Галицький фаховий коледж імені В’ячеслава Чорновола
відділення комп’ютерних технологій
циклова комісія інформатики та комп’ютерних дисциплін

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач відділення

комп’ютерних технологій

Наталія СТЕФУРАК /_____/

підпис

«___»_____ 202___р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи
освітньо-професійного ступеня «фаховий молодший бакалавр»
зі спеціальності 122 «Комп’ютерні науки»

на тему: «Програмний засіб для захищеного обміну даними з корекцією помилок»

Студент групи КН-41 Ренат ДАВЛЕТОВ _____
(підпис)

Керівник роботи Степан ІВАСЬЄВ _____
(підпис)

Консультанти:

з техніко-економічного обґрунтування Любов МЕЛЕНЧУК _____
(підпис)

Нормоконтролер Надія ГАВРИШКІВ _____
(підпис)

Тернопіль – 2025

Галицький фаховий коледж імені В’ячеслава Чорновола
відділення комп’ютерних технологій
циклова комісія інформатики та комп’ютерних дисциплін

ЗАТВЕРДЖУЮ

Завідувач відділення

комп’ютерних технологій

Наталія СТЕФУРАК /_____/

підпис

«___»_____ 202___р.

ЗАВДАННЯ

на кваліфікаційну роботу
на здобуття освітньо-професійного ступеня «фаховий молодший бакалавр»
студенту Давлетову Ренату Рустамовичу

(прізвище, ім’я по-батькові студента)

1. Тема роботи: __Програмний засіб для захищеного обміну даними з корекцією помилок._____

_____

затверджена наказом по коледжу від “25” листопада 2024р., №253а-н

2. Термін здачі студентом завершеної роботи “13” червня 2025р.

3. Вихідні дані до роботи _нормативні документи та стандарти з інформаційної безпеки; методи забезпечення конфіденційності та цілісності даних; технології та інструменти шифрування і аналізу даних, теоретичні основи механізмів виявлення та корекції помилок; технічні характеристики засобів реалізації.____

4. Перелік питань, які повинні бути розроблені:

а) основна частина _аналіз актуальності та існуючих механізмів захисту даних; дослідження методів забезпечення цілісності даних та коригувальних кодів; розробка алгоритмів шифрування, дешифрування та корекції помилок із використанням коду Хеммінга на основі модульної арифметики; реалізація та тестування програмного засобу з інтуїтивно зрозумілим графічним інтерфейсом користувача._____

б) техніко-економічного обґрунтування _аналіз ринку програмних засобів для захищеного обміну даними; оцінка витрат на розробку та впровадження програмного продукту; обґрунтування економічної доцільності створення програмного засобу з урахуванням потреб користувачів і сучасних технологій._

5. Перелік графічного матеріалу_____

_1. Схема алгоритму шифрування даних._____

_2. Схема алгоритму дешифрування даних._____

_3. Головне вікно програмного засобу._____

6. Консультанти проекту: _____

Розділ	Консультанти	Підпис, дата	
		Завдання видано	Завдання прийнято
з техніко-економічного обґрунтування	_____ (вчена ступень, звання П.І.Б. _____ консультанта)	25.11.2024	

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

№ п/п	Найменування етапу	Терміни	
		початку	завершення
1	Вибір теми, ознайомлення з вимогами до кваліфікаційної роботи	20.11.2024	25.11.2024
2	Аналіз предметної області, огляд механізмів захисту та цілісності даних, написання відповідного розділу ПЗ	25.11.2024	04.12.2024
3	Дослідження механізмів виявлення та корекції помилок кодів Хеммінга та написання відповідного розділу ПЗ	05.12.2024	07.12.2024
4	Формування вимог до програмного засобу та проєктування його структури. Написання відповідного розділу ПЗ	08.12.2024	12.12.2024
5	Встановлення і налаштування середовища розробки, створення базових модулів	13.12.2024	14.12.2024
6	Проєктування архітектури програмного засобу, розробка алгоритмів шифрування та дешифрування	15.12.2024	26.12.2024
7	Реалізація програмного засобу, створення графічного інтерфейсу та модулів обробки даних	27.12.2024	26.01.2025
8	Доопрацювання функціональних модулів, забезпечення стабільної роботи	27.03.2025	02.04.2025
9	Тестування та налагодження програмного засобу, написання відповідного розділу	03.04.2025	30.04.2025
10	Підготовка спеціального розділу з питань техніко-економічного обґрунтування проєкту	1.05.2025	5.05.2025
11	Оформлення пояснювальної записки	06.05.2025	13.06.2025
12	Попередній захист, коригування роботи згідно з зауваженнями	13.06.2025	13.06.2025
13	Підготовка до захисту (презентація, оформлення, роздруківка, тренування)	14.06.2025	24.06.2025
14	Захист кваліфікаційної роботи	25.06.2025	25.06.2025

7. Дата видачі завдання “____” _____ 2024 р.

Керівник _____ / Степан ІВАСЬЄВ

Завдання прийняв до виконання_____ / Ренат ДАВЛЕТОВ

Реферат

Програмний засіб для захищеного обміну даними з корекцією помилок. Кваліфікаційна робота. Давлетов Ренат Рустамович. Галицький фаховий коледж імені В'ячеслава Чорновола, відділення комп'ютерних технологій. Спеціальність 122 «Комп'ютерні науки», 2025. Сторінок – 78, рисунків – 52, таблиць – 10, додатків – 1.

Об'єкт дослідження - процес забезпечення захищеної передачі даних у інформаційно-комунікаційних системах.

Мета роботи – створення програмного засобу, що забезпечує захищений обмін даними з виявленням та виправленням помилок під час передачі.

Завданням проекту є розробка алгоритму шифрування з використанням коригувальних кодів та реалізація користувацького інтерфейсу для зручної роботи з зашифрованими повідомленнями або файлами.

