичних носіях.

Описано відновлення даних згідно запропонованого методу проекцій, для чого побудовано алгоритми відновлення при втраті одного із блоків даних та запису відновленого файлу.

Описано варіанти пошкодження залишків у системі з $n = 4$, на основі яких виведено математичні формули обчислення кількості помилок в залежності від кількості файлів залишків із підтвердженою цілісністю. Визначено залежність ефективності відновлення файлу від кількості помилок при різній кількості пошкоджених файлів залишків. Наведено приклад застосування методу відновлення при виникненні трьох помилок у системі із $n = 4$ та при підтвердженні цілісності двох із чотирьох залишків.

РОЗДІЛ 5. ПРИКЛАДНА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ДЛЯ РОЗПОДІЛЕНОГО ВІДДАЛЕНОГО ЗБЕРІГАННЯ ДАНИХ

5.1. Архітектура системи розподіленого захищеного зберігання даних

На основі наведених у четвертому розділі алгоритмів шифрування залишків, перевірки цілісності файлів залишків, їх дешифрування, перевірки дешифрованого файлу та алгоритмів відновлення пошкодженого файлу розроблено програмне забезпечення СРЗЗД.

Запропонована система зберігання даних складається із шести модулів, а залишки записуються на відповідні ПЗД (рисунок 5.1).

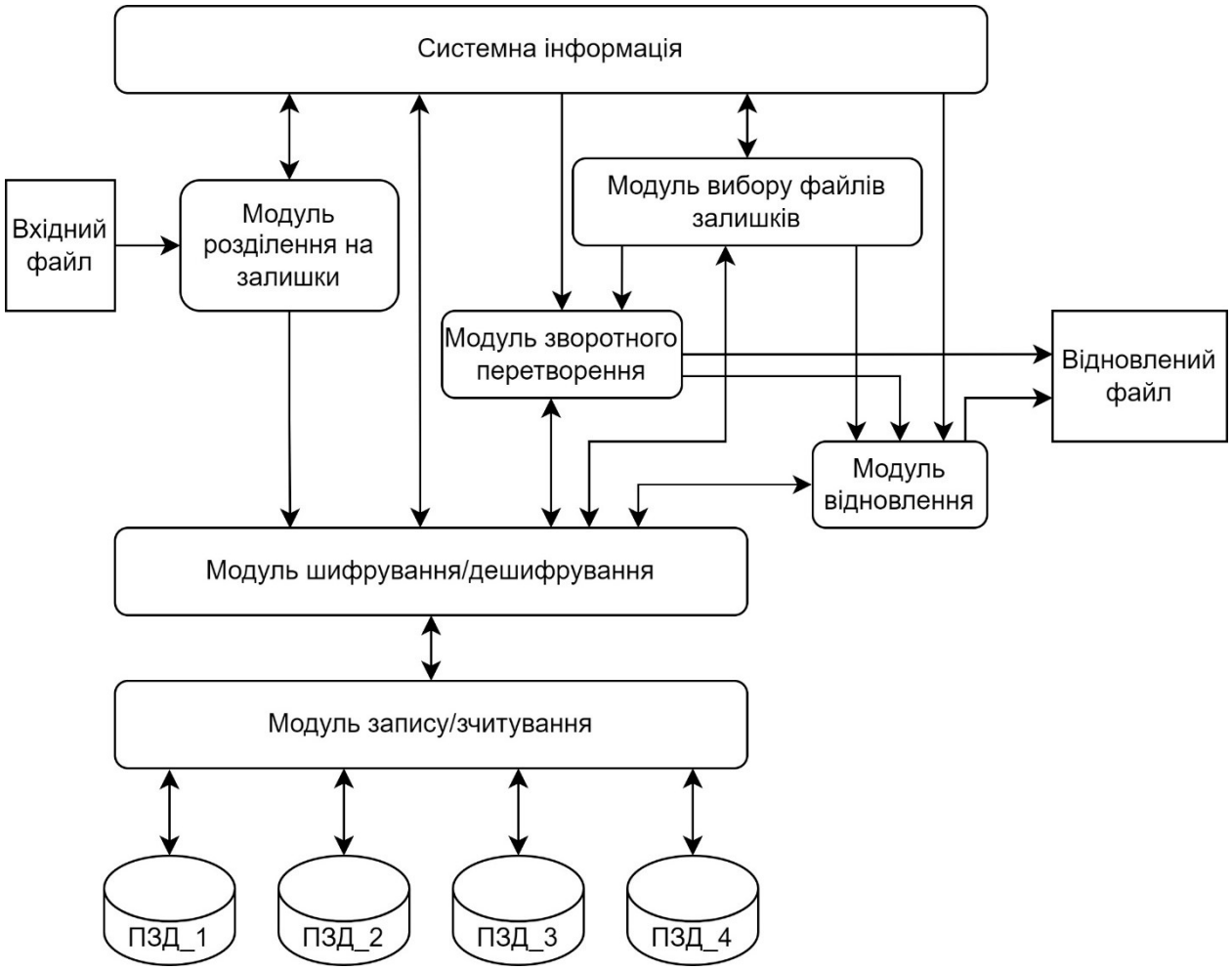

Рисунок 5.1 – Функціональна схема системи розподіленого захищеного зберігання даних

Для забезпечення функціональності системи згідно поставленого завдання необхідно обрати кількість та значення модулів, сформувати зв'язки між компонентами системи – модулями та ввести необхідні додаткові компоненти.

Схема роботи СРЗЗД включає в себе такі модулі як: розділення на залишки, шифрування/дешифрування, запису/зчитування, вибору файлів залишків, зворотного перетворення та відновлення.

Модуль розділення на залишки використовується для розділення вхідного файлу на масиви залишків (рисунок 5.2).

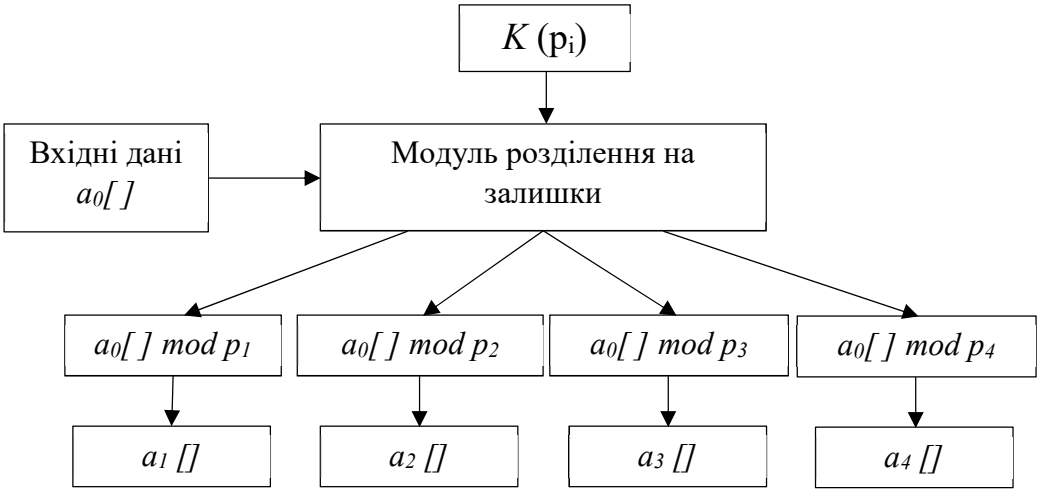

Рисунок 5.2 – Функціональна схема розподілу вхідного масиву даних на масиви залишків

Початковий файл перед розподілом на файли залишків необхідно перетворити в масив даних $a_0[\,]$, з яким надалі і проводяться всі операції.

В залежності від обраного набору модулів вхідний файл зчитується блоками, розмір яких залежить від робочого діапазону системи. Для спрощення роботи зазначені блоки записуються в масив $a_0[\,]$, що дозволяє на наступному кроці зразу обчислювати залишок від ділення на кожен із модулів. Результатом роботи модуля розділення на залишки є чотири масиви $a_1[\,] - a_4[\,]$, в кожному з яких міститься послідовність залишків обчислених від двійкового значення вмісту початкового файлу.

Для перетворення вхідного файлу в масив $a_0[\ ]$ та масивів $a_1[\ ] – a_4[\ ]$ у файли залишків використовується окремий елемент – модуль запису/зчитування (рисунок 5.3).

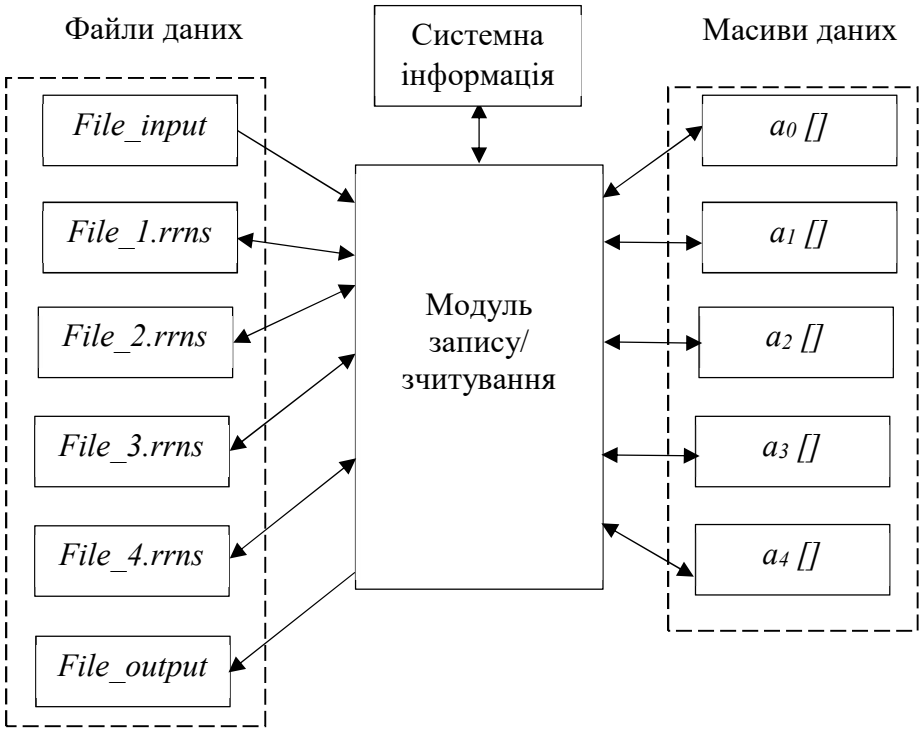

Рисунок 5.3 – Функціональна схема модуля запису/зчитування

Окрім зберігання даних, даний модуль виконує запис службової інформації у файли залишків, обчислення значення геш-функції та CRC-32 від файлів, а також їх запис у системну інформацію та файли залишків.

Модуль запису/зчитування служить буфером обміну та забезпечує зв’язок інших модулів із підключеними ПЗД, наприклад, дозволяє записати сформовані файли залишків на відповідні ПЗД. Окрім того, цей модуль після формування файлів залишків обчислює значення геш-функції та записує його в назву файлу для подальшого визначення будь-яких механічних пошкоджень їх вмісту на пристроях ПЗД. Для перевірки геш-значеннь на спотворення застосовується додатковий крок захисту, а саме обчислення контрольної суми CRC-32. Після додавання (конкатенації) значення CRC-32 в назви відповідних файлів залишків розділення даних можна вважати завершеним.

Використання надлишкової СЗК як основи побудови системи зберігання даних дозволяє отримати початковий файл даних, використавши тільки 3 із 4 збережених файлів залишків.

На рисунку 5.4 представлено функціональну схему модуля вибору файлів залишків, за допомогою якого обирається, які модулі використовуватимуться для відновлення даних.

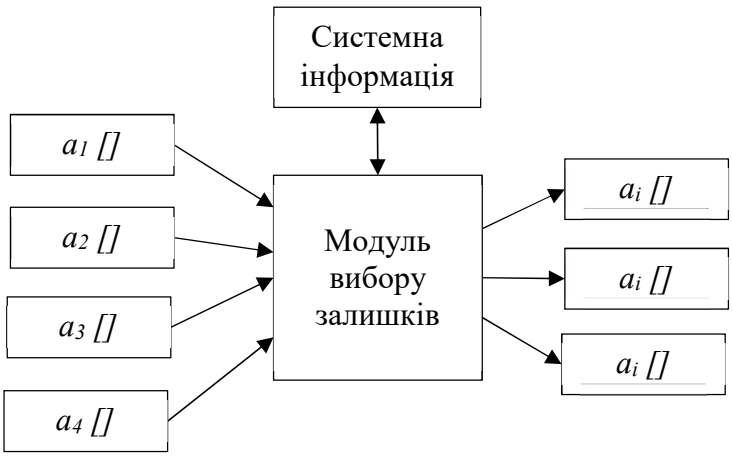

Рисунок 5.4 – Функціональна схема модулю вибору файлів залишків

Вибір модулів здійснюється шляхом порівняння геш-значень кожного із файлів залишків із геш-значеннями, записаними у назві файлу при збереженні. Модуль вибору файлів залишків визначає їх доступність в цілому, а згодом забезпечує послідовне порівняння. Якщо доступні перші три файли залишків і їх геш-значення збігаються, то цього достатньо для відновлення початкового файлу. У випадку, якщо один із файлів пошкоджений або недоступний, то перевіряється наявність і цілісність наступного файлу залишків. У випадку, якщо він також пошкоджений, користувач отримає повідомлення про необхідність запуску додаткової процедури відновлення даних.

У разі, якщо перевірка наявності та цілісності будь-яких трьох файлів залишків мала позитивний результат, то дані поступають на модуль зворотного перетворення (рисунок 5.5).

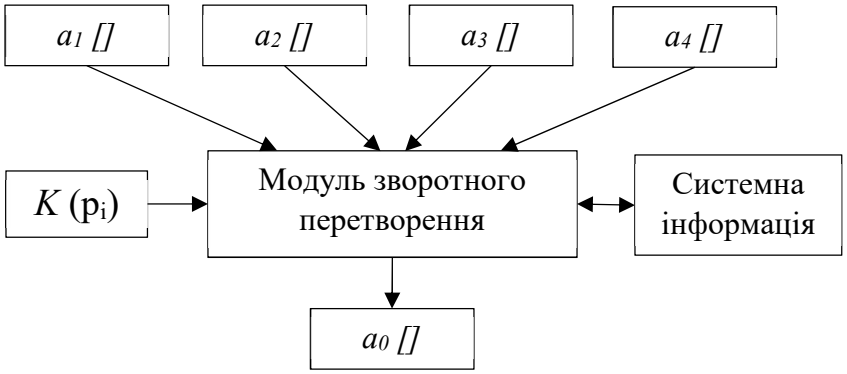

Рисунок 5.5 – Функціональна схема модулю зворотного перетворення

Модуль зворотного перетворення послідовно зчитує елементи трьох із чотирьох доступних масивів $a_1[\ ] - a_4[\ ]$ та за допомогою формули зворотного перетворення СЗК (2.5) обчислює початкове значення X та записує його в масив $a_0[\ ]$.

В кінці роботи модуля зворотного перетворення масив $a_0[\ ]$ записується у файл із використанням модуля запису/зчитування, а отриманий файл після порівняння із геш-значенням початкового файлу, збереженим у файлі сесії, можна вважати ідентичним початковому.

У випадку, коли порівняння значення геш-функції від отриманого і початкового файлу дало негативний результат або модуль вибору файлів залишків визначив, що відбулося пошкодження цілісності більше одного файлу залишків, то система спробує відновити початковий файл, використовуючи модуль відновлення, функціональна схема якого представлена на рисунку 5.6.

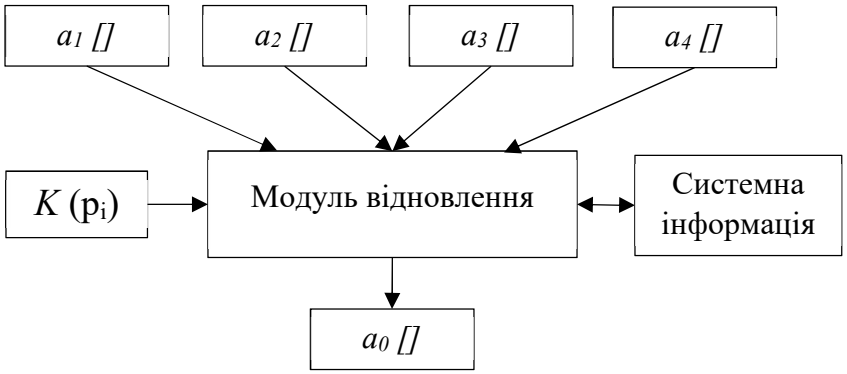

Рисунок 5.6 – Функціональна схема модуля відновлення

Робота модуля відновлення базується на принципі роботи модуля зворотного перетворення з наступними відмінностями. Якщо модуль зворотного перетворення обробляє тільки три масиви, то модуль відновлення обробляє всі чотири масиви. Як вже згадувалось, при представленні методу відновлення (пункт 4.3), для початку, використовуючи рівність $R \leq X \leq M$, знаходиться загальна кількість пошкоджених блоків відновлювального файлу та створюється необхідна кількість масивів для запису всіх можливих варіантів виникнення помилки. Якщо пошкодження не відбулося, то у всі масиви записуються однакові значення. У випадку, якщо пошкодження мало місце, то обчислюються проекції у блоці відновлювального файлу та записуються в масиви почергово.