Результат – програмний продукт, що дозволяє здійснювати шифрування, дешифрування, виявлення та корекцію одиничних помилок у переданих повідомленнях. Реалізовано підтримку символів кирилиці та латиниці, обробку як текстових, так і файлових даних. Розроблено алгоритми роботи застосунку, що використовують побудову коду Хеммінга на основі модульної арифметики.

Для реалізації програмного засобу використано мову програмування Python та бібліотеку Tkinter для створення графічного інтерфейсу користувача.

ЗАХИСТ ІНФОРМАЦІЇ, ШИФРУВАННЯ, КОРЕКЦІЯ ПОМИЛОК, ПЕРЕДАЧА ДАНИХ, МОДУЛЬНА АРИФМЕТИКА, МОВА ПРОГРАМУВАННЯ PYTHON.

Abstract

Software Tool for Secure Data Exchange with Error Correction. Qualification Thesis. Davletov Renat Rustamovych. Halytskyi Vocational College named after Viacheslav Chornovil, Department of Computer Technologies. Specialty 122 “Computer Science”, 2025. Pages – 78, Figures – 52, Tables – 10, Appendices – 1.

The object of research is the process of ensuring secure data transmission in information and communication systems.

The purpose of the work is to develop a software tool that provides secure data exchange with error detection and correction during transmission.

The task of the project is to design an encryption algorithm using error-correcting codes and to implement a user interface for convenient interaction with encrypted messages or files.

The result is a software product that enables encryption, decryption, detection, and correction of single-bit errors in transmitted messages. Support for Cyrillic and Latin characters has been implemented, along with the processing of both text and file data. Algorithms used in the application are based on Hamming code construction with modular arithmetic.

The software tool was implemented using the Python programming language and the Tkinter library to build a graphical user interface.

INFORMATION SECURITY, ENCRYPTION, ERROR CORRECTION, DATA TRANSMISSION, MODULAR ARITHMETIC, PYTHON PROGRAMMING LANGUAGE.

ЗМІСТ

					КР.КН.25.591.09.000 ПЗ		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Давлетов Р.Р.			Програмний засіб для захищеного обміну даними з корекцією помилок	Літ.	Арк. 5 / Акрушів 78
Перевір.		Івасьєв С.В.					
Рецецнз.		Кульчинська Н.З.				ГФК.ВКТ.КН-41	
Н. Контр.		Гавришків Н.Г.					
Затверд.		Стефурак Н.А.					

СКОРОЧЕННЯ І УМОВНІ ПОЗНАКИ

AES - Advanced Encryption Standard;

DES - Data Encryption Standard;

ECC - Elliptic Curve Cryptography;

RSA - Rivest-Shamir-Adleman;

SSL - Secure Sockets Layer;

TLS - Transport Layer Security;

VPN - Virtual Private Network;

КЗ - канал зв'язку;

ККП - коди з корекцією помилок;

НСД - несанкціонований доступ.

ВСТУП

Передача даних у цифровому вигляді є основою багатьох сучасних процесів, що охоплюють різні сфери життя – від повсякденної комунікації в соціальних мережах до обміну офіційними документами між державними установами. Інформаційні технології відіграють важливу роль у здійсненні фінансових операцій, що, в свою чергу, сприяє розвитку бізнесу та економіки в цілому. Особливого значення набули інформаційно-телекомунікаційні технології в рамках процесів діджиталізації. Обмін цифровими даними в цьому контексті здійснюється через національні портали, які надають громадянам доступ до державних послуг. Такі платформи дозволяють забезпечити зручний та ефективний доступ до адміністративних сервісів, спрощують процедури та підвищують прозорість в роботі державних установ.

З кожним роком обсяги переданих даних зростають, що підвищує ризики, пов'язані з їх безпекою. Інформація стає вразливою до загроз, зокрема НСД, маніпуляцій та втрати під час передавання. У таких умовах одним із пріоритетних завдань є реалізація та застосування технологій захисту, які забезпечують не лише конфіденційність, а й цілісність переданих даних. Для вирішення задачі підвищення надійності комунікацій застосовуються алгоритми шифрування та властивості коригуючих кодів, які забезпечують захист від зовнішніх впливів та зменшують ймовірність помилок під час обміну даними.

Постійне зростання кіберзагроз, складність забезпечення захисту інформації в умовах швидкого розвитку цифрових технологій та необхідність адаптації систем безпеки до нових викликів зумовлюють необхідність впровадження ефективних рішень для захисту даних. Це визначає потребу в розробці засобів для захищеного обміну даними, що є основою для функціонування сучасних бізнес-процесів і надання державних послуг. Таким чином, обрана тема роботи, орієнтована на забезпечення безпеки та корекції помилок при передачі даних, є актуальною в умовах активного розвитку

цифрових систем і зростаючих загроз кібербезпеки.

Метою даної роботи є дослідження методів забезпечення безпеки та розробка програмного засобу для надійного і швидкого обміну інформацією з корекцією помилок.

Об'єктом дослідження є процеси шифрування та виправлення помилок при передачі цифрових даних.

Предметом дослідження є засоби, що використовуються для забезпечення конфіденційності та цілісності інформації під час її передачі через інформаційно-комунікаційні канали.

Досягнення поставленої мети передбачає виконання наступних завдань:

− аналіз методів захисту від несанкціонованого доступу для забезпечення конфіденційності даних;

− аналіз механізмів забезпеченні цілісності даних;

− дослідження властивостей кодів для виявлення та виправлення помилок, які можуть виникнути під час передачі даних;

− дослідження механізмів виявлення та корекції помилок коду Хеммінга;

− розробка алгоритмів кодування та декодування даних для роботи проєктованого програмного засобу;

− створення користувацького інтерфейсу для зручного використання програмного засобу;

− тестування ефективності та надійності розробленого програмного засобу;

− виконання техніко-економічного обґрунтування роботи.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність розробки

Передача даних у цифровому вигляді є основою багатьох сучасних процесів, від комунікації у соціальних мережах до державних електронних сервісів. Відкриті канали зв’язку (КЗ), такі як Інтернет або інші публічні мережі, часто стають об'єктом різноманітних атак, таких як перехоплення, модифікація або втрата даних. Це робить необхідним використання технологій шифрування для захисту інформації під час її передачі, а також застосування інструментів для забезпечення цілісності та коректності даних, що передаються.

Сучасні організації обмінюються великою кількістю даних, що вимагає інтеграції надійних методів захисту від несанкціонованого доступу (НСД) та корекції помилок. Застосування таких методів дає змогу не тільки забезпечити конфіденційність дані, але й забезпечити їх цілісність, навіть у випадку часткових помилок під час передачі.

У реальних умовах передача даних повинна бути водночас швидкою і безпечною. Застосування систем шифрування дозволяє захистити інформацію, а коригуючі коди дозволяють виявляти і виправляти помилки, що можуть виникнути під час передавання даних. Такий підхід дозволяє досягти високого рівня надійності при збереженні швидкості передачі.

У сучасних інформаційних мережах важливо забезпечити не тільки захист від зовнішніх загроз, але й підвищити ефективність і надійність процесів обміну та зберігання інформації, що включають в себе як захист від атак, так і корекцію помилок, що виникають через технічні проблеми чи обмеження КЗ.

Постійне вдосконалення програмних засобів для шифрування та корекції помилок є необхідністю для забезпечення безпеки в умовах сучасних кіберзагроз. Такі засоби можуть бути використані в різних галузях, від фінансових організацій до медичних установ, де важливо забезпечити захист

чутливої інформації, її точність і повноту під час обміну. Тому реалізація програмного засобу що інтегрує шифрування даних та корекції помилок, стане важливим кроком у забезпеченні безпеки даних в організаціях.

1.2 Аналіз механізмів захисту від несанкціонованого доступу

Існує широке коло ефективних інструментів для захисту даних від НСД як під час їх передачі, так і під час зберігання. В реальних умовах їх застосування залежить від конкретного контексту, вимог безпеки та рівня загроз.

Механізми контролю доступу - це системи, що дозволяють визначити, хто має право доступу до певних ресурсів або інформації. Обмеження на основі ролей користувачів, де кожна роль має відповідні привілеї для доступу до певних ресурсів, або на основі адрес дозволяють мінімізувати ризики від зловмисного доступу.

Механізми аутентифікації - застосовуються для перевірки особи користувача або системи перед наданням доступу до ресурсів. Це може бути як традиційна аутентифікація за допомогою логіна та пароля, так і більш складніші методи, такі як двофакторна та багатофакторна аутентифікація або аутентифікація на основі біометричних даних.

Цифрові підписи - використовуються для забезпечення автентичності та цілісності переданих даних. Вони підтверджують, що дані не були змінені в процесі їх передачі, і що вони походять від автора, якого зазначено в підписі.

Системи виявлення та запобігання вторгненням - призначені для моніторингу та аналізу мережевих з'єднань і виявлення підозрілих або несанкціонованих спроб доступу до переданих даних. Вони можуть блокувати або сигналізувати про спроби доступу під час передачі інформації через мережу.

Мережеві брандмауери - захищають систему від неконтрольованого доступу, контролюючи і фільтруючи мережевий трафік, що проходить між користувачем і внутрішніми мережами або між різними мережами.

Перелічені механізми є основою захисту від НСД в сучасних інформаційних системах і сприяють забезпеченню конфіденційності, цілісності та доступності даних.

Одним із основних методів, що гарантує конфіденційність інформації при передачі є шифрування даних:

– Шифрування - використовуються для захисту даних, що передаються через відкриті КЗ, наприклад, в мережі Інтернет. Технології TLS (Transport Layer Security) та SSL (Secure Sockets Layer) забезпечують зашифроване з'єднання між клієнтом та сервером, що унеможливлює перехоплення і модифікацію даних під час передачі (рисунок 1.1) [1].

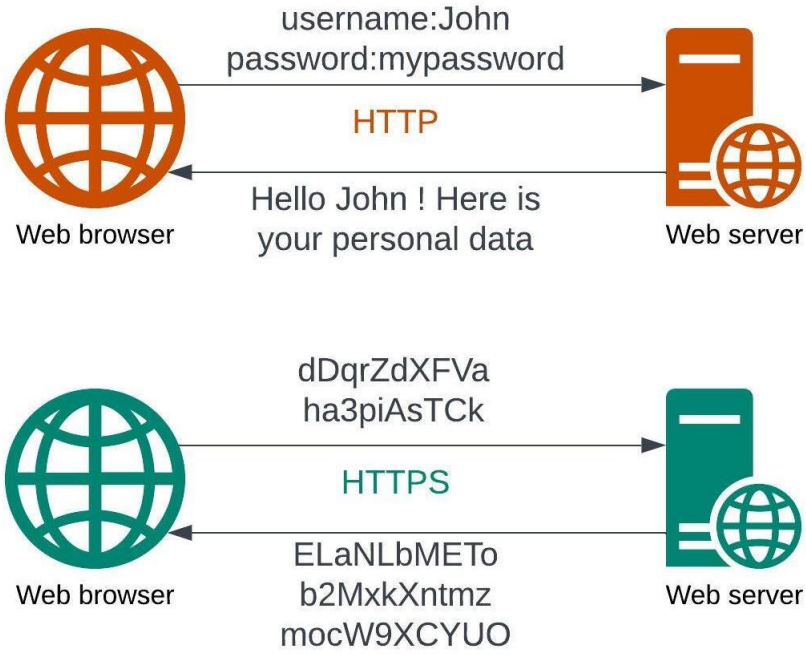

Рисунок 1.1 – Принцип роботи SSL/TLS

– VPN (Virtual Private Network) - створює зашифроване з'єднання між користувачем і віддаленим сервером, забезпечуючи захист від неконтрольованого доступу до даних при їх передачі через незахищені мережі [2]. VPN дозволяє зберегти конфіденційність і цілісність інформації при передачі через публічні КЗ (рисунок 1.2).