Як вже зазначалося, модуль зчитування/запису проводить обчислення значення геш-функції від файлів та запис цієї інформації в системну інформацію, тому саме порівняння геш-значеннь кожного із створених масивів та геш-значення, яке було записане заздалегідь, дає змогу підтвердити успішне відновлення початкового файлу.

5.2. Програмна реалізація системи

У рамках виконання дисертаційного дослідження реалізовано програмне забезпечення на основі розробленої СЗРЗД. Даний продукт базується на використанні удосконаленого методу проекцій СЗК. Проектована система була реалізована на мові програмування Python.

5.2.1. Інтерфейс користувача

Дизайн основного вікна програми виконано відповідно до стандартних налаштувань операційної системи Windows та зображено на рисунку 5.7.

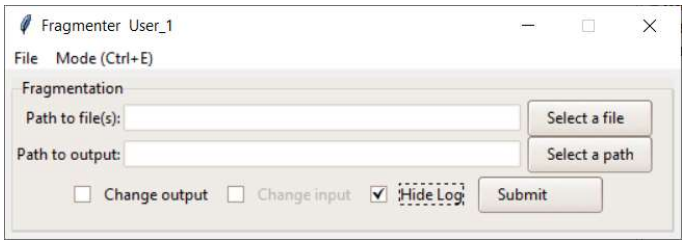

Рисунок 5.7 – Основне вікно програми

При запуску програми перший раз можна зразу розпочати роботу. Також кожен із користувачів, які працювали раніше, можуть підключити свою, попередньо збережену, сесію. Для цього необхідно вибрати меню «File >> Open» (рисунок 5.8).

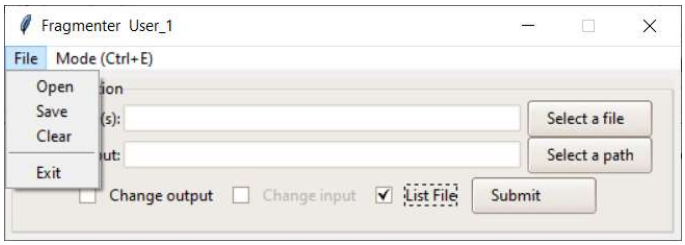

Рисунок 5.8 – Випадаюче меню File

Ідентифікація користувачів виконується по збереженому файлі сесії та паролю. Файл сесії може містити довільну назву користувача та автоматично при збереженні отримає розширення *.ssn (рисунок 5.9).

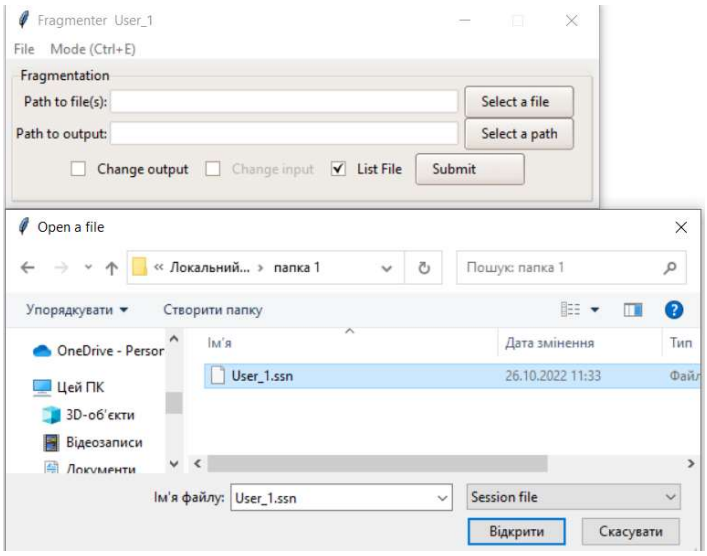

Рисунок 5.9 – Відкриття файлу сесії

Для підтвердження автентичності кожного із користувачів потрібно ввести пароль (рисунок 5.10), який кодується за допомогою модуля *hashlib.*

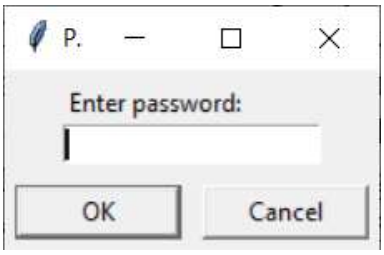

Рисунок 5.10 – Підтвердження прав користувача

Цей модуль реалізує загальний інтерфейс для багатьох різних безпечних алгоритмів гешування. Він включає в себе безпечні згідно стандартів обробки інформації геш-алгоритми SHA1, SHA224, SHA256, SHA384 і SHA512, а також алгоритм RSA MD5, визначений в стандарті RFC 1321 [122].

В нашому випадку використовується алгоритм SHA256, а саме оголошення процедури кодування матиме вигляд:

```
def hash_sha256(data: str) -> bytes:
    h = hashlib.sha256()
    h.update(data.encode())
    return h.digest()
```

Використання такого способу захисту паролю унеможливлює його перехоплення за допомогою сторонніх програм, що забезпечує додатковий захист збережених файлів.

Під час роботи з програмним продуктом основне вікно розділено на два функціональних блоки, один із яких можна приховати за допомогою відповідної кнопки. При успішному вході користувача за допомогою файлу сесії у такому вікні буде відображатися список файлів, з якими працювалося раніше «List file». На рисунку 5.11 можна переглянути вікно з такими файлами.

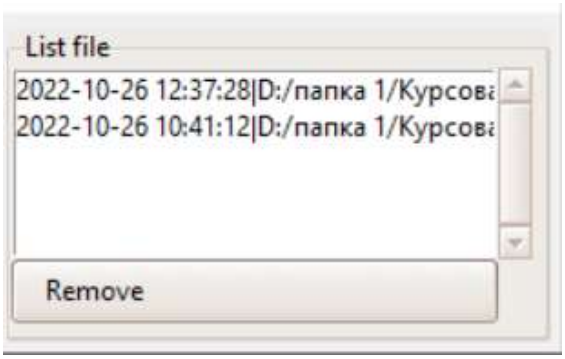

Рисунок 5.11 – Вікно списку файлів

Користувач може переглянути весь список файлів, з якими він працював від початку збереженої сесії, та, при потребі, видалити якийсь файл із даного списку.

Перед закриттям програми потрібно зберегти результати роботи. Для цього у меню «File» обирається пункт «Save». У результаті такої дії запропонується обрати місце для збереження файлу та його назву (рисунок 5.12).

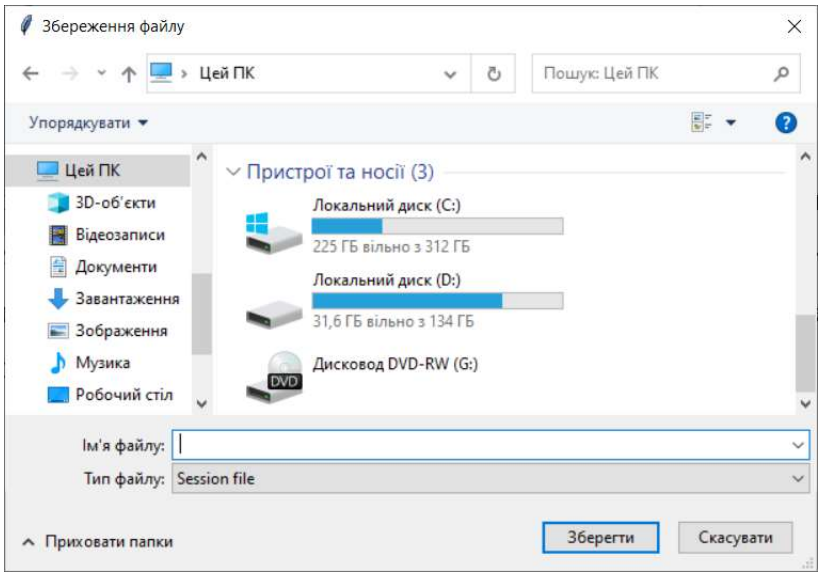

Рисунок 5.12 – Зберігання файлу сесії

Як видно із рисунку 5.12, розширення збереженого файлу сесії буде добавлено автоматично.

У меню «File», окрім уже згадуваних пунктів «Open» та «Save», також присутні стандартний пункт для виходу із програми «Exit» та додатковий пункт

«Clear», який дозволяє повністю очистити список файлів, із якими працював даний користувач, тобто видалити всю історію роботи.

При цьому, як і у випадку одиночного видалення файлів із списку кнопкою «Remove», видаляється лише запис про файл, файли залишків на віддалених носіях залишаються без змін.

5.2.2. Режими роботи програмного засобу

Розроблений продукт має два режими роботи:

– для розподіленого зберігання;

– для відновлення даних із файлів залишків.

В режимі розподіленого зберігання даних (рисунок 5.7) програма налаштована на розділення файлів на файли залишків та віддалене їх зберігання.

При виконанні таких операцій потрібно у відповідному вікні вибрати папку, в яку будуть зберігатися файли залишків (рисунок 5.13).

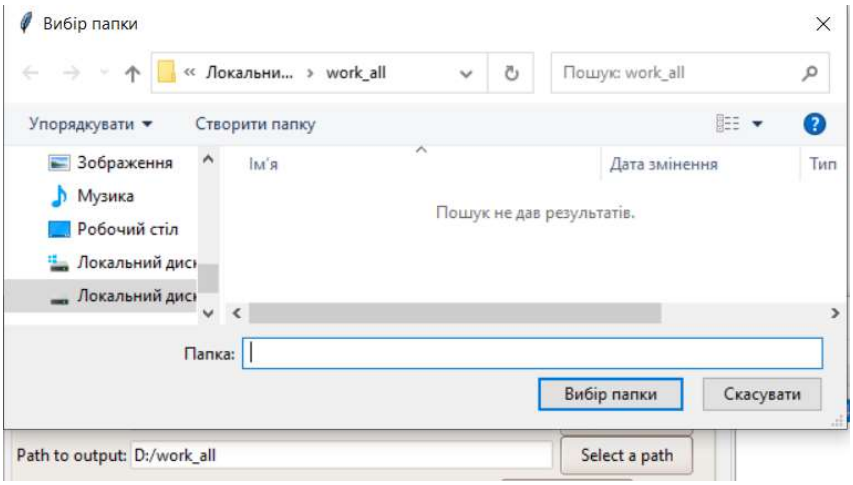

Рисунок 5.13 – Вікно вибору папки для файлів залишків

Як же згадувалося раніше, існує можливість підключити від одного до чотирьох пристроїв зберігання. При виборі більше однієї папки для зберігання потрібно натиснути відповідну радіокнопку «Different outputs».

В результаті такого кроку основне вікно програмного продукту зміниться і дозволить вибрати адресу зберігання для кожного із файлів залишків (рисунок 5.14).

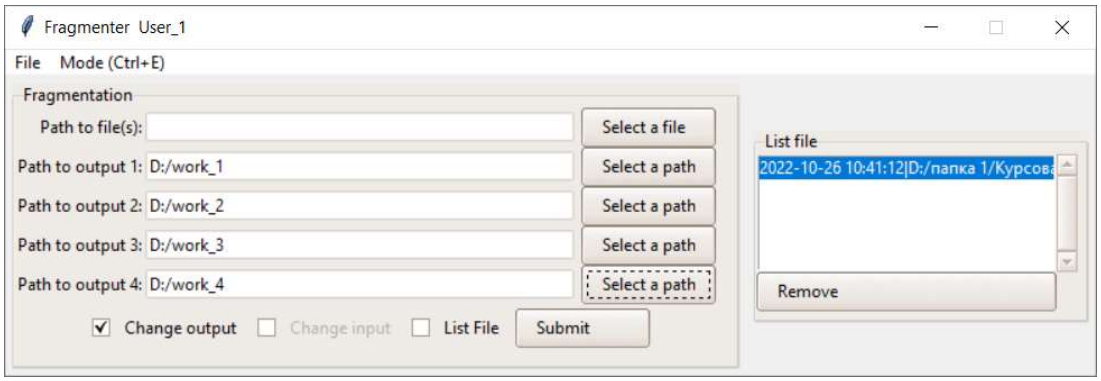

Рисунок 5.14 – Вікно вибору різних папок для кожного із файлів залишків

Коли ми говоримо про вибір різних адрес для файлів залишків, то мається на увазі, що це можуть бути: USB-флеш-накопичувачі; будь-які локальні накопичувачі (SSD, HDD); віддалені локальні диски, підключені через налаштування операційної системи.

Сучасні хмарні сервіси зберігання даних, наприклад Google Диск та Dropbox, мають дистрибутиви для синхронізації локальної папки з хмарним сховищем, завдяки чому, підключивши одночасно кілька середовищ, навіть без використання механізму API можливо налаштувати зберігання даних у хмару [124].

Після налаштування шляхів для збереження файлів залишків та авторизації користувача стає доступним вибір файлу, який необхідно розділити на залишки.

Така можливість забезпечується стандартним діалоговим вікном операційної системи після натискання на кнопку вибору файлу «Select a file» або введення адреси та назви файлу в ручному режимі (рисунок 5.15).

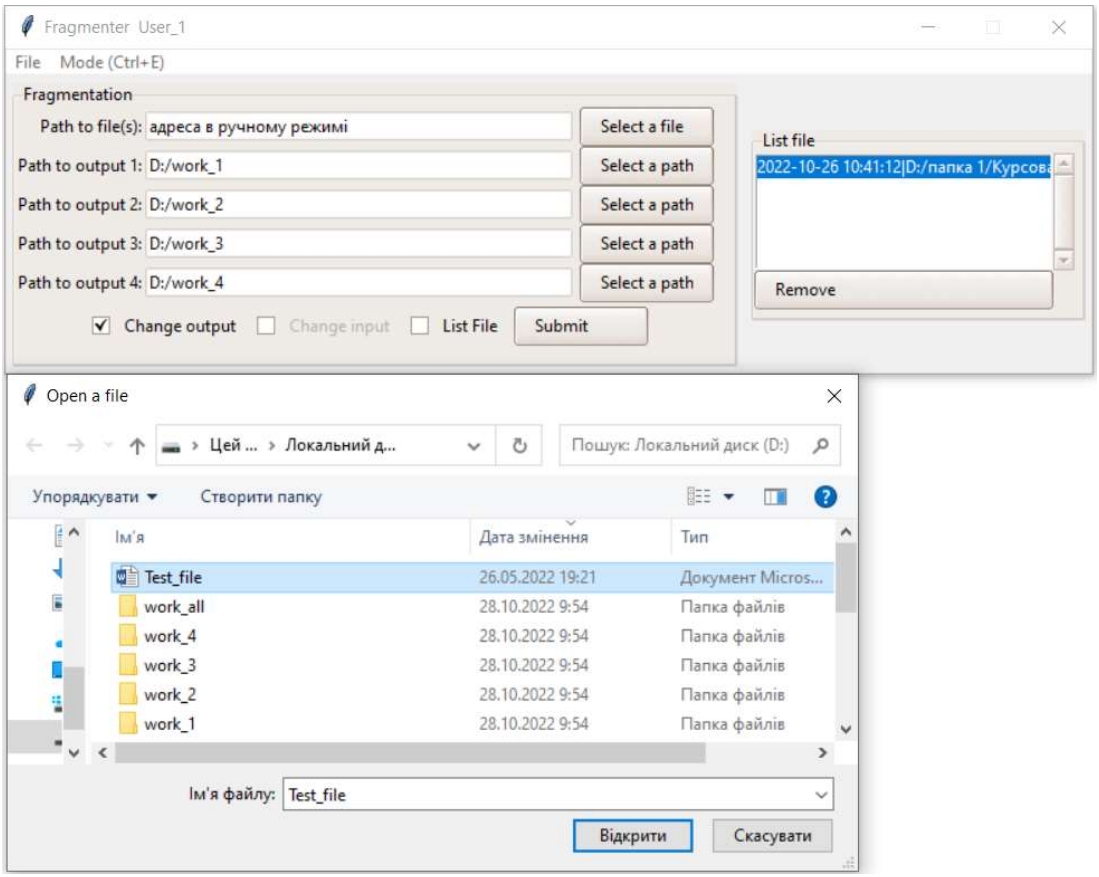

Рисунок 5.15 – Вікно вибору файлу для розділення на залишки

Коли всі попередні кроки виконано, то можна виконати розділення файлу на залишки, для чого необхідно натиснути на кнопку «Submit». Про успішне виконання розділення файлу на залишки користувач отримає відповідне повідомлення (рисунок 5.16).

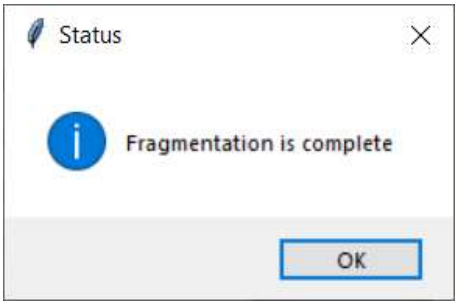

Рисунок 5.16 – Успішне розділення файлу на залишки

Час очікування повідомлення про успішне розділення файлу на залишки залежить від багатьох чинників, основними із яких є, звичайно ж, розмір файлу, тип обраних пристроїв та віддаленість до них.

Після успішного розділення файлу на залишки у списку «List file» буде відображатись назва файлу, який розділили. У випадку, коли користувач не обрав один із шляхів для зберігання файлів залишків, то такий файл буде зберігатися в першу обрану папку за замовчуванням.