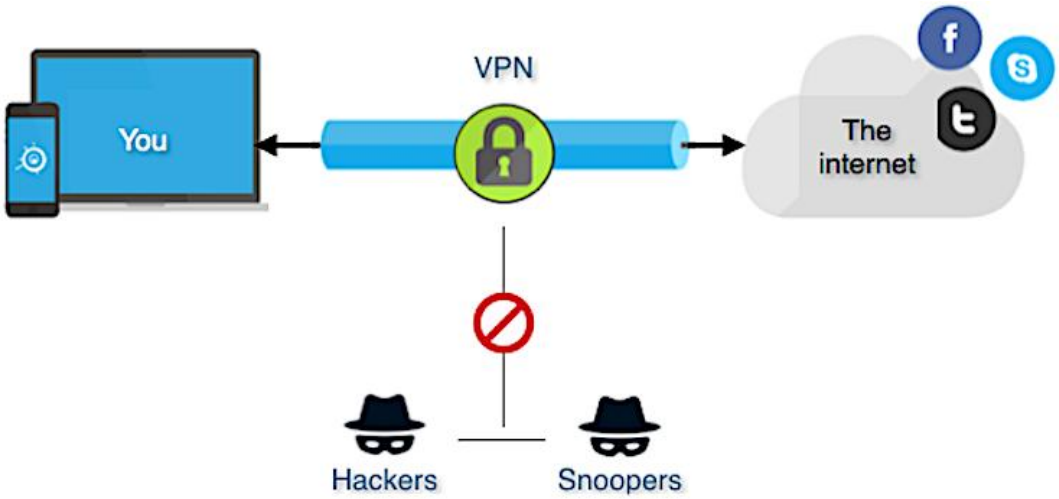

Рисунок 1.2 - Принцип роботи VPN

– Симетричне шифрування - метод шифрування, при якому один і той самий ключ використовується для шифрування та дешифрування даних [3]. Симетричне шифрування швидше за асиметричне і зазвичай застосовується для великих обсягів даних, наприклад, у протоколах передачі файлів або в системах резервного копіювання (рисунок 1.3).

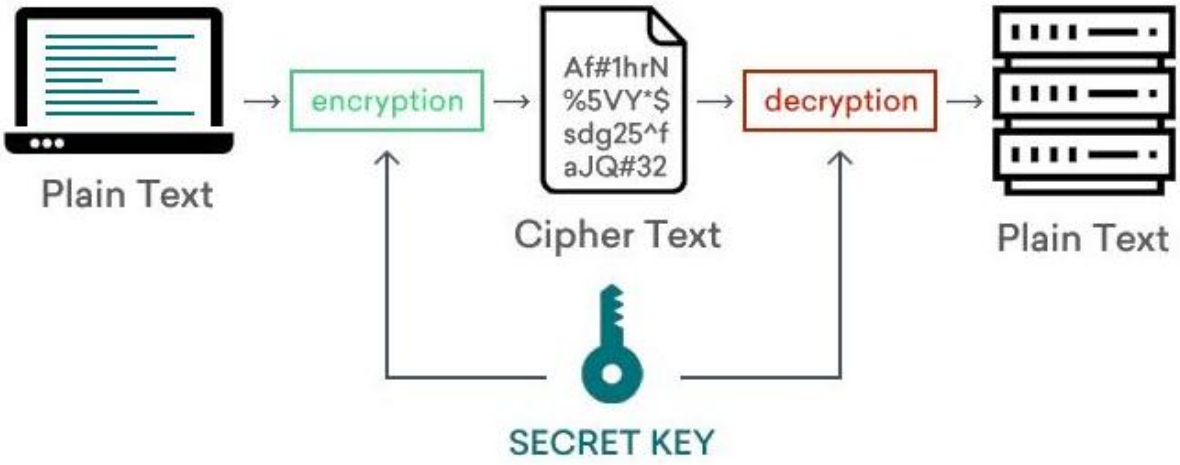

Рисунок 1.3 - Принцип роботи симетричного шифрування

– Асиметричне шифрування - використовуються для шифрування даних в процесі їх передачі [4]. Алгоритми асиметричного шифрування дозволяють забезпечити захист інформації за допомогою публічного ключа, що унеможливлює її дешифрування без наявності відповідного приватного ключа (рисунок 1.4).

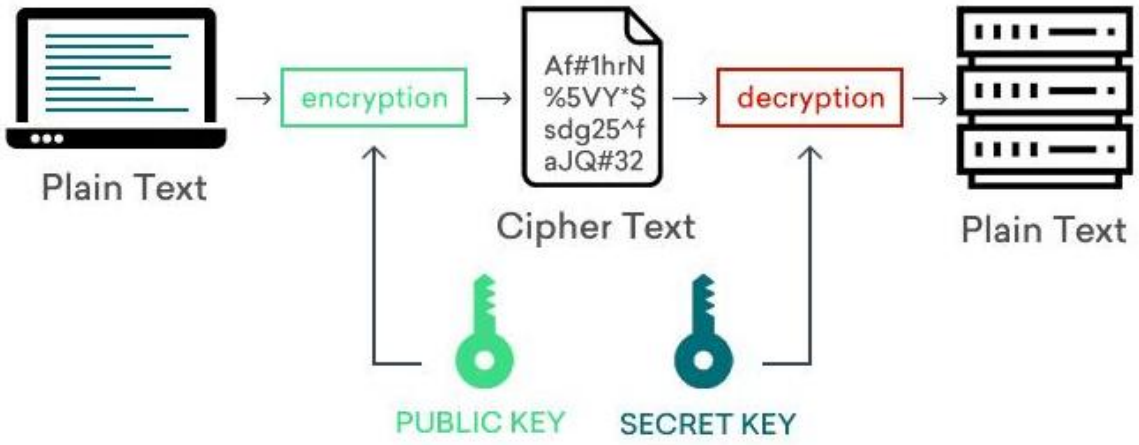

Рисунок 1.4 - Принцип роботи асиметричного шифрування

Для зберігання даних використовуються наступні методи:

– шифрування даних на дисках, наприклад, AES, для забезпечення їх конфіденційності при зберіганні на пристроях;

– резервне копіювання та відновлення даних - забезпечує збереження важливої інформації та можливість її відновлення в разі пошкодження або втрати;

– механізми контролю доступу - дозволяють визначити, хто має доступ до конкретних даних на різних рівнях зберігання;

– механізми аудитів і журналів - використовуються для моніторингу доступу і змін у даних, що допомагає у виявленні неузгоджених дій.

Серед існуючих алгоритмів захисту від несанкціонованого проникнення для забезпечення конфіденційності даних найбільш широко використовується симетричне та асиметричне шифрування.

Алгоритм AES (Advanced Encryption Standard) використовує шифрування з симетричним ключем (рисунок 1.5) [5-7]. Принцип його роботи використовує один ключ для шифрування та дешифрування даних.

Перевагою такого алгоритму є висока швидкість, ефективність при обробці великих обсягів даних. Недоліком є проблеми з безпекою при зберіганні ключів, необхідність передавати ключи через захищені КЗ.

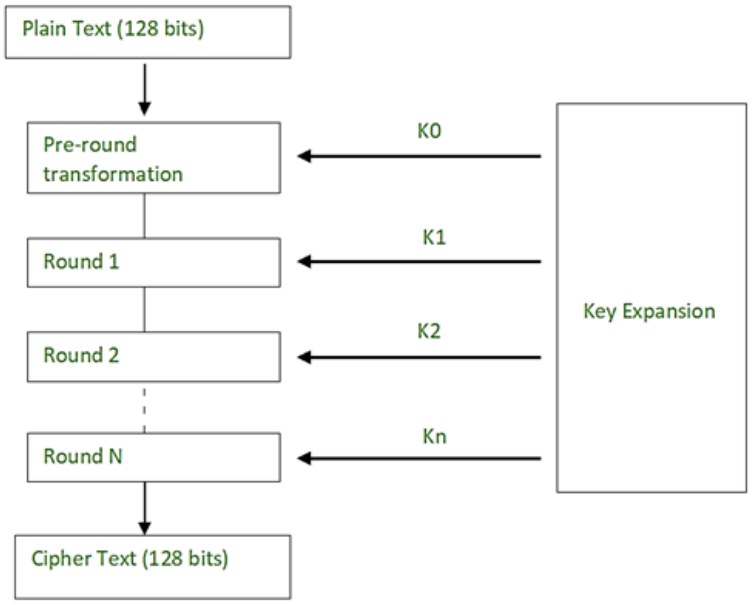

Рисунок 1.5 – Алгоритм шифрування AES

Ще одним симетричним алгоритмом є DES (Data Encryption Standard), використовує 56-бітний ключ для шифрування даних (рисунок 1.5) [8-10]. Серед переваг DES варто відзначити простоту його реалізації. Проте значним недоліком є недостатній рівень безпеки через короткий ключ (піддається атакам типу "перебір").

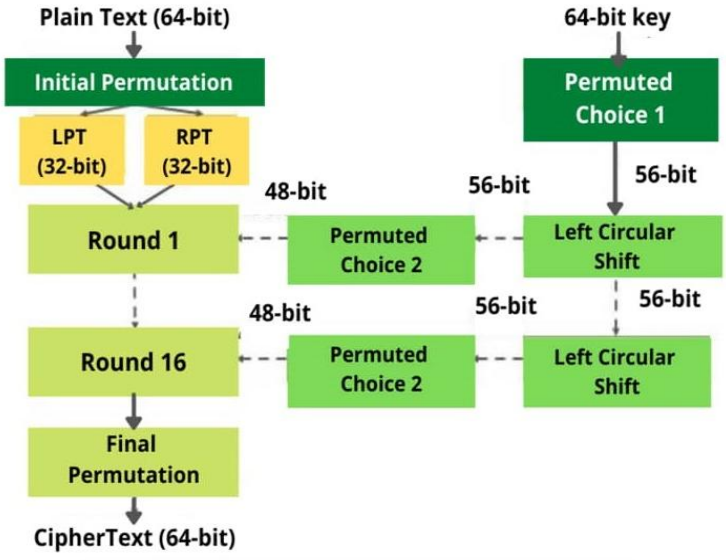

Рисунок 1.6 - Шифрування даних DES

AES є набагато більш безпечним і ефективним, ніж DES, і є стандартом для шифрування даних у багатьох організаціях.

RSA (Rivest-Shamir-Adleman) є алгоритмом шифрування з асиметричним ключем [11-14]. Принцип роботи використовує пару ключів - публічний та приватний. Публічний ключ шифрує дані, а лише власник приватного ключа може їх дешифрувати (рисунок 1.7). Переваги включають безпеку для обміну ключами через відкриті КЗ. Недоліком є те, що алгоритм повільніший за симетричні алгоритми, великі обсяги даних потребують додаткового шифрування.

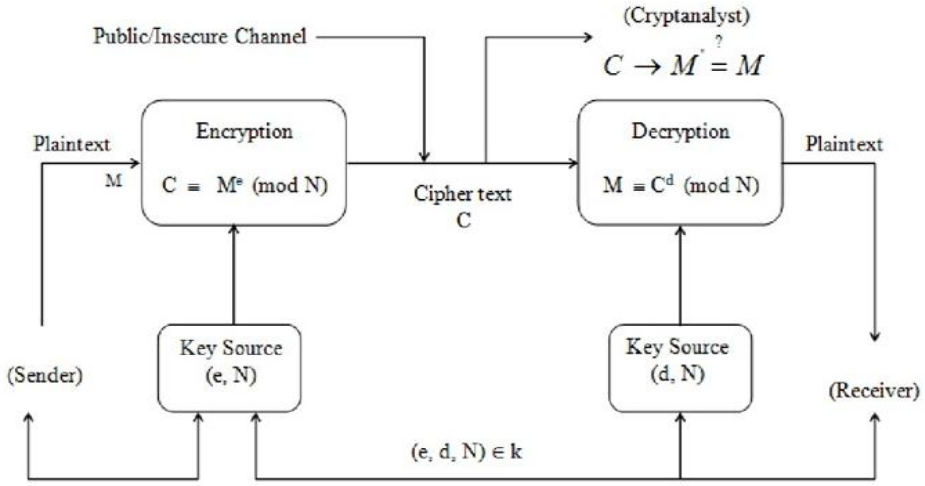

Рисунок 1.7 - Криптографія з відкритим ключем RSA

Асиметричні алгоритми ECC (Elliptic Curve Cryptography) використовують математику еліптичних кривих (рисунок 1.8) для забезпечення безпеки, як і RSA, але з меншою довжиною ключа для досягнення того ж рівня безпеки [15-17]. Їх переваги полягають у вищій безпеці при меншому обсязі ключа та більш швидкому шифруванні. Проте недоліком є те, що потрібен спеціалізований апарат для ефективного виконання операцій.