Як вже не раз відмічалося, для успішного відновлення файлу при втраті доступу чи пошкодженні одного з носіїв, кожен із файлів залишків необхідно зберігати на окремому розподіленому пристрої. Тільки у такому випадку, маючи три із чотирьох файлів залишків, можливо відновити файл.

5.3. Реалізація методу відновлення даних із блоків

Окрім розділення файлів на залишки, необхідно також відновлювати початкові файли. Для виконання цього кроку потрібно змінити режим роботи «Mode», що забезпечується відповідним пунктом меню. За замовчуванням програмний продукт відкривається у режимі розділення на файли залишків (рисунок 5.17).

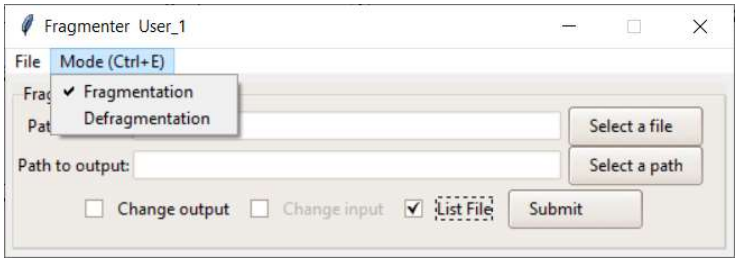

Рисунок 5.17 – Режим розділення на залишки

Використовуючи режим відновлення, можна отримати назад початковий вигляд наших файлів. Після зміни режиму роботи основна частина вікна змінить свій функціонал та матиме вигляд, який можна побачити на рисунку 5.18.

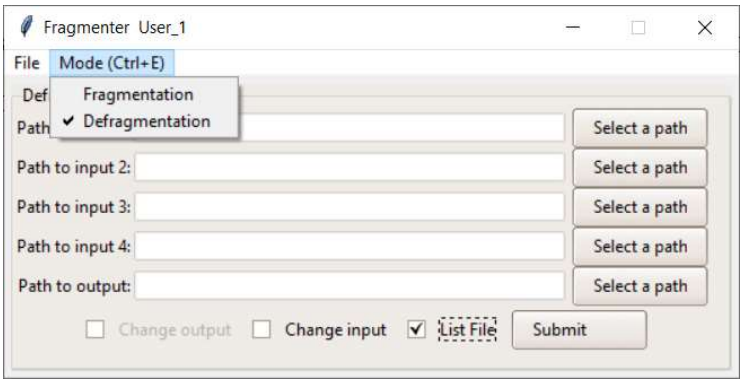

Рисунок 5.18 – Режим відновлення

У режимі відновлення для роботи необхідно у відповідному вікні обрати шлях для зберігання відновлених файлів «Path to output». Як і у випадку із шляхом для збереження файлів залишків, шлях для зберігання відновленого файлу можна написати вручну (рисунок 5.19).

Наступний крок буде залежати від того, чи є необхідний нам файл у переліку «List file», тобто чи даний користувач розділяв файл, який потрібно відновити, на залишки.

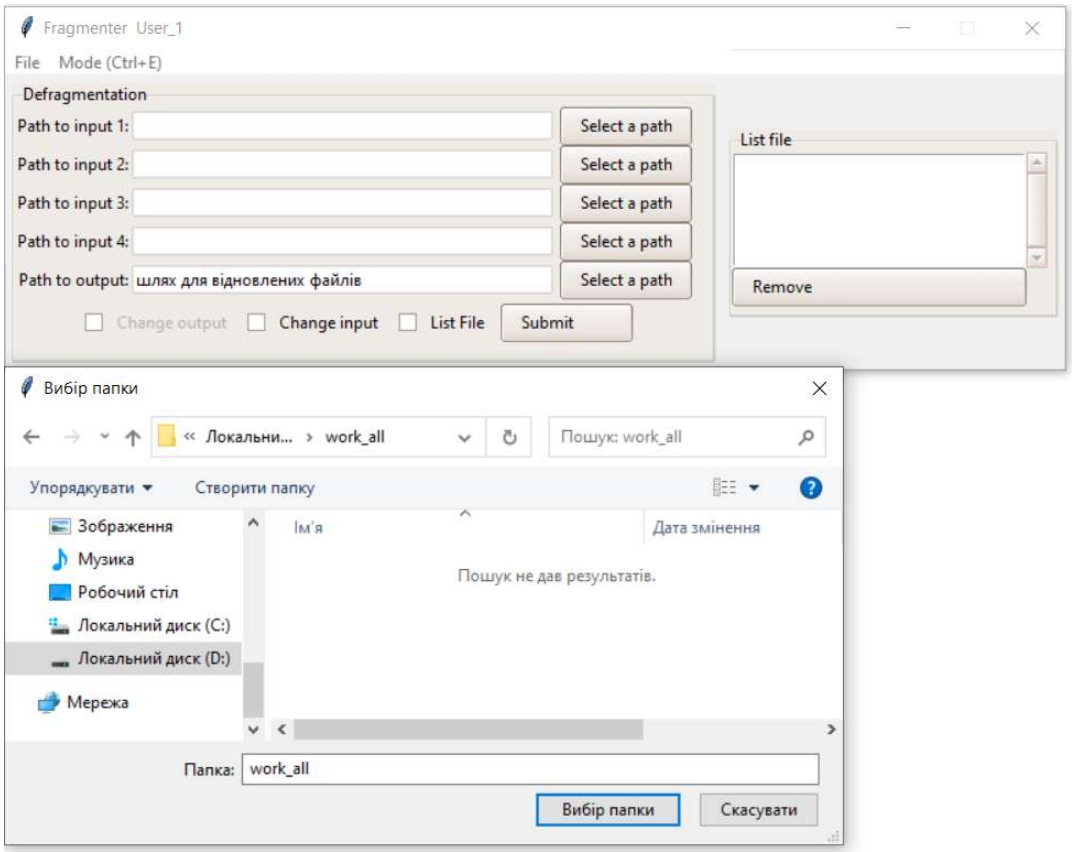

Рисунок 5.19 – Вікно вибору шляху для відновлення файлів

Якщо файл відображається у списку, то після клацання по відповідній радіокнопці «Change input» з’явиться можливість вибрати його в списку та таким чином відновити (рисунок 5.20).

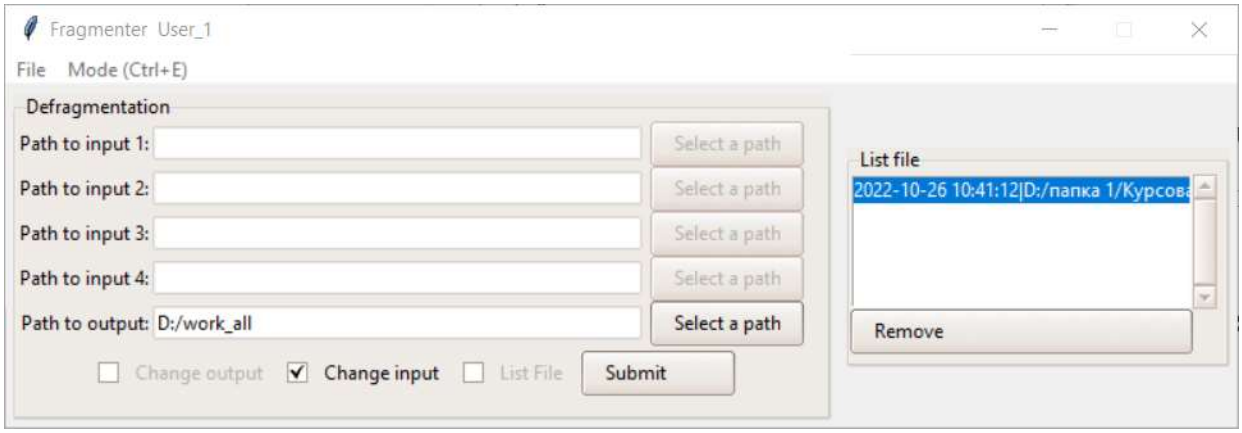

Рисунок 5.20 – Вікно вибору файлу для відновлення з списку

Як видно з рисунку 5.20, шляхи для вибору файлів залишків стали не активними, як і радіокнопка приховування списку файлів «List file».

Після натискання кнопки «Submit» отримується повідомлення про успішне відновлення початкового файлу (рисунок 5.21).

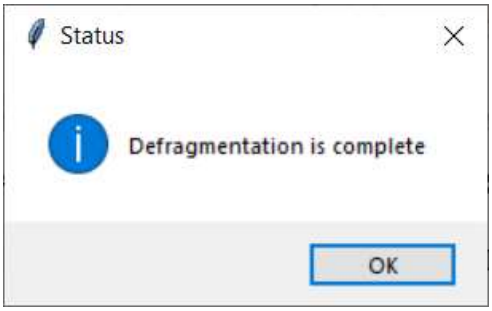

Рисунок 5.21 – Успішне відновлення початкового файлу

Для відновлення початкового файлу, окрім відновлення зі списку, також доступне відновлення шляхом вибору кожного із файлів залишків.

Для цього користувачу необхідно знати послідовність файлів залишків, тому при розділенні початкового файлу кожна частина пишеться в окрему папку

та отримує відповідне розширення: *1.rrns для першого, *2.rrns для другого, *3.rrns для третього і, відповідно, *4.rrns для останнього.

Завдяки цьому при ручному виборі файлів залишків користувач чітко знає, який саме файл і в якому порядку необхідно обирати (рисунок 5.22).

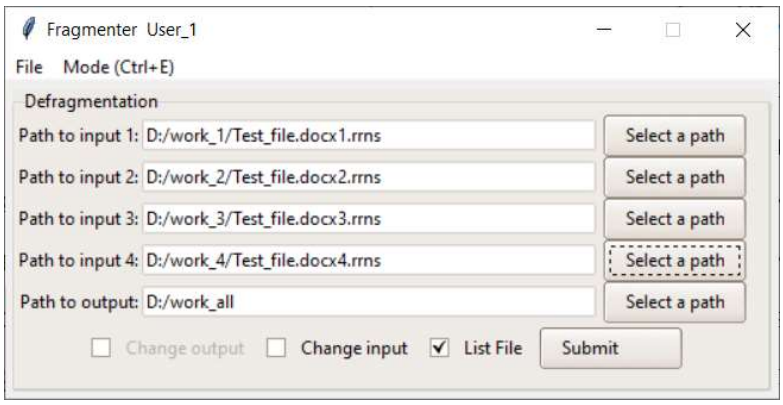

Рисунок 5.22 – Вибір кожного із файлів залишків для відновлення початкового файлу

Після натискання кнопки «Submit» виводиться повідомлення про успішне відновлення початкового файлу в обраній папці.

У розробленому автором методі пропонується використовувати геш-функцію для перевірки цілісності кожного із файлів залишків (розділ 2).

Проте, при локальній реалізації така перевірка зазвичай є надлишковою, тому в спроектованій системі для підтвердження функціональності кроку відновлення, тобто відновлення по трьох з чотирьох файлів залишків, одне із полів залишається пустим (рисунок 5.23).

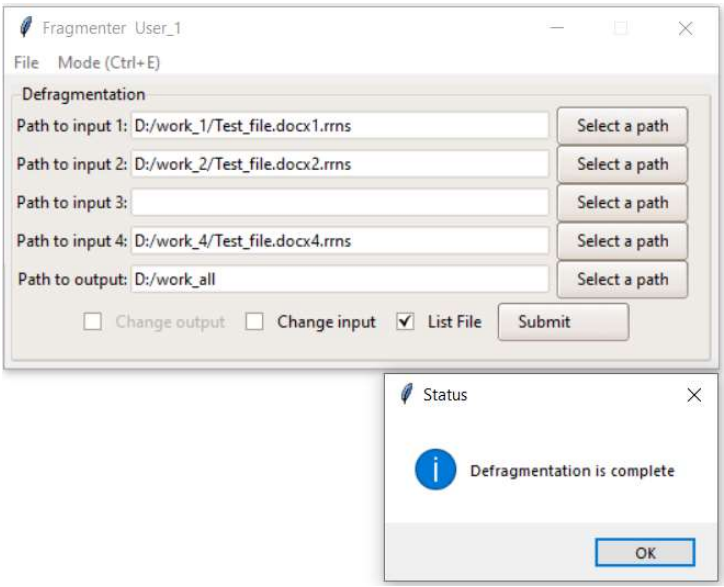

Рисунок 5.23 – Вибір трьох із файлів залишків для відновлення початкового файлу

Завдяки перевірці геш-значень файлів залишків відбувається перевірка цілісності інформації, а завдяки тому, що файли залишків не дають навіть частково отримати інформацію про вміст початкового файлу, забезпечується захищеність даних при зберіганні на віддалених носіях або хмарних сховищах.

5.4. Дослідження розробленого методу та аналіз результатів

Результативність методів кодування та декодування даних залежить від методу, яким кодували інформацію, ступеня захищеності при передачі та збереженні даних.

У пункті 2.2 було розглянуто метод кодування даних, згідно якого на першому кроці необхідно вибрати послідовність інформаційних та перевірочних модулів, які будуть використовуватися при проектуванні системи. Для оптимізації такого вибору у пункті 4.1 було проведено аналіз використання модулів різної розрядності та визначено умови, які забезпечать виконання поставленої задачі.

Для практичної реалізації було обрано набір залишків p_i = [10313, 10321, 10331, 10333], які забезпечують робочий діапазон $H = 1{,}10*10^{12}$.

На кроці 2 зазначеного методу проводиться розбиття файлу на блоки, які, згідно з проведеними у пункті 3.1 обчисленнями, будуть мати розмір п’ять байт.

Для підтвердження функціональності програмного рішення було обрано файл розміром 28 059 байт. У результаті виконання кроку 2 отримується масив *a[]*, який містить 5 612 блоків, останній з яких міститиме не п’ять байт, а чотири (рисунок 5.24).

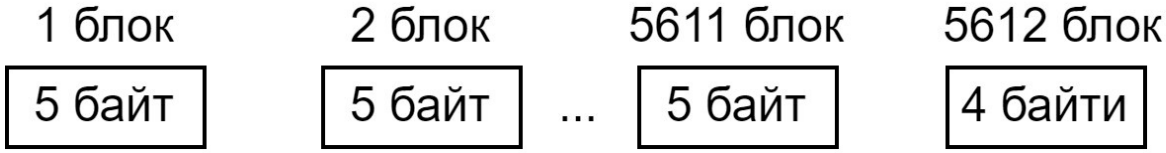

Рисунок 5.24 – Розділення файлу на блоки

На кроці 3 проводиться обчислення залишків по кожному із блоків. Спроектована система складається з чотирьох модулів, а отже файлів залишків буде також чотири (рисунок 5.25).

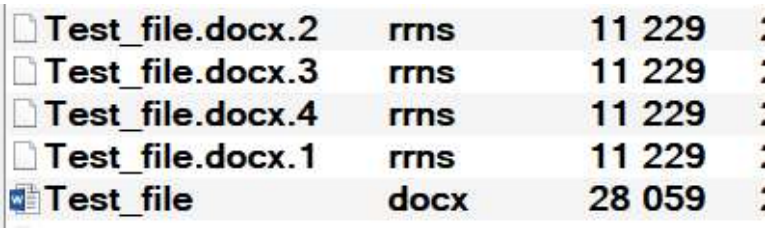

Рисунок 5.25 – Процес розділення даних на блоки та формування файлів залишків

Як вже зазначалося раніше, кількість блоків у всіх файлах залишків буде однакова. Отже, кожен з них міститиме по 5 612 інформаційних блоків розміром по 2 байти, тобто розмір кожного із файлів становитиме 5 612 *2 = 11 224 байти. Проте, окрім блоків, у файли записується ще системна інформація розміром 5 байт.

На кроці 4 проводиться обчислення значення геш-функції та її запис у назву файлу для подальшої перевірки цілісності при декодуванні.

Для виконання цієї операції викликається наступна процедура:

```
def add_hash_as_extension(file_name):
    with open(file_name, 'rb') as file:
        sha256_hash = hashlib.sha256()
        for chunk in iter(lambda: file.read(4096), b''):
            sha256_hash.update(chunk)
        hash_value = sha256_hash.hexdigest()
    file_extension = os.path.splitext(file_name)[1]
    new_file_name = file_name + '.' + hash_value + file_extension
    os.rename(file_name, new_file_name)
    return new_file_name
```

У результаті її роботи в кінці назви файлу добавляється обчислене значення геш-функції SHA-256 (рисунок 5.26).

Рисунок 5.26 – Запис значення геш-функції в назви файлів залишків

Для захисту геш-значень, збережених у назві, від спотворення в процесі завантаження файлу обчислюється контрольна сума з використанням CRC-32 коду, що реалізовано наступною процедурою:

```
def calculate_hash_crc(hash_value):
    CRC32_func = crcmod.predefined.mkCrcFun('crc32')
    hash_bytes = bytes.fromhex(hash_value)
    crc = CRC32_func(hash_bytes)
    crc_hex = format(crc, '08x')
    return crc_hex
```

Отримане в результаті значення *crc_hex* за допомогою наступної процедури добавляється в назву файлу з залишками:

```
def add_crc_as_extension(file_path):
    file_extension = os.path.splitext(file_path)[1]
    file_hash = calculate_file_hash(file_path)
    CRC-32_func = crcmod.predefined.mkCrcFun('crc-32')
    crc_value = CRC-32_func(bytes.fromhex(file_hash))
    crc_hex = format(crc_value, '08x')
    new_file_name = os.path.splitext(file_path)[0] + '_' + crc_hex + file_extension
    new_file_path = os.path.join(os.path.dirname(file_path), new_file_name)
    os.rename(file_path, new_file_path)
    return new_file_path
```

В результаті такої послідовності дій файли залишків, згідно кроку 6, містять у своїй назві значення геш-функції та CRC 32 для підтвердження цілісності при відновленні (рисунок 5.27).

Рисунок 5.26 –Запис CRC-32 в назви файлів залишків

Як видо з рисунку 5.26, контрольна сума CRC-32 відділена від геш-значення знаком «_» для спрощення пошуку та відділення при відновленні.

Останній, сьомий крок процедури кодування, полягає в переміщенні файлів залишків на відповідні носії або у хмарні сховища, завдяки чому забезпечується розподіленість системи.

На цьому процес кодування файлу, в результаті якого утворюються файли залишків, збережені в розподілених сховищах, вважається завершеним.

У випадку, коли необхідно отримати доступ до початкового файлу, необхідно запустити процедуру декодування. Згідно методу декодування, запропонованого у пункті 2.2, а також алгоритмів, розроблених у пунктах 4.2-4.4, для отримання початкового файлу необхідно здійснити наступну послідовність дій.

Першим кроком декодування є отримання доступу до системної інформації, а саме до набору залишків, на основі яких спроектована система. У нашому випадку використовувався набір залишків p_i = [10313, 10321, 10331, 10333], для яких були розраховані базисні числа b_i = [9875156009338430, 2852470894029210, 3440328946688150, 6557132607895170], необхідні для проведення операції відновлення. Окрім цього, з системної інформації, яка збережена у файлі сесії, завантажуються шляхи до файлів залишків.

Як вже зазначалося раніше, завдяки порівнянню геш-значень можливе відновлення початкового вмісту файлу по вмісту інформаційних модулів. У спроектованій системі кількість інформаційних модулів k = 3, тому для початку на кроці два відбувається завантаження трьох файлів залишків.

На третьому кроці відбувається перевірка контрольної суми CRC-32 від геш-значення. Обидва значення беруться з назви файлу. У результаті такої перевірки стає відомо, чи відбулося якесь спотворення при переміщенні файлів на розподілені носії або в хмару. В нашому випадку зберігання відбувалося на локальний носій, тому спотворення не могло виникнути.

У випадку, коли перевірка контрольної суми CRC-32 була успішною, на кроці чотири здійснюється обчислення функції *hash* (F_i) від кожного із трьох обраних файлів.

На кроці п’ять здійснюється порівняння обчисленого значення геш-функції зі значенням, збереженим у назві файлу, завдяки чому підтверджується цілісність кожного із файлів залишків.

Шостий крок відповідає за відновлення вмісту початкового файлу у випадку позитивного проходження зазначених вище перевірок. Таке відновлення здійснюється згідно формули (2.3) і в результаті отримується початковий файл (рисунок 5.27).

Рисунок 5.27 – Відновлення початкового файлу

У випадку, якщо геш-значення або контрольна сума CRC-32, збережені в назві файлів, не співпадає із повторно обчисленою, то на сьомому кроці, згідно запропонованого методу, здійснюється завантаження наступного ($k+1$) файлу, тобто четвертого файлу залишків.

На восьмому кроці необхідно провести перевірку геш-значення та CRC-32 нового завантаженого файлу залишків. У випадку, коли перевірка пройшла успішно, користувач отримує відновлений файл.

При умові, що геш-значення або контрольна сума CRC-32 не співпадає у більше ніж двох файлів залишків, тоді застосовується метод відновлення даних на основі удосконаленого методу проекцій (пункті 4.4). Дана процедура дозволяє відновити дані в частині випадків (таблиця 4.6).

При розробці пропонованого методу виявлення та виправлення помилок на основі геш-функцій було проведено порівняння з існуючим методом виявлення та виправлення помилок, а саме методом проекцій.

Дослідження було проведено для порівняння методу проекцій надлишкової СЗК з трьома інформаційними та двома перевірочними модулями ($n = 5$) і реалізованого методу на основі удосконаленого методу проекцій та порівняння значення геш-функції з трьома інформаційними та одним перевірочним відповідно (рисунок 5.28).

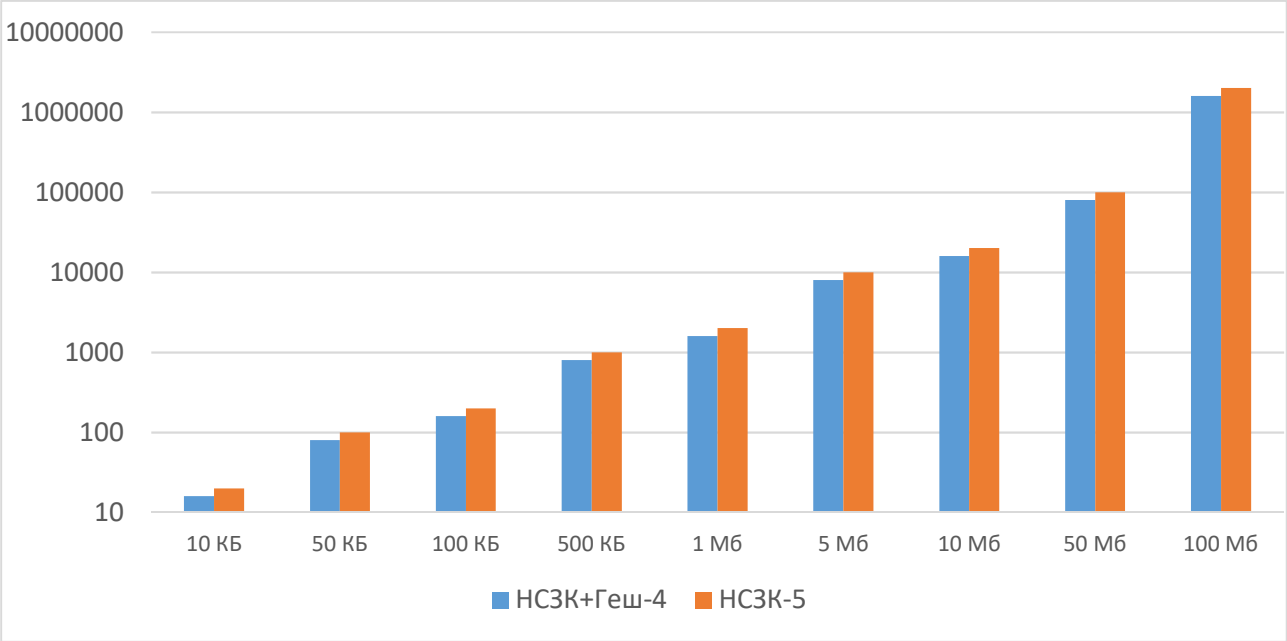

Рисунок 5.28 – Надлишковість пропонованого методу та методу проекцій НСЗК

На основі порівняння надлишковості обох методів можна зазначити, що розроблений метод для реалізації потребує на один носій менше у порівнянні з відомим методом, забезпечуючи ту саму функціональність, а завдяки порівнянню обчислених значень геш-функції є більш ефективним, зменшуючи складність виявлення та виправлення помилок.

Висновки до розділу 5

У розділі розроблено функціональну схему СРЗЗД. Детально описано функціональні складові, що забезпечують роботу системи, а саме модулі: розділення на залишки, запису/зчитування, вибору файлів залишків, зворотного перетворення та відновлення даних.

У рамках виконання дисертаційного дослідження реалізовано програмний продукт на основі описаної системи розподіленого захищеного зберігання даних. Розглянуто режими роботи для розділення на залишки та для зворотного перетворення файлів залишків у початковий файл. Описано метод відновлення даних при використанні трьох файлів залишків з чотирьох.

Наведено реалізацію процедур обчислення геш-функції та контрольної суми CRC-32 на мові Python. Представлено графічне підтвердження функціонування приведених процедур.

Дослідження надлишковості реалізованого методу та методу проекцій на основі використання трьох інформаційних модулів показало, що розроблений метод для реалізації потребує на один носій менше у порівнянні з відомим методом, забезпечуючи ту саму функціональність, а за результатами порівняння геш-значень є більш ефективним, оскільки зменшує складність виявлення та виправлення помилок.

ВИСНОВКИ

У дисертаційній роботі розв’язано актуальну науково-прикладну задачу підвищення захищеності та надійності зберігання даних у локальних та мережевих сховищах шляхом розробки методів і алгоритмів шифрування та кодування даних на основі надлишкової системи залишкових класів. Отримано наступні наукові результати:

1. Проведено аналіз методів резервного копіювання даних та використання апаратно-програмних систем зберігання даних, а саме DAS, NAS, SAN та RAID сховища, а також порівняння ефективності різних типів RAID масивів. Обґрунтовано переваги використання коригуючих кодів системи залишкових класів для підвищення надійності зберігання та забезпечення додаткового рівня захисту даних.

2. Запропоновано метод надійного зберігання даних на основі надлишкової системи залишкових класів та геш-функції. Визначені основні кроки кодування та декодування даних. Проведені дослідження запропонованого методу показали зменшення надлишковості на 33%, для наборів модулів з $n = 4$, в порівнянні з іншими методами виправлення помилок в системі залишкових класів.

3. Проведено порівняння часової складності методів зворотного перетворення СЗК на основі Китайської теореми про залишки. Сформовано оптимальні набори модулів різної розрядності для використання у системах зберігання даних, які забезпечують мінімальну надлишковість.

4. Досліджено криптографічну стійкість шифрування даних на основі асимптотичного розподілу простих чисел. Обчислено залежність криптографічної стійкості від розрядності модулів при різній їх кількості. При оцінці криптографічної стійкості шифрування даних вперше запропоновано враховувати розмір файлів залишків, що забезпечило зменшення розрядності модулів на 8 біт при заданому рівні захисту.

5. Удосконалено метод шифрування даних в системі залишкових класів

шляхом зміни позицій залишків з використанням М-послідовності, що забезпечило зменшення розрядності модулів з 32 до 16 біт при кількості модулів n=4.

6. Проведено дослідження криптографічної стійкості запропонованого методу шифрування даних в надлишковій СЗК та оцінено складність перебору залишків при невідомих модулях. Запропонований метод забезпечує в 11-15 раз вищу криптографічну стійкість з 8-16 бітними модулями, у порівнянні з базовим шифруванням в СЗК, і в середньому у 3 рази з 96-128-бітними модулями.

7. Розроблено алгоритм формування назви файлу залишків, який зберігає значення геш суми та циклічного коду CRC-32 в назві файлу, що забезпечило перевірку цілісності файлу залишків, а також виявлення спотворення значення геш суми в назві файлу за допомогою CRC-32.

8. На основі запропонованих методів кодування та шифрування даних розроблено функціональну схему системи розподіленого захищеного зберігання даних та реалізовано прототип системи розподіленого захищеного зберігання даних, яка забезпечує гарантоване відновлення даних при втраті одного носія даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Горбенко, І., Замула, О., Осипенко, Ю. (2022). Концепція оцінки ризиків кібербезпеки інформаційної системи об'єкта критичної інфраструктури. *Радіотехніка*, 209, 118–129. https://doi.org/10.30837/rt.2022.2.209.12.
2. MITRE ATT&CK®. (n.d.). *Initial Access, Tactic TA0001 – Enterprise.* https://attack.mitre.org/tactics/TA0001.
3. Verizon Business. (n.d.). *DBIR Report 2023 - Master's Guide.* https://www.verizon.com/business/resources/reports/dbir/2023/master-guide/.
4. NATO Advanced Research Workshop on Web Intelligence and Security (2009 'En Boķeķ, Israel). *Web intelligence and security: Advances in data and text mining techniques for detecting and preventing terrorist activities on the web*. Amsterdam : IOS Press, 2010. 265 p.
5. ENISA. (n.d.). *Threat landscape.* https://www.enisa.europa.eu/topics/cyber-threats/threats-and-trends.
6. Nakashima, E. (2018, February 26). *Russian spies hacked the Olympics and tried to make it look like North Korea did it, U.S. officials say.* Washington Post. https://www.washingtonpost.com/world/national-security/russian-spies-hacked-the-olympics-and-tried-to-make-it-look-like-north-korea-did-it-us-officials-say/2018/02/24/44b5468e-18f2-11e8-92c9-376b4fe57ff7_story.html.
7. Valsorda, F. (2021). *Yet another padding oracle in OpenSSL CBC ciphersuites.* The Cloudflare Blog. https://blog.cloudflare.com/yet-another-padding-oracle-in-openssl-cbc-ciphersuites/.
8. Ragan, S. (2016, October 3). *Here are the 61 passwords that powered the Mirai IoT botnet.* CSO Online. https://www.csoonline.com/article/3126924/here-are-the-61-passwords-that-powered-the-mirai-iot-botnet.html.
9. Bunge, J. (2021, June 10). *JBS paid $11 million to resolve ransomware attack.* WSJ. https://www.wsj.com/articles/jbs-paid-11-million-to-resolve-ransomware-attack-11623280781.

10. House, W. (2021). *Press briefing by Press Secretary Jen Psaki, Secretary of Energy Jennifer Granholm, and Secretary of Homeland Security Alejandro Mayorkas*. The White House. https://www.whitehouse.gov/briefing-room/press-briefings/2021/05/11/press-briefing-by-press-secretary-jen-psaki-secretary-of-energy-jennifer-granholm-and-secretary-of-homeland-security-alejandro-mayorkas-may-11-2021/.
11. Cybersecurity and Infrastructure Security Agency. (n.d.). *#StopRansomware Guide - CISA*. https://www.cisa.gov/stopransomware/ransomware-guide.
12. RiskRecon. (n.d.). *White Paper: Five Lessons Learned from Ransomware Attacks*. http://www.riskrecon.com/report-five-lessons-learned-from-ransomware-attacks?utm_term=ransomware&utm_campaign=US+-+Ransomware+Attacks+03-2022&%20utm_source=adwords&utm_medium=ppc&.
13. The Foundation for Defense of Democracies. (n.d.). *Secure the Data, Not the Device: How Decentralized File Storage Creates Resilience Against the Risk of Ransomware.* https://www.fdd.org/wp-content/uploads/2021/10/fdd-memo-secure-the-data-not-the-device.pdf.
14. Cybersecurity and Infrastructure Security Agency. (n.d.). *How can I protect against ransomware?* https://www.cisa.gov/stopransomware/how-can-i-protect-against-ransomware.
15. Chen, D., Yuan, H., Hu, S., Wang, Q., & Wang, C. (2021). BOSSA: a decentralized system for proofs of data retrievability and replication. *IEEE Transactions on Parallel and Distributed Systems*, 32(4), 786–798. https://doi.org/10.1109/tpds.2020.3030063.
16. Ali, S., Wang, G., White, B., & Cottrell, R. L. (2018). A Blockchain-Based Decentralized Data Storage and Access Framework for PingER. *17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, New York, NY, USA, 1303-1308. https://doi.org/10.1109/TrustCom/BigDataSE.2018.00179.

17. Hassan, S., Ibrahim, R., Bingi, K., Chung, T., & Saad, N. (2017). Application of wireless technology for control: A wirelesshart perspective. *Procedia Computer Science*, 105, 240-249.
18. Ge, X., Yang, F., & Han, Q.-L. (2017). Distributed networked control systems: A brief overview. *Information Sciences*, 380, 117-131.
19. Ben Amor, R., & Elloumi, S. (2017). On decentralized control techniques of intercon-nected systems-application to a double-parallel inverted pendulum. *International Conference on Advanced Systems and Electric Technologies*, Hammamet, Tunisia, 85-90. https://doi.org/10.1109/ASET.2017.7983671.
20. Rtibi, H., & Elloumi, S. (2017). Robust decentralized nonlinear control for multimachine power systems. *International Conference on Advanced Systems and Electric Technologies (IC_ASET),* Hammamet, Tunisia, 473-480. https://doi.org/10.1109/ASET.2017.7983739.
21. Kunifuji, T. (2017). Safety Technologies in Autonomous Decentralized Railway Control System. *IEEE 13th International Symposium on Autonomous Decentralized System (ISADS)*, Bangkok, Thailand, 137-142. https://doi.org/10.1109/ISADS.2017.15.
22. Vacca, J. R. (2020). Cloud Computing Security. *In CRC Press eBooks*. https://doi.org/10.1201/9780429055126.
23. Force, J. T. (2018). Risk management framework for information systems and organizations. https://doi.org/10.6028/nist.sp.800-37r2.
24. SNIA. (n.d.). *The SNIA Dictionary* https://www.snia.org/education/dictionary/about-dictionary.
25. Tekin, S., Bicakci, K., Mersin, O., Erdem, G.N., Canbay, A. & Uzunay, Y. (2023). Optimal data backup policies for information systems subject to sudden failure. *Journal of Quality in Maintenance Engineering*, 29, 338-355. https://doi.org/10.1108/JQME-01-2022-0009.
26. Timothy L. Warner. (2020). Backing Up and Restoring Your Azure Data. in *Microsoft Azure For Dummies*. *Wiley*, 211-228.