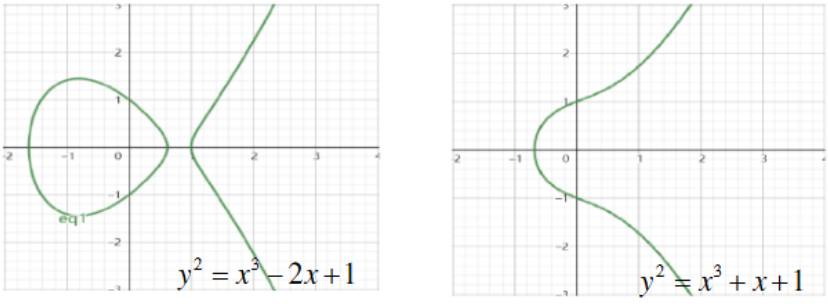

Рисунок 1.8 – Приклад еліптичних кривих

RSA є стандартом для публічних ключів, але ECC забезпечує більшу ефективність і безпеку, особливо в мобільних пристроях та IoT.

Менш поширеними є алгоритми , що використовують гібридні методи шифрування. В такому випадку початкове шифрування даних реалізоване за допомогою симетричного ключа, а потім передача ключа через асиметричне шифрування (наприклад, в протоколі SSL/TLS). Принцип роботи полягає у тому, що симетричний ключ використовується для швидкого шифрування великих обсягів даних, а асиметричний - для безпечного обміну ключем.

Перевагою таких механізмів є поєднання швидкость симетричних методів і безпеку асиметричних. Недоліком - необхідність в підтримці обох методів шифрування.

Ще одним механізмом захисту даних є квантове шифрування, що ґрунтується на принципах квантової суперпозиції та сплутаності. Його принцип роботи використовує квантову механіку для захисту даних. Теоретично такий алгоритм є абсолютно безпечним, оскільки будь-яка спроба перехопити зашифровану інформацію змінює її. Проте для його реалізації потрібне значне апаратне забезпечення і поки що є дорогим і експериментальним.

Хоча існуючі механізми захисту даних, такі як шифрування (AES, RSA, ECC) та протоколи безпеки (TLS/SSL, VPN), забезпечують високу конфіденційність і цілісність даних, вони мають кілька суттєвих недоліків. По-перше, ці механізми часто потребують значних обчислювальних ресурсів, особливо при використанні складних алгоритмів, таких як RSA, що може бути проблемою при передачі невеликих обсягів даних або в умовах обмежених ресурсів (наприклад, на мобільних пристроях або в IoT-системах). Це може знижувати ефективність системи, збільшуючи затримки та енергоспоживання.

По-друге, більшість традиційних механізмів шифрування не забезпечують корекцію помилок, що може призвести до втрати даних або зниження їх достовірності в разі пошкодження під час передачі через неякісні КЗ. Такі системи не здатні автоматично виявляти та виправляти

помилки, що робить передачу даних у таких системах вразливою до збоїв.

У зв'язку з цим, розробка програмного засобу для безпечної передачі даних, який поєднує шифрування з методами корекції помилок є важливою для забезпечення надійної, ефективної та безпечної передачі даних. Такий засіб дозволить підвищити як цілісність, так і надійність передачі даних, зменшуючи ризики помилок і зниження продуктивності при обміні інформацією в умовах з обмеженими ресурсами.

1.3 Аналіз механізмів забезпеченні цілісності даних

У рамках дослідження та розробки програмного засобу для безпечного обміну інформацією на основі коригуючих кодів варто звернути увагу на існуючі програмні рішення в цій галузі. Оскільки система повинна поєднувати як методи шифрування для забезпечення конфіденційності, так і алгоритми корекції помилок для забезпечення надійності переданих даних, розглянемо кілька програмних засобів, що вже реалізують подібні функції.

PGP (Pretty Good Privacy) це криптографічний програмний пакет призначений для шифрування та підписування електронних повідомлень і файлів [18]. Він забезпечує конфіденційність даних, їх цілісність та автентичність за допомогою криптографічних методів (рисунок 1.9).