27. Gadget-Info. (2019). *7 Найкраще програмне забезпечення для шифрування для Windows*. https://uk.gadget-info.com/17956-7-best-encryption-software-for-windows.
28. Gibson, G. A., & Van Meter, R. (2000). Network attached storage architecture. *Communications of the ACM*, 43(11), 37-45. https://doi.org/10.1145/353360.353362.
29. Murugapoopathi, S., & Nawaz, G. M. K. (2015). Maximum performance with minimum cost in data mining applications through the novel online data warehouse architecture by using storage area network with fibre channel fabric. *International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015]*, Nagercoil, India, 1-7. https://doi.org/10.1109/iccpct.2015.7159498.
30. Bombich Software. (n.d.). *Backing up to a disk image* https://bombich.com/kb/ccc6/backing-up-disk-image.
31. CommVault. (2023). *Application Backup & Application Data Management.* https://www.commvault.com/use-cases/backup-solutions/application-backup.
32. Русин, Б. П., Погрелюк, Л. В., Висоцька, В. А., Осипов, М. М., Варецький, Я. Ю., & Капшій, О. В. (2019). Архітектура системи дедублікації та розподілу даних у хмарних сховищах під час резервного копіювання. *Інформаційні технології та комп'ютерна інженерія*, 45(2), 40–63. https://doi.org/10.31649/1999-9941-2019-45-2-40-63.
33. Nakamura, S., Zhao, X., & Nakagawa, T. (2017). Constant and Random Full Backup Models with Incremental and Differential Backup Schemes. *International Journal of Reliability, Quality and Safety Engineering*, 24(03), 1750015. https://doi.org/10.1142/s0218539317500152.
34. SIM-Networks. (n.d.). *Data backup methods: full, incremental and differential backups*. https://www.sim-networks.com/ukr/blog/backup-full-increment-differential.
35. ArchWiki. (n.d.). *File systems (Українська)*. https://wiki.archlinux.org/title/File_systems_(%D0%A3%D0%BA%D1%80%D0%B0%D1%97%D0%BD%D1%81%D1%8C%D0%BA%D0%B0).

36. ESET. (n.d.). *Шифрування файлів - що це і для чого потрібно шифрувати файли?* https://www.eset.com/ua/support/information/entsiklopediya-ugroz/shifrovaniye-faylov/.
37. Prabhakar, A., Savin, P. S., & Chandrasekaran, K. (2015). On-the-Fly encryption security in remote storage. *Fifth International Conference on Advances in Computing and Communications (ICACC),* Kochi, India, 163-168. https://doi.org/10.1109/icacc.2015.102.
38. University of Galway. (n.d.). *BitLocker - University of Galway.* https://www.universityofgalway.ie/information-solutions-services/servicesforstaff/desktopsupport/dataencryption/bitlocker/.
39. Disk Recovery Software and Hard Drive Recovery tool for Windows, Mac, and Linux. (n.d.). *All About Apple FileVault Encryption.* https://www.r-studio.com/What-is-Apple-FileVault.html.
40. VeraCrypt. (n.d.). *VeraCrypt - Free Open source disk encryption with strong security for the Paranoid.* https://www.veracrypt.fr/code/VeraCrypt/.
41. Amster. (2023, June 16). *PC encryption with TrueCrypt.* HackYourMom. https://hackyourmom.com/pryvatnist/perevirka-anonimnosti/shyfruvannya-pk-iz-truecrypt/.
42. ArchWiki. (n.d.). *dm-crypt (Українська).* https://wiki.archlinux.org/title/Dm-crypt_(%D0%A3%D0%BA%D1%80%D0%B0%D1%97%D0%BD%D1%81%D1%8C%D0%BA%D0%B0).
43. Dyne.org. (n.d.). *Tomb: File Encryption on GNU/Linux.* https://dyne.org/software/tomb/.
44. soringpcrepair.com. (n.d.). *Програми для шифрування папок і файлів* https://uk.soringpcrepair.com/program-to-encrypt-folders-and-files/.
45. Martyshko, V. (2023). *Як зашифрувати файли і папки використовуючи EFS в Windows 10, 8 або 7.* Hetman Software. https://hetmanrecovery.com/uk/recovery_news/how-to-encrypt-files-and-folders-using-efs-in-windows-10-8-or-7.htm.
46. ArchWiki. (n.d.). *EncFS.* https://wiki.archlinux.org/title/EncFS.

47. Libfuse. (n.d.). *GitHub - libfuse/libfuse: The reference implementation of the Linux FUSE (Filesystem in Userspace) interface.* GitHub. https://github.com/libfuse/libfuse.
48. Messmer, S. (n.d.). *CryFS: How it works.* CryFS. https://www.cryfs.org/howitworks.
49. Messmer, S., Rill, J., Achenbach, D., & Müller-Quade, J. (2017). A novel cryptographic framework for cloud file systems and CryFS, a Provably-Secure construction. In *Lecture Notes in Computer Science,* 409–429. https://doi.org/10.1007/978-3-319-61176-1_23.
50. eCryptfs. (n.d.). *eCryptfs.* https://www.ecryptfs.org/.
51. IETF Datatracker. (1998, November 1). *RFC 2440: OpenPGP Message Format.* https://datatracker.ietf.org/doc/html/rfc2440.
52. AxCrypt. (n.d.). *File Security made easy.* https://axcrypt.net/.
53. NordLocker file encryption: store and share data securely. (n.d.). *Secure your files in a private cloud.* NordLocker File Encryption: Store and Share Data Securely. https://nordlocker.com/.
54. Lexie. (2021). *Zero-knowledge proofs explained: Part 1.* Home of Internet Privacy. https://www.expressvpn.com/blog/zero-knowledge-proofs-explained/.
55. Kai, Z. (2021). Research on network data storage Technology based on Autonomous Controllable system. *In 2021 International Conference on Computer Engineering and Artificial Intelligence (ICCEAI),* Shanghai, China, 183-186. https://doi.org/10.1109/icceai52939.2021.00035.
56. Chukry, S., & Sbeyti, H. (2019). Security enhancement in storage area network. *7th International Symposium on Digital Forensics and Security (ISDFS)*, Barcelos, Portugal, 1-5. https://doi.org/10.1109/isdfs.2019.8757492.
57. Chamberlain, R. D., & Shands, B. (2007). Direct-Attached Disk Subsystem Performance Assessment. *Fourth International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI 2007)*, San Diego, CA, USA, 71-78. https://doi.org/10.1109/snapi.2007.23.

58. Djemaiel, Y., & Boudriga, N. (2008). Dynamic detection and tolerance of attacks in storage area networks. *22nd International Conference on Advanced Information Networking and Applications - Workshops (aina workshops 2008)*, Gino-wan, Japan, 377-382. https://doi.org/10.1109/waina.2008.34.
59. Yuan, F. (2011). The construction of the network storage system in the data centers of colleges and universities. *IEEE 3rd International Conference on Communication Software and Networks*, Xi'an, China, 208-211. https://doi.org/10.1109/iccsn.2011.6014253.
60. Lanka, A., & Garzevas, A. (2018). Remotely accessible, low power network attached storage device. *Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, Coimbatore, India, 1083-1088. https://doi.org/10.1109/icicct.2018.8473164.
61. Marcel. (2018). Comparing cache-accelerated approach on DAS and NAS as storage solution for small scale virtual desktop deployment – A Case Study at ABC University. *4th International Conference on Wireless and Telematics (ICWT)*, Nusa Dua, Bali, Indonesia, 1-6. https://doi.org/10.1109/ICWT.2018.8527830.
62. Katal, A., Gupta, N., Sharma, S., & Goudar, R. H. (2012). "Information storage on the cloud: A survey of effective storage management system," Students Conference on Engineering and Systems, Allahabad, India, 2012, pp. 1-6, https://doi.org/10.1109/SCES.2012.6199040.
63. Hussain, T., & Habib., S. (2013). A redesign methodology for storage management virtualization. *IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, Ghent, Belgium, pp. 676-679.
64. Aarthy Priscilla., G. (2022). Cloud Computing Elastic Storage System for Storage Optimization. *International Conference on Inventive Computation Technologies (ICICT),* Nepal, 928-934. https://doi.org/10.1109/ICICT54344.2022.9850781.
65. Spillner, J., & Müller., J. (2014). Tutorial on Distributed Data Storage: From Dispersed Files to Stealth Databases. *IEEE/ACM 7th International Conference on*

Utility and Cloud Computing. London, UK, 535-536. https://doi.org/10.1109/UCC.2014.80.

66. Zhang, G., Huang, Z., Ma, X., Yang, S., Wang, Z., & Zheng, W. (2018). RAID+: Deterministic and Balanced Data Distribution for Large Disk Enclosures. 1*6th USENIX Conference on File and Storage Technologies*, Oakland, February 12–15, 280-283.
67. Patterson, D. A., Gibson, G. A., & Katz, R. H. (1988). A case for redundant arrays of inexpensive disks (RAID). *ACM SIGMOD Record*, 17(3), 109–116. https://doi.org/10.1145/50202.50214.
68. Яцків, В.В., Кулина, С.В. (2019). Метод надійного зберігання даних на основі надлишкової системи залишкових класів. *Вісник Хмельницького національного університету. Технічні науки*, 6, 98-104. http://journals.khnu.km.ua/vestnik/wp-content/uploads/2021/01/20-9.pdf.
69. Wan, J., Qu, X., Zhao, N., Wang, J., & Xie, C. (2015). Thin RAID: Thinning Down RAID Array for Energy Conservation. *IEEE Transactions on Parallel & Distributed Systems*, 26, 2903-2915.
70. Rahman, P. A., & Shavier, G. D. N. F. (2018). Reliability model of disk arrays RAID-5 with data striping. *IOP Conference Series*, 327, 788-794. https://doi.org/10.1088/1757-899x/327/2/022087IOP.
71. Kachhala, K., & Gangarde, R. (2016). RAID (Redundant Array of independent Disks). *International Journal of Engineering Trends and Technology,* 35(12), 574–577. https://doi.org/10.14445/22315381/ijett-v35p316.
72. Li, Y., Lee, P. P. C., & Lui, J. C. S. (2016). Analysis of Reliability Dynamics of SSD RAID. *IEEE Transactions on Computers*, 65(4), 1131–1144. https://doi.org/10.1109/tc.2014.2349505.
73. Celesti, A., Fazio, M., Villari, M., & Puliafito, A. (2016). Adding long-term availability, obfuscation, and encryption to multi-cloud storage systems. *Journal of Network and Computer Applications*, 59, 208–218. https://doi.org/10.1016/j.jnca.2014.09.021.

74. Tay, T. F., & Chang, C. (2014). A new algorithm for single residue digit error correction in Redundant Residue Number System. *IEEE International Symposium on Circuits and Systems (ISCAS),* 1748-1751 https://doi.org/10.1109/iscas.2014.6865493.
75. Soma, D. K., Pradhan, A. K., & Narayanan, K. R. (2021). Errors and Erasures decoding of product codes for optical transport networks. *IEEE Communications Letters*, 25(8), 2482–2486. https://doi.org/10.1109/lcomm.2021.3077027.
76. Bardis, N. G., Doukas, N., & Markovskyi, O. P. (2016). Erasure Code for Efficient Error Correction in Block Data Transmission. *Third International Conference on Mathematics and Computers in Sciences and in Industry (MCSI)*, 296-301. https://doi.org/10.1109/mcsi.2016.061.
77. Aguilera, M. K., Janakiraman, R., & Xu, L. (2005). Using Erasure Codes Efficiently for Storage in a Distributed System. *International Conference on Dependable Systems and Networks (DSN'05)*, 336-345. https://doi.org/10.1109/dsn.2005.96.
78. Цаволик, Т.Г. (2016). Коригуючі коди в системі залишкових класів зі спеціальними модулями. *Вимірювальна та обчислювальна техніка в технологічних процесах. Технічні науки*, 3. 100–104.
79. Koshman, S., Krasnobayev, V., Nikolsky, S., & Kovalchuk, D. (2023). The structure of the computer system in the residual classes. *Advanced Information Systems*, 7(2), 41–48. https://doi.org/10.20998/2522-9052.2023.2.06.
80. Goh, V. T., & Siddiqi, M. U. (2008). Multiple error detection and correction based on redundant residue number systems. *IEEE Transactions on Communications*, 56(3), 325–330. https://doi.org/10.1109/tcomm.2008.050401.
81. Barsi, F., & Maestrini, P. (1973). Error correcting properties of redundant residue number systems. *IEEE Transactions on Computers*, 22(3), 307–315. https://doi.org/10.1109/t-c.1973.223711.
82. Кошман, С., Попенко, В., & Кононченко, А. (2019). Приклади використання методу корекції помилок даних, що представлені в класі лишків. *Комп'ютерні науки та кібербезпека*, (3), 27-36.

83. Николайчук, Я. М.; Волинський, О. І.; Кулина, С. В. (2007) Теоретичні основи побудови та структура спецпроцесорів в базисі Крестенсона. *Вісник Хмельницького національного університету*, 3, 85-90.
84. Krasnobayev, V., Koshman, S., Yanko, A., & Martynenko, A. A. (2018). Method of Error Control of the Information Presented in the Modular Number System. *International Scientific-Practical Conference Problems of Infocommunications*. Science and Technology (PIC S&T), 39-42. https://doi.org/10.1109/infocommst.2018.8632049.
85. Krasnobayev, V., Kuznetsov, A., Yanko, A., & Kuznetsov, A. (2019). Correction Codes in the System of Residual Classes. IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T), 488–492. https://doi.org/10.1109/picst47496.2019.9061253.
86. Mohan, P. V. A. (2016). Residue Number Systems. *Springer eBooks*. https://doi.org/10.1007/978-3-319-41385-3.
87. Krasnobayev, V., Koshman, S., Moroz, S., Kalashnikov, V., & Kalashnikov, V. (2020). Data errors control in the modular number system based on the nullification procedure. *International Journal of Computing*, 19(2), 237-246. https://doi.org/10.47839/ijc.19.2.1767.
88. Яцків, В.В. (2015). Виявлення та виправлення багатократних помилок на основі модулярних коректуючих кодів. *Інформаційні технології та комп'ютерна інженерія,* 33, 77–82.
89. Кулина, С.В. (2018). Виявлення помилок на основі коригуючих кодів системи залишкових класів. *Фізико-технологічні проблеми передавання, оброблення та зберігання інформації в інфокомунікаційних системах: Матеріали VII Міжнародної науково-практичної конференції,* 126-127.
90. Кулина, С. В. (2022). Виявлення та виправлення помилок у захищених системах зберігання даних методом обчислення синдрому. *Збірник матеріалів проблемної наукової міжгалузевої конференції «Автоматизація та комп'ютерно-інтегровані технології» (АКІТ-2022),* Тернопіль, 67-69.

91. Кулина, С. В. (2019). Виправлення помилок при передачі даних на основі обчислення проекцій числа. *Прикладні науково-технічні дослідження: матеріали III міжнар. наук.-практ. конф., 3-5 квіт. Академія технічних наук України*, Івано-Франківськ: Симфонія форте, 36.
92. Кулина, С. В. (2022). Виявлення та виправлення помилок у захищених системах зберігання даних методом обчислення проекції числа. *Збірник матеріалів науково-практичної конференції молодих вчених, аспірантів та студентів «Кібербезпека та комп'ютерно-інтегровані технології» (КБКІТ-2022)*, Тернопіль, 13-16.
93. Кулина, С. В. (2022). Виявлення та виправлення помилок у захищених системах зберігання даних на основі обчислення проекцій числа. *Informatics and Mathematical Methods in Simulation*, 12, 337-345. http://nbuv.gov.ua/UJRN/Itmm_2022_12_4_10.
94. Кулина, С. В. (2023). Система розподіленого захищеного зберігання даних. *Вісник Львівського державного університету безпеки життєдіяльності*, 27, 48–59. https://doi.org/10.32447/20784643.27.2023.06.
95. Youssef, M.I., Emam, A.E., & Elghany, M.A. (2012). Multi-Layer Data Encryption using Residue Number System in DNA Sequence. *International Journal of Computer Applications*, 45, 19-24.
96. Agbedemnab, P. A., Baagyere, E. Y., & Daabo, M. I. (2019). A New Image Encryption and Decryption Technique using Genetic Algorithm and Residual Numbers. *IEEE AFRICON*, 1–9. https://doi.org/10.1109/africon46755.2019.9133919.
97. Цаволик, Т. Г., Яцків, В. В. (2017). Метод формування корегувальних кодів у системі залишкових класів. *Науковий вісник НЛТУ України*, 27 (3), 191–194.
98. Mohan, P. V. A. (2016). Residue number systems. *In Springer eBooks*. https://doi.org/10.1007/978-3-319-41385-3.
99. Xiao, H., Garg, H., Hu, J., & Xiao, G. (2016). New error control algorithms for residue number system codes. *Etri Journal*, 38(2), 326–336. https://doi.org/10.4218/etrij.16.0115.0575.

100. Yatskiv, V., Kulyna, S., Bykovyy, P., Maksymyuk, T., & Sachenko, A. (2020). Method of Reliable Data Storage Based on Redundant Residue Number System. *IEEE 5th International Symposium on Smart and Wireless Systems Within the Conferences on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS–SWS)*, 1–4. https://doi.org/10.1109/idaacs-sws50031.2020.9297052.

101. Wang, Y. (2000). Residue-to-binary converters based on new Chinese remainder theorems. *IEEE Transactions on Circuits and Systems Ii: Analog and Digital Signal Processing*, 47(3), 197–205. https://doi.org/10.1109/82.826745.

102. Kulyna, S. (2022). Evaluation of the reverse transformation methods complexity of the residual number system for secure data storage. *Вісник Тернопільського Національного Технічного Університету*, 107(3), 21–28. https://doi.org/10.33108/visnyk_tntu2022.03.021.

103. Omondi, A. R., & Premkumar, B. (2007). Residue number systems: Theory and Implementation. *World Scientific*.

104. Якименко, І. З., Касянчук, М. М., Тимошенко, Л. М., Гребень, Н. Є. (2013). Алгоритми опрацювання інформаційних потоків в комп'ютерних системах. *Інформатика та математичні методи в моделюванні*, 3, 266-274. http://nbuv.gov.ua/UJRN/Itmm_2013_3_3_10.

105. Янко, А. С., Ковальчук, Д. М. (2023). Дослідження можливості відмовостійкого функціонування комп'ютерної системи в непозиційній системі числення в залишкових класах. *Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління : тези доп. 13-ї міжнар. наук.-техн. конф., 26-27 квітня 2023 р.*, Харків: Impress, 50. https://repository.kpi.kharkov.ua/handle/KhPI-Press/65563.

106. The Backblaze Storage Cloud. (2022). *Backblaze Drive Stats for 2022.* https://www.backblaze.com/company/about.html.

107. Nykolaychuk, Y., Yakymenko, I., Vozna, N., & Kasianchuk, M. (2022). Residue Number System Asymmetric Cryptoalgorithms. *Cybernetics and Systems Analysis*, 58(4), 611–618. https://doi.org/10.1007/s10559-022-00494-7.

108. Zawiślak, S., Kasianchuk, M., Iakymenko, I., & Jancarczyk, D. (2022). Methods of crypto-stable symmetric encryption in the residual number system. *Procedia Computer Science*, 207, 128–137. https://doi.org/10.1016/j.procs.2022.09.045.

109. Kasianchuk, M., Yakymenko, I., & Nykolaychuk, Y. (2021). Symmetric cryptoalgorithms in the residue number system. *Cybernetics and Systems Analysis*, 57(2), 329–336. https://doi.org/10.1007/s10559-021-00358-6.

110. Biyashev, R., Nyssanbayeva, S., Haumen, A., & Kapalova, N. (2016). Modified Symmetric Block Encryption-Decryption Algorithm Based on Modular Arithmetic. *International Conference on Advanced Materials Science and Environmental Engineering*. https://doi.org/10.2991/amsee-16.2016.69.

111. Kapalova, N., & Dyusenbayev, D. (2016). Security analysis of an encryption scheme based on nonpositional polynomial notations. *Open Engineering*, 6(1), 250-258. https://doi.org/10.1515/eng-2016-0034 .

112. Abood, O. G., Guirguis, S. K. (2018). A survey on cryptography algorithms. *International Journal of Scientific and Research Publications*, 8.(7), 410-415. https://doi.org/10.29322/IJSRP.8.7.2018.p7978.

113. Kapalova, N., Sakan, K., Algazy, K., & Dyusenbayev, D. (2022). Development and Study of an Encryption Algorithm. *Computation*, 10(11), 198. https://doi.org/10.3390/computation10110198.

114. Gautam, T., & Jain, A. (2015). Analysis of brute force attack using TG — Dataset. *SAI Intelligent Systems Conference (IntelliSys)*, 984–988. https://doi.org/10.1109/intellisys.2015.7361263.

115. Idhom, M., Wahanani, H. E., & Fauzi, A. (2020). Network Security System on Multiple Servers Against Brute Force Attacks. *6th Information Technology International Seminar (ITIS),* 258–262. https://doi.org/10.1109/itis50118.2020.9321108.

116. Riman, C. F., & Abi-Char, P. E. (2015). Comparative Analysis of Block Cipher-Based Encryption Algorithms: A survey. *Information Security and Computer Fraud*, 3(1), 1–7. https://doi.org/10.12691/iscf-3-1-1.

117. Dixit, P., Gupta, A. K., Trivedi, M. C., & Yadav, V. K. (2017). Traditional and Hybrid Encryption Techniques: A survey. *In Lecture notes on data engineering and communications technologies,* 239–248. https://doi.org/10.1007/978-981-10-4600-1_22.

118. Papachristodoulou, L., Fournaris, A. P., Papagiannopoulos, K., & Batina, L. (2018). Practical evaluation of protected residue number System scalar multiplication. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 259–282. https://doi.org/10.46586/tches.v2019.i1.259-282.

119. Zawiślak, S., Kasianchuk, M., Iakymenko, I., & Jancarczyk, D. (2022). Methods of crypto-stable symmetric encryption in the residual number system. *Procedia Computer Science*, 207, 128–137. https://doi.org/10.1016/j.procs.2022.09.045.

120. Schoinianakis, D. (2020). Residue arithmetic systems in cryptography: a survey on modern security applications. *Journal of Cryptographic Engineering*, 10(3), 249–267. https://doi.org/10.1007/s13389-020-00231-w.

121. Shirvanimoghaddam, M. (2021). On the Hamming Weight Distribution of Subsequences of Pseudorandom Sequences. *IEEE International Symposium on Information Theory (ISIT),* 1671–1675. https://doi.org/10.1109/isit45174.2021.9517946.

122. Kopec, D. (2019). *Classic Computer Science Problems in Python*. Manning Publications.

123. García, J. E., Cotrina, G., Peinado, A., & Ortiz, A. (2022). Security and efficiency of linear feedback shift registers in GF(2N) using N-BIT grouped operations. *Mathematics*, 10(6), 996. https://doi.org/10.3390/math10060996.

124. Yatskiv, V., Kulyna, S., Yatskiv, N., & Kulyna, H. (2020). Protected Distributed Data Storage Based on Residue Number System and Cloud Services. *10th International Conference on Advanced Computer Information Technologies (ACIT)*, 796–799. https://doi.org/10.1109/acit49673.2020.9208849.

ДОДАТКИ

ДОДАТОК А

Алгоритми та типи шифрування поширених програмних продуктів

№	Назва програмного продукту	Операційна система	Алгоритм шифрування	Тип шифру-вання
1	**2**	**3**	**4**	**5**
1.	BitLocker	Windows	AES CBC або AES XTS 128-/256- bit key	OTFE
2.	FileVault	Mac OS	XTS-AES-128-/256- bit key	OTFE
3.	TrueCrypt	Windows, Mac OS X, Linux	AES 256-bit key, Blowfish 448-bit key, CAST5 128-bit key, Serpent 256-bit key, Triple DES, Twofish 256-bit key.	OTFE
4.	VeraCrypt	Windows, Mac OS, Linux	AES, Serpent, Twofish, Camellia, Кузнечик	OTFE
5.	dm-crypt / LUKS	Linux, FreeBSD	XTS-AES-128-/256- bit key	OTFE, FBE
6.	Tomb	Linux	XTS-AES-128-/256- bit key	OTFE
7.	EFS	Windows	AES, SHA, ECC	FBE
8.	EncFS	Mac OS, Linux, FreeBSD	AES 16-байтний блоковий шифр з довжиною ключа 128-/256- bit; Blowfish 8-байтний блоковий шифр з довжиною ключа 128-/256- bit	FBE
9.	CryFS	Mac OS X, Linux	XTS-AES-128-/256- bit key	FBE

Прод. Додатку А

1	2	3	4	5
№	Назва програмного продукту	Операційна система	Алгоритм шифрування	Тип шифру-вання
10.	eCryptfs	Linux	AES з розміром блоку – 16 і розміром ключа в байтах - 16, 32; Blowfish з розміром блоку 8 і розміром ключа в байтах - 16, 56; DES3_EDE з розміром блоку 8 і розміром ключа в байтах – 24; Twofish з розміром блоку 16; розмір ключа в байтах - 16, 32 CAST6 з розміром блоку 16; розмір ключа в байтах - 16, 32 CAST5 із розміром блоку 8; розмір ключа в байтах - 5, 16	FBE
11.	AxCrypt	Windows, Mac OS X, iOS, Android	AES із 128-/256-бітним ключем	FBE
12.	NordLocker	Windows, Mac OS, Android	AES-256, ECC	FBE

ДОДАТОК Б

Набори оптимальних модулів для проектування систем зберігання даних на основі СЗК

№	H	H_b	P_{r_b}	N	N, %	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_{per}
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
1.	$3{,}23*10^2$	1	3	2	200	17	19	---	---	---	---	---	23
2.	$6{,}76*10^4$	2	6	4	200	257	263	---	---	---	---	---	269
3.	$8{,}29*10^4$	2	4	2	100	41	43	47	---	---	---	---	53
4.	$1{,}69*10^7$	3	6	3	100	4099	4111	---	---	---	---	---	4127
5.	$1{,}82*10^7$	3	8	5	167	257	263	269	---	---	---	---	271
6.	$2{,}12*10^7$	3	5	2	67	61	67	71	73	---	---	---	79
7.	$4{,}30*10^9$	4	9	5	125	65537	65539	---	---	---	---	---	65543
8.	$4{,}32*10^9$	4	8	4	100	1621	1627	1637	---	---	---	---	1657
9.	$4{,}93*10^9$	4	10	6	150	257	263	269	271	---	---	---	277
10.	$7{,}45*10^9$	4	6	2	50	83	89	97	101	103	---	---	107
11.	$1{,}10*10^{12}$	5	9	4	80	1049963	1049977	---	---	---	---	---	1049999
12.	$1{,}10*10^{12}$	5	8	3	60	10313	10321	10331	---	---	---	---	10333
13.	$1{,}16*10^{12}$	5	10	5	100	1031	1033	1039	1049	---	---	---	1051
14.	$1{,}36*10^{12}$	5	12	7	140	257	263	269	271	277	---	---	281
15.	$1{,}74*10^{12}$	5	7	2	40	101	103	107	109	113	127	---	131
16.	$2{,}82*10^{14}$	6	12	6	100	16777259	16777289	---	---	---	---	---	16777291
17.	$2{,}82*10^{14}$	6	12	6	100	65537	65539	65543	---	---	---	---	65551
18.	$2{,}87*10^{14}$	6	10	4	67	4099	4111	4127	4129	---	---	---	4133
19.	$3{,}38*10^{14}$	6	12	6	100	787	797	809	811	821	---	---	823

Прод. Додатку Б

1	2	3	4	5	6	7	8	9	10	11	12	13	14
20.	$3,84*10^{14}$	6	12	6	100	257	263	269	271	277	281	---	283
21.	$3,09*10^{14}$	6	8	2	33	103	107	109	113	127	131	137	139
22.	$7,21*10^{16}$	7	12	5	71	416107	416147	416149	---	---	---	---	416153
23.	$7,22*10^{16}$	7	10	3	43	16369	16381	16411	16417	---	---	---	16421
24.	$7,44*10^{16}$	7	12	5	71	2351	2357	2371	2377	2381	---	---	2383
25.	$7,96*10^{16}$	7	14	7	100	643	647	653	659	661	673	---	677
26.	$1,09*10^{17}$	7	16	9	129	257	263	269	271	277	281	283	293
27.	$1,84*10^{19}$	8	12	4	50	2642323	2642329	2642333	---	---	---	---	2642351
28.	$1,85*10^{19}$	8	15	7	88	65537	65539	65543	65551	---	---	---	65557
29.	$4,72*10^{21}$	8	15	6	67	262147	262151	262153	262187	---	---	---	262193
30.	$1,90*10^{19}$	8	12	4	50	7151	7159	7177	7187	7193	---	---	7207
31.	$1,98*10^{19}$	8	14	6	75	1621	1627	1637	1657	1663	1667	---	1669
32.	$2,35*10^{19}$	8	16	8	100	569	571	577	587	593	599	601	607
33.	$4,72*10^{21}$	9	16	7	78	16777259	16777289	16777291	---	---	---	---	16777331
34.	$4,76*10^{21}$	9	12	3	33	21617	21647	21649	21661	21673	---	---	21683
35.	$4,91*10^{21}$	9	14	5	56	4099	4111	4127	4129	4133	4139	---	4153
36.	$5,25*10^{21}$	9	16	7	78	1237	1249	1259	1277	1279	1283	1289	1291
37.	$1,21*10^{24}$	10	15	5	50	1048583	1048589	1048601	1048609	---	---	---	1048613
38.	$1,21*10^{24}$	10	18	8	80	65537	65539	65543	65551	65557	---	---	65563
39.	$1,21*10^{24}$	10	14	4	40	10313	10321	10331	10333	10337	10343	---	10357
40.	$1,29*10^{24}$	10	16	6	60	2753	2767	2777	2789	2791	2797	2801	2803
41.	$3,10*10^{26}$	11	18	7	64	198673	198689	198701	198719	198733	---	---	198761
42.	$3,11*10^{26}$	11	14	3	27	26003	26017	26021	26029	26041	26053	---	26083

Прод. Додатку Б

1	2	3	4	5	6	7	8	9	10	11	12	13	14
43.	$3{,}16*10^{26}$	11	16	5	45	6079	6089	6091	6101	6113	6121	6131	6133
44.	$7{,}92*10^{28}$	12	20	8	67	16777259	16777289	16777291	16777331	---	---	---	16777333
45.	$7{,}93*10^{28}$	12	21	9	75	65537	65539	65543	65551	65557	65563	---	65579
46.	$7{,}98*10^{28}$	12	16	4	33	13421	13441	13451	13457	13463	13469	13477	13487
47.	$2{,}03*10^{31}$	13	21	8	62	165133	165161	165173	165181	165203	165211	---	165229
48.	$2{,}03*10^{31}$	13	16	3	23	29641	29663	29669	29671	29683	29717	29723	29741
49.	$5{,}19*10^{33}$	14	18	4	29	5534411	5534413	5534429	5534483	5534489	---	---	5534519
50.	$5{,}19*10^{33}$	14	21	7	50	416107	416147	416149	416153	416159	416167	---	416201
51.	$5{,}20*10^{33}$	14	24	10	71	65537	65539	65543	65551	65557	65563	65579	65581
52.	$1{,}33*10^{36}$	15	24	9	60	16777259	16777289	16777291	16777331	16777333	---	---	16777337
53.	$1{,}33*10^{36}$	15	21	6	40	1048571	1048573	1048583	1048589	1048601	1048609	---	1048613
54.	$1{,}33*10^{36}$	15	24	9	60	144719	144731	144737	144751	144757	144763	144773	144779
55.	$3{,}40*10^{38}$	16	21	5	31	2642231	2642239	2642257	2642287	2642291	2642323	---	2642329
56.	$3{,}40*10^{38}$	16	24	8	50	319547	319567	319577	319589	319591	319601	319607	319639
57.	$8{,}71*10^{40}$	17	21	4	24	6658013	6658027	6658049	6658051	6658079	6658097	---	6658103
58.	$8{,}72*10^{40}$	17	24	7	41	705631	705643	705689	705713	705737	705751	705763	705769
59.	$2{,}23*10^{43}$	18	28	10	56	16777259	16777289	16777291	16777331	16777333	16777337	---	16777381
60.	$2{,}23*10^{43}$	18	24	6	33	1558129	1558177	1558189	1558201	1558213	1558217	1558223	1558243
61.	$5{,}71*10^{45}$	19	24	5	26	3440719	3440771	3440807	3440819	3440839	3440849	3440861	3440863
62.	$1{,}46*10^{48}$	20	24	4	20	7597763	7597769	7597823	7597829	7597847	7597859	7597873	7597901
63.	$3{,}74*10^{50}$	21	32	11	52	16777259	16777289	16777291	16777331	16777333	16777337	16777381	16777421