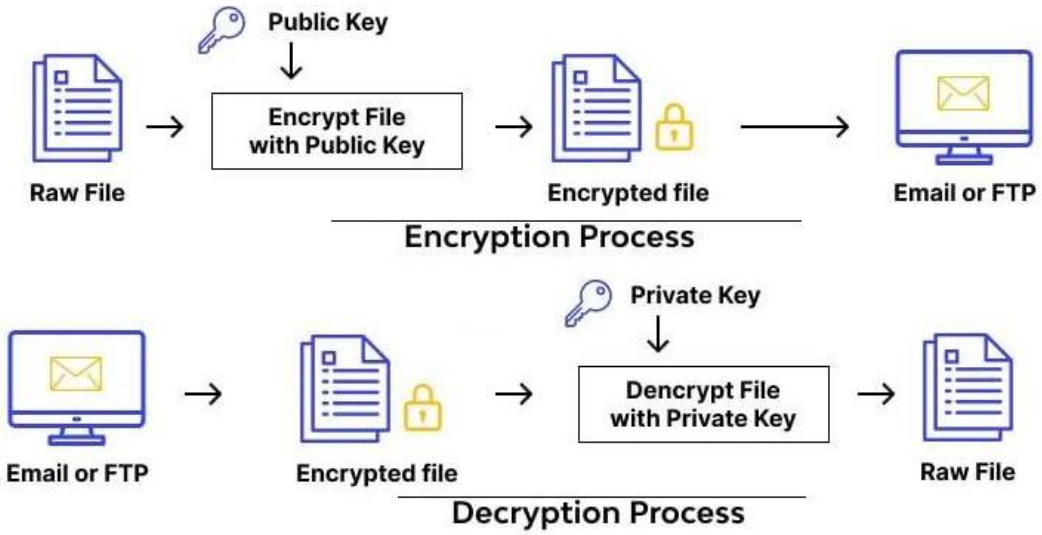

Рисунок 1.9 – Шифрування та дешифрування PGP

Інтерфейс забезпечується командним рядком та графічним інтерфейсом

користувача (GUI). Взаємодія з іншими програмами реалізована через розширення для електронної пошти (наприклад, Outlook, Thunderbird).

Переваги PGP полягають у наступному:

− підтримка сильних алгоритмів шифрування (RSA, AES);

− використання відкритих ключів для зручності обміну ключами;

− підтримка інтеграції з різними платформами і програмами для шифрування електронної пошти;

− можливість використання цифрових підписів для забезпечення автентичності повідомлень.

До недоліків можна віднести:

− складність налаштування для нових користувачів;

− підвищену потреба в ресурсах для обробки великих файлів;

− застарілий підхід до управління ключами в порівнянні з новими рішеннями.

OpenSSL є відкритою бібліотекою криптографічних алгоритмів і інструментом командного рядка для реалізації шифрування, цифрових підписів і управління сертифікатами [19]. Вона підтримує протокол TLS/SSL, що використовується для захищеного передавання даних в інтернеті. OpenSSL не містить вбудованих механізмів корекції помилок, однак її криптографічні механізми сприяють виявленню та запобіганню змін у переданих даних (рисунок 1.10).

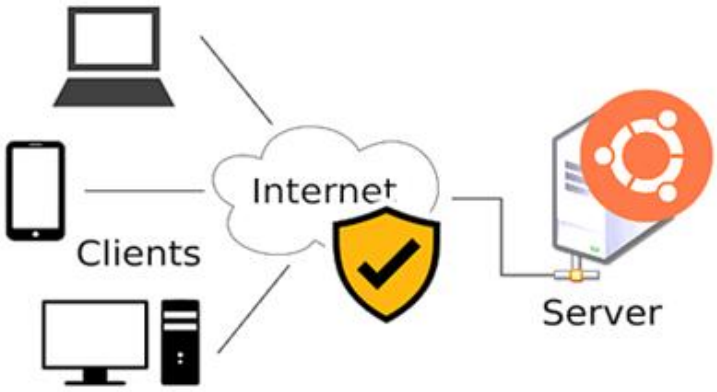

Рисунок 1.10 – Використання OpenSSL

Переваги використання OpenSSL включають:

– підтримку великої кількості криптографічних алгоритмів;

– широке застосування в різних сферах, включаючи веб-безпеку;

– підтримку багатьох протоколів для безпечної передачі даних.

Серед недоліків можна зазначити:

– складність в налаштуванні та використанні, особливо для недосвідчених користувачів;

– відсутність функцій для безпосередньої корекції помилок у процесі передачі даних;

– застарілий інтерфейс командного рядка.

Zlib – бібліотека, що використовується для стиснення переданих даних без втрат у криптографічних протоколах [20]. Стиснення дозволяє зменшити обсяг переданих даних що може знизити ймовірність помилок у зашумлених КЗ (рисунок 1.11).

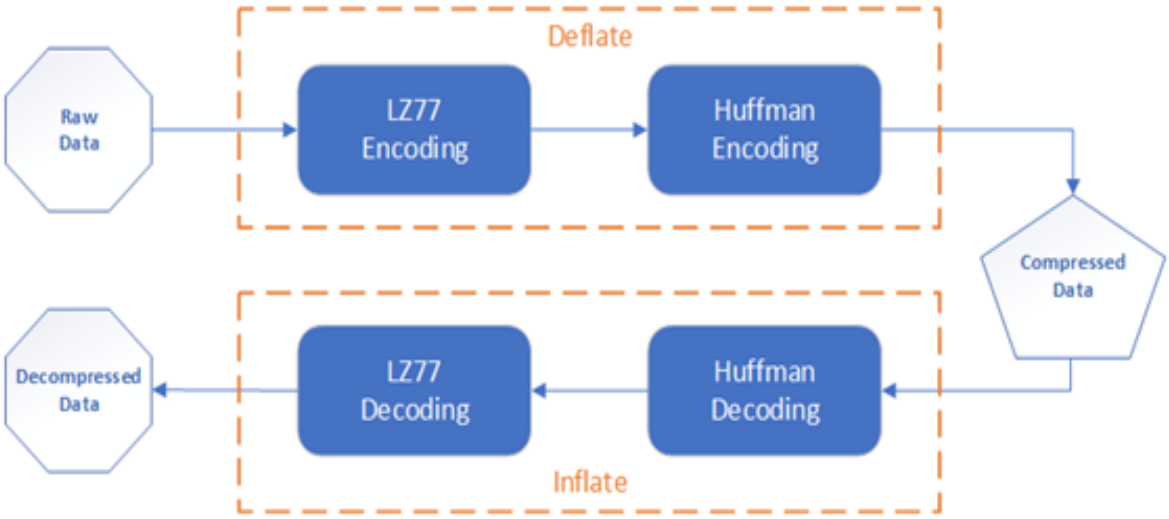

Рисунок 1.11 – Принцип роботи Zlib

Zlib дозволяє забезпечити перевірку цілісності переданої/збереженої інформації. Вона виявляє помилки за допомогою контрольної суми CRC-32, що допомагає перевірити правильність даних після стиснення або передачі. Проте не виконує виправлення помилок, тобто не має коригуючих властивостей. Програмне API для стиснення та розпакування даних,

використовується в багатьох системах для обробки великих обсягів даних. Перевагами Zlib є:

- висока швидкість стиснення і розпакування;
- простота інтеграції з іншими програмними засобами;
- відкритий вихідний код та широке використання у багатьох додатках.