ДОДАТОК В

ПРОГРАМНИЙ КОД РЕАЛІЗАЦІЇ СИСТЕМИ РОЗПОДІЛЕНОГО ЗАХИЩЕНОГО ЗБЕРІГАННЯ ДАНИХ

main.py

```
import os
import math
import datetime
import constants
import cryptocode
import tkinter as tk
import tkinter.simpledialog
from ttkthemes import ThemedTk
from hashing import hash_sha256
from funcs import fragment, defragment
from tkinter import BooleanVar, filedialog as fd, ttk, messagebox
class App(ThemedTk):
    def __init__(self):
        # Theme of the window
        super().__init__(
            theme="clearlooks"
        )
            # Name of the window
        self.title('Fragmenter Guest')
        # Turn off the window resizing
        self.resizable(
            width=False,
            height=False,
        )
        # Main Frame
        self.main_frame = ttk.LabelFrame(
            master=self,
            text="Fragmentation",
        )
        self.main_frame.grid(
            column=0,
            row=0,
            pady=5,
            padx=5,
            ipadx=5,
            ipady=5,
            sticky='n',
```

```
)
# Input part
self.input_label = ttk.Label(
    self.main_frame,
    text='Path to file(s):',
)
self.input_label.grid(
    row=0,
    column=0,
    sticky='e',
)
self.input_entry = ttk.Entry(
    self.main_frame,
    width=50,
)
self.input_entry.grid(
    row=0,
    column=1,
)
self.input_button = ttk.Button(
    self.main_frame,
    text='Select a file',
    width=11,
    command=self.__select_files,
)
self.input_button.grid(
    row=0,
    column=2,
    padx=5,
)
# Output part
self.output_label = ttk.Label(
    self.main_frame,
    text='Path to output:',
)
self.output_label.grid(
    row=1,
    column=0,
    sticky='e',
)
self.output_entry = ttk.Entry(
    self.main_frame,
    width=50,
```

```
)
self.output_entry.grid(
    row=1,
    column=1,
)
self.output_button = ttk.Button(
    self.main_frame,
    text='Select a path',
    width=11,
    command=(lambda: self.__select_path()),
)
self.output_button.grid(
    row=1,
    column=2,
    padx=5,
)
# The frame that consists of checkbuttons and the submit button
self.control_frame = ttk.Frame(
    master=self.main_frame,
    borderwidth=3
)
self.control_frame.grid(
    column=0,
    row=3,
    sticky='n',
    columnspan=3,
)
# Storages
self.output_entries: list[ttk.Entry] = []
self.output_entries.append(self.output_entry)
self.output_labels: list[ttk.Label] = []
self.output_buttons: list[ttk.Button] = []
self.input_entries: list[ttk.Entry] = []
self.input_entries.append(self.input_entry)
self.input_labels: list[ttk.Label] = []
self.input_buttons: list[ttk.Button] = []
self.output_var = tk.BooleanVar()
self.log_input_var = tk.BooleanVar()
self.log_var = tk.BooleanVar(value=True)
self.sort_var = tk.StringVar()
self.mode_var = BooleanVar()
self.sorting_parameters = (
    'Name',
```

```
    'Time',
)
self._last_keycode = 0
self.records = {}
# Change output to several selected paths
self.output_checkbutton = ttk.Checkbutton(
    master=self.control_frame,
    text='Change output',
    variable=self.output_var,
    onvalue=1,
    offvalue=0,
    command=self.__change_output,
)
self.output_checkbutton.grid(
    row=0,
    column=0,
    padx=5,
)
# Change input to choosing from the log
self.input_checkbutton = ttk.Checkbutton(
    master=self.control_frame,
    text='Change input',
    variable=self.log_input_var,
    onvalue=1,
    offvalue=0,
    command=self.__change_input,
    state=tk.DISABLED,
)
self.input_checkbutton.grid(
    row=0,
    column=1,
    padx=5,
)
# Show the log
self.log_checkbutton = ttk.Checkbutton(
    master=self.control_frame,
    text='Files list',
    variable=self.log_var,
    onvalue=False,
    offvalue=True,
    command=self.__recent_actions,
)
self.log_checkbutton.grid(
```

```
    row=0,
    column=2,
    padx=5,
)
# Submit the selected operation over input files
self.submit_button = ttk.Button(
    self.control_frame,
    text='Submit',
    width=11,
    command=(lambda: fragment(
        self.tk.splitlist(self.input_entry.get())[0],
        fun=self.__add_to_listbox,
        output=[self.output_entry.get()],
    ))
)
self.submit_button.grid(
    row=0,
    column=3,
    padx=5,
)
# The log frame that contains recent actions
self.log_frame = ttk.LabelFrame(
    text="Log",
)
self.log_frame.grid(
    column=1,
    row=0,
)
# A list that containts recent actions
self.log = tk.Listbox(
    self.log_frame,
    listvariable=[],
    selectmode='extended',
)
self.log.grid(
    column=0,
    row=0,
    sticky='nwes',
    ipady=15,
    columnspan = 3,
)
# Log Y coordinate scrollbar
self.log_y_scrollbar = ttk.Scrollbar(
```

```
    master=self.log,
    orient=tk.VERTICAL,
    command=self.log.yview,
)
self.log['yscrollcommand'] = self.log_y_scrollbar.set
self.log_y_scrollbar.pack(
    side=tk.RIGHT,
    fill=tk.Y,
)
# Log X coordinate scrollbar
self.log_x_scrollbar = ttk.Scrollbar(
    master=self.log,
    orient=tk.HORIZONTAL,
    command=self.log.xview,
)
self.log['xscrollcommand'] = self.log_x_scrollbar.set
self.log_x_scrollbar.pack(
    side=tk.BOTTOM,
    fill=tk.X,
)
# Remove selected actions from the log
self.log_remove_button = ttk.Button(
    master=self.log_frame,
    text='Remove',
    width=10,
    command=self.__delete_records,
)
self.log_remove_button.grid(
    column=0,
    row=1,
    sticky='nswe',
)
# Sort the log by a selected value
self.log_sort_button = ttk.Button(
    master=self.log_frame,
    text='Sort',
    width=10,
    command=self.__sort_records,
)
self.log_sort_button.grid(
    column=1,
    row=1,
    sticky='nswe',
```

```
)
# Sorting combobox
self.sort_combobox = ttk.Combobox(
    master=self.log_frame,
    width=10,
    textvariable=self.sort_var,
)
self.sort_combobox[constants.VALUES] = self.sorting_parameters
self.sort_combobox.grid(
    column=2,
    row=1,
    sticky='nswe'
)
self.sort_combobox.current(1)
# Hide the log
self.__recent_actions()
self.bind(
    "<KeyPress>",
    func=self.__handle_keypress
)
# Add handler to closing window
self.protocol(
    name="WM_DELETE_WINDOW",
    func=self.__on_closing,
)
# Menu
self.menubar = tk.Menu(self)
filemenu = tk.Menu(
    master=self.menubar,
    tearoff=0,
)
filemenu.add_command(
    label='Open',
    command=self.__load_file,
)
filemenu.add_command(
    label='Save',
    command=self.__save_file,
)
filemenu.add_command(
    label='Clear',
    command=self.__clear,
)
```

```
        filemenu.add_separator()
        filemenu.add_command(
            label='Exit',
            command=self.__on_closing,
        )
        modemenu = tk.Menu(
            master=self.menubar,
            tearoff=0,
        )
        modemenu.add_radiobutton(
            label="Fragmentation",
            command=self.__change_mode,
            variable=self.mode_var,
            value=False,
        )
        modemenu.add_radiobutton(
            label="Defragmentation",
            command=self.__change_mode,
            variable=self.mode_var,
            value=True,
        )
        self.menubar.add_cascade(
            label='File',
            menu=filemenu,
        )
        self.menubar.add_cascade(
            label='Mode (Ctrl+E)',
            menu=modemenu,
        )
        self.config(
            menu=self.menubar,
        )
    def __load_file(self):
        """Load a session file"""
        files = [('Session file', '*.ssn')]
        filename = fd.askopenfilename(
            title='Open a file',
            initialdir='/',
            filetypes=files,
        )
        if filename:
            password = tkinter.simpledialog.askstring(
                title="Password",
```

```
            prompt="Enter password:",
            show='*',
        )
        data = open(filename).read()
        data = cryptocode.decrypt(data, password)
        if isinstance(data, bool) and not data:
            messagebox.showerror("Error", "A wrong password")
            return
        records, data = data.split(constants.SEPARATOR)
        self.records = eval(records)
        self.log.insert(0, *data.split('\n'))
        self.title(f'Fragmenter {filename.split("/")[-1].split(".")[0]}')
        messagebox.showinfo("Info", "A right password")
    def __select_files(self):
        """Вибирає файли для дефрагментації"""
        filename = fd.askopenfilenames(
            title='Open a file',
            initialdir='/',
        )
        self.input_entry.delete(0, last=len(self.input_entry.get()))
        self.input_entry.insert(0, filename)
    def __select_file(self, n):
        """Вибирає файл для дефрагментації"""
        filename = fd.askopenfilename(
            title='Open a file',
            initialdir='/',
        )
        self.input_entries[n].delete(0, last=tk.END)
        self.input_entries[n].insert(0, filename)
    def __select_path(self, n=0):
        """Вибирає каталог зберігання"""
        path = fd.askdirectory()
        self.output_entries[n].delete(0, last=len(self.output_entries[n].get()))
        self.output_entries[n].insert(0, path)
    def __change_output(self):
        """Міняє каталог зберігання від одного до декількох й навпаки"""
        if self.output_var.get():
            self.control_frame.grid(
                column=0,
                row=6,
                sticky='n',
                columnspan=3,
            )
```

```
self.output_label.destroy()
self.output_entry.destroy()
self.output_button.destroy()
self.output_entries.clear()
self.output_buttons.clear()
for i in range(1, 5):
    label = ttk.Label(
        self.main_frame,
        text=f'Path to output {i}:',
    )
    self.output_labels.append(label)
    label.grid(
        row=i,
        column=0,
        sticky='e',
    )
    entry = ttk.Entry(
        self.main_frame,
        width=50,
    )
    self.output_entries.append(entry)
    entry.grid(
        row=i,
        column=1,
    )
    button = ttk.Button(
        self.main_frame,
        text='Select a path',
        width=11,
        command=(
            lambda i=i: self.__select_path(i - 1)
        ),
    )
    button.grid(
        row=i,
        column=2,
        padx=5
    )
    self.output_buttons.append(button)
output = (
    lambda: [entry.get() or self.output_entries[0].get() for entry in self.output_entries]
)
self.submit_button.configure(
```

```
            command=(
                lambda: fragment(
                    self.tk.splitlist(self.input_entry.get())[0],
                    output=output(),
                    fun=self.__add_to_listbox,
                )
            ),
        )
    else:
        self.control_frame.grid(
            column=0,
            row=3,
            sticky='n',
            columnspan=3,
        )
        list(map(ttk.Entry.destroy, self.output_entries))
        list(map(ttk.Button.destroy, self.output_buttons))
        list(map(ttk.Label.destroy, self.output_labels))
        self.output_labels.clear()
        self.output_entries.clear()
        self.output_buttons.clear()
        self.output_label = ttk.Label(
            self.main_frame,
            text='Path to output:'
        )
        self.output_label.grid(
            row=1,
            column=0,
            sticky='e')
        self.output_entry = ttk.Entry(
            self.main_frame,
            width=50
        )
        self.output_entry.grid(
            row=1,
            column=1
        )
        self.output_entries.append(self.output_entry)
        self.output_button = ttk.Button(
            self.main_frame,
            text='Select a path',
            width=11,
            command=(
```

```
            lambda: self.__select_path()
        )
    )
    self.output_button.grid(
        row=1,
        column=2,
        padx=5
    )
    self.submit_button.configure(
        command=(
            lambda: fragment(
                self.tk.splitlist(self.input_entry.get())[0],
                fun=self.__add_to_listbox,
                output=[self.output_entry.get()],
            )
        ),
    )
def __add_to_listbox(self, files=[]):
    """Додає елементи до ListBox"""
    date = datetime.datetime.now().strftime(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S'))
    [file.close() for file in files]
    info = f'{files[0].name[:-6].split("/")[-1]}|{date}|{self.__convert_size(os.path.getsize(files[0].name) * 4)}'
    self.records[date] = [(hash_sha256(open(file.name, errors="ignore").read()), file.name) for file in files]
    self.log.insert(0, info)
def __change_input(self):
    """Міняє ввід файлів на вибір із ListBox"""
    self.output_checkbutton.configure(
        state=tk.DISABLED,
    )
    if self.log_var.get():
        self.log_var.set(False)
        self.__recent_actions()
    if self.log_input_var.get():
        new_input = (
            lambda: defragment(
                self.__check_hash(self.records[self.log.get(self.log.curselection()).split('|')[1]]),
                output=[self.output_entry.get()],
            )
        )
        if self.mode_var.get():
            list(map(lambda entry: entry.configure(state=tk.DISABLED), self.input_entries))
            list(map(lambda button: button.configure(state=tk.DISABLED), self.input_buttons))
        else:
```

```
            self.input_entry.configure(
                state=tk.DISABLED,
            )
            self.input_button.configure(
                state=tk.DISABLED,
            )
        self.submit_button.configure(
            command=new_input,
        )
        self.log_checkbutton.configure(
            state=tk.DISABLED,
        )
    else:
        if self.mode_var.get():
            list(map(lambda entry: entry.configure(state=tk.NORMAL), self.input_entries))
            list(map(lambda button: button.configure(state=tk.NORMAL), self.input_buttons))
        else:
            self.input_entry.configure(
                state=tk.NORMAL,
            )
            self.input_button.configure(
                state=tk.NORMAL,
            )
        self.log_checkbutton.configure(
            state=tk.NORMAL,
        )
        input = (
            lambda: [entry.get() if entry.get() else None for entry in self.input_entries]
        )
        self.submit_button.configure(
            command=(
                lambda: defragment(
                    raw_files=input(),
                    output=[self.output_entry.get()]
                )
            )
        )

def __convert_size(self, size_bytes):
    """Конвертує байти в нормальний вигляд"""
    if size_bytes == 0: return "0B"
    size_name = ("B", "KB", "MB", "GB", "TB", "PB", "EB", "ZB", "YB")
    i = int(math.floor(math.log(size_bytes, 1024)))
```

```
    p = math.pow(1024, i)
    s = round(size_bytes / p, 2)
    return "%s %s" % (s, size_name[i])
def __save_file(self):
    """Зберігає сесію"""
    files = [('Session file', '*.ssn')]
    file = fd.asksaveasfile(
        filetypes=files,
        defaultextension=files,
    )
    if file:
        password = tkinter.simpledialog.askstring(
            title="Password",
            prompt="Enter password:",
            show='*',
        )
        data = '\n'.join(self.log.get(0, self.log.size()))
        data = cryptocode.encrypt(repr(self.records) + constants.SEPARATOR + data, password)
        file.write(data)
        self.title(f'Fragmenter {file.name.split("/")[-1].split(".")[0]}')
def __on_closing(self):
    """Контролює закриття програми"""
    if messagebox.askokcancel("Quit", "Do you want to quit?"):
        if messagebox.askyesno('Save', "Do you want to save a session?"):
            self.__save_file()
        self.destroy()
def __delete_records(self):
    """Видаляє вибрані записі з ListBox"""
    for selected_checkbox in self.log.curselection():
        del self.records[self.log.get(selected_checkbox).split('|')[1]]
        self.log.delete(selected_checkbox)
def __sort_records(self):
    """Сортує записи в ListBox"""
    records = self.log.get(0, tk.END)
    self.log.delete(0, tk.END)
    self.log.insert(0, *sorted(records, key=lambda record: record.split('|')[self.sort_combobox.current()]))
def __clear(self):
    """Очищає сесію"""
    self.records = {}
    self.log.delete(0, tk.END)
    self.title('Fragmenter Guest')

def __recent_actions(self):
```

```
        """Ховає й витягує вікно Log"""
        if self.log_var.get():
            self.log_frame.grid_forget()
        else:
            self.log_frame.grid(
                column=1,
                row=0,
                pady=5,
                padx=5,
                ipadx=3,
                ipady=3,
            )
    def __change_mode(self):
        """Turn modes"""
        # defragmentation
        if self.mode_var.get():
            self.input_label.destroy()
            self.input_entry.destroy()
            self.input_button.destroy()
            self.input_buttons.clear()
            self.input_entries.clear()
            self.input_labels.clear()
            for i in range(4):
                label = ttk.Label(
                    self.main_frame,
                    text=f'Path to input {i + 1}:',
                )
                self.input_labels.append(label)
                label.grid(
                    row=i,
                    column=0,
                    sticky='e',
                )
                entry = ttk.Entry(
                    self.main_frame,
                    width=50,
                )
                self.input_entries.append(entry)
                entry.grid(
                    row=i,
                    column=1
                )
                button = ttk.Button(
```

```
            self.main_frame,
            text='Select a path',
            width=11,
            command=(
                lambda i=i: self.__select_file(i)
            ),
        )
        button.grid(
            row=i,
            column=2,
            padx=5,
        )
        self.input_buttons.append(button)

self.main_frame.configure(
    text="Defragmentation"
)
if self.output_var.get():
    self.output_var.set(False)
    self.__change_output()
self.output_label.grid(
    row=i + 1,
    column=0,
)
self.output_entry.grid(
    row=i + 1,
    column=1,
)
self.output_button.grid(
    row=i + 1,
    column=2,
)
self.control_frame.grid(
    column=0,
    row=6,
    sticky='n',
    columnspan=3,
)
input = (
    lambda: [entry.get() if entry.get() else None for entry in self.input_entries]
)
self.submit_button.configure(
    command=(
```

```
            lambda: defragment(
                raw_files=input(),
                output=[self.output_entry.get()]
            )
        )
    )
    if self.output_var.get():
        self.output_var.set(False)
        self.__change_output()
    self.output_checkbutton.configure(
        state=tk.DISABLED
    )
    self.input_checkbutton.configure(
        state=tk.NORMAL
    )
# fragmentation
else:
    if self.log_input_var.get():
        self.log_input_var.set(False)
        self.mode_var.set(True)
        self.__change_input()
        self.mode_var.set(False)
    self.main_frame.configure(
        text="Fragmentation"
    )
    list(map(ttk.Entry.destroy, self.input_entries))
    list(map(ttk.Button.destroy, self.input_buttons))
    list(map(ttk.Label.destroy, self.input_labels))
    self.input_labels.clear()
    self.input_entries.clear()
    self.input_buttons.clear()
    self.control_frame.grid(
        column=0,
        row=3,
        sticky='n',
        columnspan=3,
    )
    self.submit_button.configure(
        command=(
            lambda: fragment(
                self.tk.splitlist(self.input_entry.get())[0],
                fun=self.__add_to_listbox,
                output=[self.output_entry.get()],
```

```
        )
    ),
)
self.output_checkbutton.configure(
    state=tk.NORMAL
)
self.input_checkbutton.configure(
    state=tk.DISABLED
)
self.input_label = ttk.Label(
    self.main_frame,
    text='Path to file(s):',
)
self.input_label.grid(
    row=0,
    column=0,
    sticky='e',
)
self.input_entry = ttk.Entry(
    self.main_frame,
    width=50,
)
self.input_entry.grid(
    row=0,
    column=1,
)
self.input_entries.append(self.input_entry)
self.input_button = ttk.Button(
    self.main_frame,
    text='Select a file',
    width=11,
    command=self.__select_files,
)
self.input_button.grid(
    row=0,
    column=2,
    padx=5,
)
self.output_label.grid(
    row=2,
    column=0,
)
self.output_entry.grid(
```

```
            row=2,
            column=1,
        )
        self.output_button.grid(
            row=2,
            column=2,
        )
    def __handle_keypress(self, e):
        """handle the key pressing"""
        if self._last_keycode == constants.LCTRL_KEYCODE and e.keycode == constants.E_KEYCODE:
            self.mode_var.set(
                False if self.mode_var.get() else True
            )
            self.__change_mode()
            return

        self._last_keycode = e.keycode
    def __check_hash(self, records: list[tuple[int, str]]) -> list[str]:
        """Check if hashes of files are valid"""
        result = []
        for record in records:
            try:
                if record[0] == hash_sha256(open(record[1], errors="ignore").read()):
                    result.append(record[1])
            except:
                result.append(None)
        return result
root = App()
root.mainloop()
```

funcs.py

```
import os
import time
from tkinter import messagebox
def fragment(file, M=[65479, 65497, 65519, 65521], output=[], fun=None):
    a = []
    rest = 0
    with open(file, "rb") as f: # Записує данні з файлу в десятковому представленні і знаходить залишок
        length = os.stat(file).st_size
        for i in range(0, length, 5):
            if i + 5 < length:
                byte = f.read(5)
            else:
                rest = length - i
                byte = f.read(rest)
            i = int.from_bytes(byte, 'big')
            a.append(i)
    length_of_list = len(a)
    if not output[0]:
        files = [open(f'{file}{i}.rrns', 'wb') for i in range(1, 5)] # Відкриває файли
    else:
        if len(output) == 4:
            files = [open(f'{output[i]}/{file.split("/")[-1]}{i + 1}.rrns', 'wb') for i in range(4)]
        else:
            files = [open(f'{output[0]}/{file.split("/")[-1]}{i + 1}.rrns', 'wb') for i in range(4)]
    one_bytes = rest.to_bytes(1, 'big') # Залишок в байтах
    four_bytes = length_of_list.to_bytes(4, 'big') # Довжина в байтах
    for i in range(4): # Записує залишок й довжину в кожний файл
        files[i].write(one_bytes + four_bytes)
    for i in range(length_of_list): # Записує дані в файли
        x = [a[i] % M[j] for j in range(4)]
        for i in range(4):
            files[i].write(x[i].to_bytes(2, 'big'))
    if fun: fun(files)
    messagebox.showinfo("Status", "Fragmentation is complete")
def defragment(raw_files: list[str], M=[65479, 65497, 65519, 65521], k=[36466, 22997, 38753, 32793], output=[]):
    R = M[0] * M[1] * M[2] * M[3]
    R1 = M[0] * M[1] * M[2]
    R2 = M[0] * M[1] * M[3]
    R3 = M[0] * M[2] * M[3]
    R4 = M[1] * M[2] * M[3]
    t = [R // M[0] * k[0], R // M[1] * k[1], R // M[2] * k[2], R // M[3] * k[3]] # Обчислюємо ортогональні базиси
```

```
if not raw_files[0]:
    exch = raw_files[1].split('/')
    if output[0]:
        new_file = open(f"{output[0]}/new_{exch[-1].replace('2.rrns', ")}", "wb")
    else:
        new_file = open(raw_files[1].replace(exch[-1], 'new_' + exch[-1].replace('2.rrns', ")), "wb")
    files = [open(f, 'rb') for f in raw_files if f]
    rest = int.from_bytes(files[0].read(1), byteorder='big', signed=False)
    length_of_list = int.from_bytes(files[0].read(4), byteorder='big', signed=False)
    files[1].read(5), files[2].read(5)
    a = []
    end = b''
    for i in range(length_of_list):
        x1 = int.from_bytes(files[0].read(2), byteorder='big', signed=False)
        x2 = int.from_bytes(files[1].read(2), byteorder='big', signed=False)
        x3 = int.from_bytes(files[2].read(2), byteorder='big', signed=False)
        z = (x1 * t[1] + x2 * t[2] + x3 * t[3]) % R4
        a.append(z)
    if rest > 0:
        length_of_list = length_of_list - 1
        for i in range(length_of_list):
            four_bytes = a[i].to_bytes(5, byteorder='big', signed=False)
            new_file.write(four_bytes)
        i = len(a) - 1
        four_bytes = a[i].to_bytes(rest, byteorder='big', signed=False)
        new_file.write(four_bytes)
    else:
        for i in range(length_of_list):
            four_bytes = a[i].to_bytes(5, byteorder='big', signed=False)
            new_file.write(four_bytes)
    new_file.write(end)
elif not raw_files[1]:
    exch = raw_files[0].split('/')
    if output[0]:
        new_file = open(f"{output[0]}/new_{exch[-1].replace('1.rrns', ")}", "wb")
    else:
        new_file = open(raw_files[0].replace(exch[-1], 'new_' + exch[-1].replace('1.rrns', ")), "wb")
    files = [open(f, 'rb') for f in raw_files if f]
    rest = int.from_bytes(files[0].read(1), byteorder='big', signed=False)
    length_of_list = int.from_bytes(files[0].read(4), byteorder='big', signed=False)
    files[1].read(5), files[2].read(5)
    a = []
    end = b''
```

```
    for i in range(length_of_list):
        x0 = int.from_bytes(files[0].read(2), byteorder='big', signed=False)
        x2 = int.from_bytes(files[1].read(2), byteorder='big', signed=False)
        x3 = int.from_bytes(files[2].read(2), byteorder='big', signed=False)
        z = (x0 * t[0] + x2 * t[2] + x3 * t[3]) % R3
        a.append(z)
    if rest > 0:
        length_of_list = length_of_list - 1
        for i in range(length_of_list):
            four_bytes = a[i].to_bytes(5, byteorder='big', signed=False)
            new_file.write(four_bytes)
        i = len(a) - 1
        four_bytes = a[i].to_bytes(rest, byteorder='big', signed=False)
        new_file.write(four_bytes)
    else:
        for i in range(length_of_list):
            four_bytes = a[i].to_bytes(5, byteorder='big', signed=False)
            new_file.write(four_bytes)
    new_file.write(end)
elif not raw_files[2]:
    exch = raw_files[0].split('/')
    if output[0]:
        new_file = open(f"{output[0]}/new_{exch[-1].replace('1.rrns', ")}", "wb")
    else:
        new_file = open(raw_files[0].replace(exch[-1], 'new_' + exch[-1].replace('1.rrns', ")), "wb")
    files = [open(f, 'rb') for f in raw_files if f]
    rest = int.from_bytes(files[0].read(1), byteorder='big', signed=False)
    length_of_list = int.from_bytes(files[0].read(4), byteorder='big', signed=False)
    files[1].read(5), files[2].read(5)
    a = []
    end = b''
    for i in range(length_of_list):
        x0 = int.from_bytes(files[0].read(2), byteorder='big', signed=False)
        x1 = int.from_bytes(files[1].read(2), byteorder='big', signed=False)
        x3 = int.from_bytes(files[2].read(2), byteorder='big', signed=False)
        z = (x0 * t[0] + x1 * t[1] + x3 * t[3]) % R2
        a.append(z)
    if rest > 0:
        length_of_list = length_of_list - 1
        for i in range(length_of_list):
            four_bytes = a[i].to_bytes(5, byteorder='big', signed=False)
            new_file.write(four_bytes)
        i = len(a) - 1
```

```
            four_bytes = a[i].to_bytes(rest, byteorder='big', signed=False)
            new_file.write(four_bytes)
        else:
            for i in range(length_of_list):
                four_bytes = a[i].to_bytes(5, byteorder='big', signed=False)
                new_file.write(four_bytes)
        new_file.write(end)
    elif not raw_files[3]:
        exch = raw_files[0].split('/')
        if output[0]:
            new_file = open(f"{output[0]}/new_{exch[-1].replace('1.rrns', ")}", "wb")
        else:
            new_file = open(raw_files[0].replace(exch[-1], 'new_' + exch[-1].replace('1.rrns', ")), "wb")
        files = [open(f, 'rb') for f in raw_files if f]
        rest = int.from_bytes(files[0].read(1), byteorder='big', signed=False)
        length_of_list = int.from_bytes(files[0].read(4), byteorder='big', signed=False)
        files[1].read(5), files[2].read(5)
        a = []
        end = b''
        for i in range(length_of_list):
            x0 = int.from_bytes(files[0].read(2), byteorder='big', signed=False)
            x1 = int.from_bytes(files[1].read(2), byteorder='big', signed=False)
            x2 = int.from_bytes(files[2].read(2), byteorder='big', signed=False)
            z = (x0 * t[0] + x1 * t[1] + x2 * t[2]) % R1
            a.append(z)
        if rest > 0:
            length_of_list = length_of_list - 1
            for i in range(length_of_list):
                four_bytes = a[i].to_bytes(5, byteorder='big', signed=False)
                new_file.write(four_bytes)
            i = len(a) - 1
            four_bytes = a[i].to_bytes(rest, byteorder='big', signed=False)
            new_file.write(four_bytes)
        else:
            for i in range(length_of_list):
                four_bytes = a[i].to_bytes(5, byteorder='big', signed=False)
                new_file.write(four_bytes)
        new_file.write(end)
    else:
        files = [open(f, 'rb') for f in raw_files]
        rest = int.from_bytes(files[0].read(1), 'big')
        length_of_list = int.from_bytes(files[0].read(4), 'big')
        files[1].read(5), files[2].read(5), files[3].read(5)
```

```
        a = []
        exch = raw_files[0].split('/')
        if output[0]:
            new_file = open(f"{output[0]}/new_{exch[-1].replace('1.rrns', ")}", "wb")
        else:
            new_file = open(raw_files[0].replace(exch[-1], 'new_' + exch[-1].replace('1.rrns', ")), "wb")
        for i in range(length_of_list):
            x0 = int.from_bytes(files[0].read(2), 'big')
            x1 = int.from_bytes(files[1].read(2), 'big')
            x2 = int.from_bytes(files[2].read(2), 'big')
            x3 = int.from_bytes(files[3].read(2), 'big')
            z = (x0 * t[0] + x1 * t[1] + x2 * t[2] + x3 * t[3]) % R
            a.append(z)
        if rest > 0:
            length_of_list = length_of_list - 1
            for i in range(length_of_list):
                four_bytes = a[i].to_bytes(5, 'big')
                new_file.write(four_bytes)
            i = len(a) - 1
            four_bytes = a[i].to_bytes(rest, 'big')
            new_file.write(four_bytes)
        else:
            for i in range(length_of_list):
                four_bytes = a[i].to_bytes(5, 'big')
                new_file.write(four_bytes)
    messagebox.showinfo("Status", "Defragmentation is complete")
if __name__ == '__main__':
    time_current = time.time()
    fragment('eiffel_tower.jpg')
    time_end = time.time()
    print(f'Fragment time: {time_end - time_current}')
    time_current = time.time()
    defragment([
        'eiffel_tower.jpt[1].rrns',
        'eiffel_tower.jpt[2].rrns',
        'eiffel_tower.jpt[3].rrns',
        'eiffel_tower.jpg4.rrns',
    ])
    time_end = time.time()
    print(f'Defragment time: {time_end - time_current}')
```

ЗАТВЕРДЖУЮ
Перший проректор
Західноукраїнського національного
університету
_____ Микола ШИНКАРИК
«___» _____ 2023 р.

АКТ

про впровадження результатів дисертаційної роботи
Кулини Сергія Васильовича
“Методи та алгоритми захищеного розподіленого зберігання даних на основі надлишкової системи залишкових класів”

Комісія у складі декана факультету комп’ютерних інформаційних технологій Західноукраїнського національного університету (ЗУНУ), д.т.н., проф. Дивака Миколи Петровича (голова комісії), завідувача кафедри кібербезпеки ЗУНУ, д.т.н., проф. Яцківа Василя Васильовича (член комісії), професор кафедри кібербезпеки ЗУНУ, д.т.н., проф. Касянчука Михайла Миколайовича (член комісії), підтверджує, що результати дисертаційної роботи на здобуття ступеня доктора філософії за спеціальністю 125 «Кібербезпека та захист інформації» Кулини С.В. впроваджені і використовуються в навчальному процесі при підготовці магістрів та аспірантів на кафедрі кібербезпеки Західноукраїнського національного університету при вивченні дисциплін:

1. Дослідження і проектування систем захисту інформації:
   - кодування інформаційних потоків в системі залишкових класів;
   - рішення прикладних задач теорії чисел на основі модульних операцій базису Крестенсона.
2. Оцінка складності алгоритмів шифрування:
   - оцінка складності переходу від однієї основи систем числення до іншої;
   - алгоритми обчислення теоретико-числових функцій.
3. Методи шифрування в системі залишкових класів:
   - коректність, стійкість та надійність криптоалгоритмів в СЗК;
   - симетричні та асиметричні криптоалгоритми з використанням надлишкової СЗК.

Декан факультету комп’ютерних
інформаційних технологій,
д.т.н., професор Микола ДИВАК

Завідувач кафедри
кібербезпеки, д.т.н., професор Василь ЯЦКІВ

професор кафедри
кібербезпеки, д.т.н., професор Михайло КАСЯНЧУК

ЗАТВЕРДЖУЮ
Проректор з наукової роботи
Західноукраїнського національного
університету
_____ Зеновій ЗАДОРОЖНИЙ
“ ” _____ 2023 р.

АКТ

про використання результатів дисертаційної роботи
Кулини Сергія Васильовича на тему:
«Методи та алгоритми захищеного розподіленого зберігання даних на основі надлишкової системи залишкових класів»

Комісія у складі: голови – декана факультету комп’ютерних інформаційних технологій, д.т.н., проф. Дивака М.П. та членів: начальника науково-дослідної частини Семанюк В.З., завідувача кафедри кібербезпеки, д.т.н., професора Яцківа В.В. склали цей акт про те, що дослідження та результати дисертаційної роботи Кулини С.В. використані під час виконання науково-дослідних робіт на кафедрі кібербезпеки факультету комп’ютерних інформаційних технологій, зокрема в рамках держбюджетних НДР:

1) “Теоретичні основи та апаратні засоби підвищення продуктивності роботи безпровідних сенсорних мереж” (державний реєстраційний номер 0117U000414) на посаді молодшого наукового співробітника у 2018 році, у якій автором досліджено методи виправлення багатократних помилок на основі коригуючих кодів СЗК та розроблено структурну схему кодування/декодування даних;

2) з виконання завдань Перспективного плану розвитку наукового напряму «Технічні науки» ЗУНУ (1 етап: “Розробка методів та алгоритмів захищеного зберігання даних”, 2 етап: “Розвиток систем підтримки рішень, керованих моделями та даними, в умовах невизначеності”, державний реєстраційний номер 0121U114705) на посаді молодшого наукового співробітника, відповідального виконавця у 2021 році та наукового співробітника у 2022 році, у якій автором досліджено швидкодію відновлення даних та розроблено алгоритм розподіленого зберігання даних.

А також в рамках госпдоговірних НДР:

1) “Розробка алгоритмів надійного розподіленого зберігання даних на основі модулярних коригуючих кодів” (державний реєстраційний номер 0118U100457) у 2018-2019 роках, у якій автором було розроблено алгоритми надійного розподіленого зберігання даних;

2) “Методи та алгоритми захищеного зберігання даних на основі кодів системи залишкових класів” (державний реєстраційний номер 0121U107651) у 2021 році, у якій автором було спроектовано та реалізовано систему розподіленого зберігання даних на основі системи залишкових класів та принципів побудови віртуальних сховищ.

Голова комісії
декан факультету комп’ютерних
інформаційних технологій,
д.т.н., професор Микола ДИВАК

Члени комісії:
начальниця НДЧ
д.е.н., професор Віта СЕМАНЮК

завідувач кафедри кібербезпеки,
д.т.н., професор Василь ЯЦКІВ

ТОВ «Тервікнопласт»
Код ЄДРПОУ 33992917, Україна,
46002, м.Тернопіль, проспект Степана Бандери, 38
тел.: 0 800 50 53 52, www.viknaroff.ua

Банк АТ «ПУМБ»
UA373348510000000002600543948
ТФ АТ КБ «Приватбанк»
UA623387830000026001060502481

ЗАТВЕРДЖУЮ
Директор ТОВ «ТЕРВІКНОПЛАСТ»
_____Сергій ЗАХАРЧИШИН
«_____» _____ 2023 року

АКТ

про впровадження результатів дисертаційної роботи
Кулини Сергія Васильовича
на тему «Методи та алгоритми захищеного розподіленого зберігання даних на основі надлишкової системи залишкових класів»

Ми, комісія в складі: Чубко В.І та Драпак В.І, склали даний акт про те, що результати дисертаційної роботи Кулини С.В. на тему «Методи та алгоритми захищеного розподіленого зберігання даних на основі надлишкової системи залишкових класів» використані при розробці системи резервного копіювання даних, зокрема:

– метод та алгоритм виявлення пошкоджених файлів залишків на основі коригуючих кодів системи залишкових класів;

– метод та алгоритм виправлення помилок при спотворенні одного із файлів залишків;

– метод шифрування конфіденційних даних шляхом зміни позицій залишків з використанням псевдовипадкових послідовностей при зберіганні на хмарних сервісах.

Застосування вказаних методів дозволило підвищити надійність зберігання даних на розподілених носіях та захищеність даних при зберіганні на хмарних сервісах.

Начальник ІТ відділу _____ Чубко В.І.

Заступник начальника ІТ відділу _____ Драпак В.І.