Недоліками:

- відсутність вбудованої підтримки коригуючих кодів для виявлення та виправлення помилок;
- не забезпечує жодного рівня шифрування або захисту даних.

Wireshark - програмний засіб призначений для аналізу мережевого трафіку, виявлення проблем у передачі даних, включаючи помилки, затримки та інші аномалії [21].

Графічний інтерфейс користувача забезпечує фільтрацію і глибокий аналіз пакетів даних (рисунок 1.12).

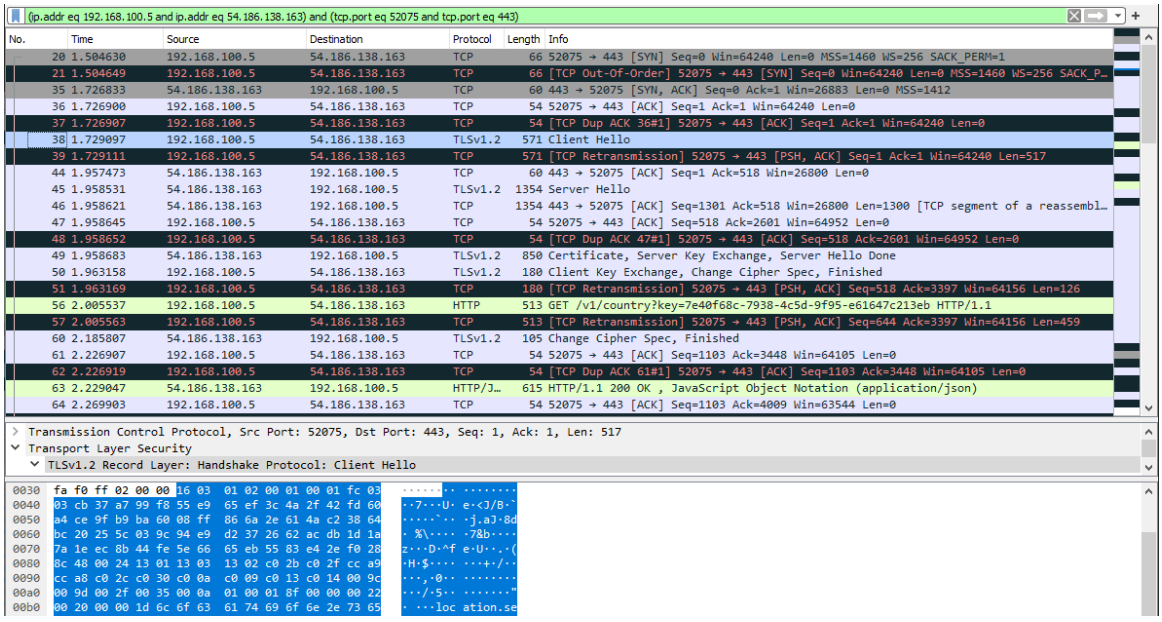

Рисунок 1.12 – Графічний інтерфейс Wireshark

Wireshark не виконує корекцію помилок – він лише аналізує трафік і може виявляти помилки та аномалії у переданих пакетах та виявляти перевідправлення пакетів (retransmissions) у випадку помилок передачі.

Перевагами аналізатора пакетів є :

− потужні інструменти для аналізу мережевого трафіку;

− можливість виявляти помилки в передачі даних на рівні мережі;

− підтримка багатьох мережевих протоколів для аналізу.

Недоліки Wireshark щодо забезпечення цілісності даних:

− не є засобом для безпосереднього коригування помилок або шифрування даних;

− потребує специфічних знань для правильного використання.

З наведеного аналізу видно, що існуючі рішення пропонують різні підходи до забезпечення безпеки та надійності передачі даних, але жодне з них не поєднує захист від НСД з методами корекції помилок в одному інтерфейсі. Розробка програмного засобу, що поєднує ці два аспекти є доцільною, оскільки вона дозволяє забезпечити надійність передачі даних при збереженні конфіденційності.

1.4 Постановка завдань

Проведений аналіз дозволяє визначити, що основними завданнями, які необхідно вирішити є забезпечення цілісності та безпеки даних при їх передачі.

Традиційні криптографічні методи захищають інформацію від НСД, але вони не гарантують її відновлення у разі пошкодження через збої або перешкоди в КЗ. З іншого боку, інструменти для контролю цілісності інформації не завжди забезпечують їх шифрування. Тому вирішення даної задачі вимагає комплексного підходу, що включає не лише використання методів шифрування, але й ефективних алгоритмів для виявлення і виправлення помилок.

Застосування кодів для виправлення помилок та їх коригуючих властивостей дозволить досягти високого рівня надійності та безпеки, що є важливим для забезпечення стабільної та безпечної роботи інформаційних систем.

Актуальність розробки програмного засобу полягає в необхідності

інтеграції методів криптографічного захисту та корекції помилок, що дозволяє одночасно вирішити дві проблеми:

− конфіденційність даних – за рахунок використання криптографічних операцій та модульної арифметики;

− цілісність інформації – за допомогою кодів Хеммінга, що дозволяють виправляти одиничні помилки.

Запропонована розробка дозволить забезпечити безпечний обмін даними, наприклад, у фінансових системах, корпоративних комунікаціях, тощо, а також зберігання конфіденційної інформації із можливістю відновлення після часткового пошкодження.

Для розробки програмного засобу безпечного обміну інформацією на основі коригуючих кодів необхідно чітко визначити вимоги до його функціональних можливостей, складу підсистем та модулів, а також категорії користувачів. Оскільки програма повинна поєднувати шифрування та корекцію помилок, важливо забезпечити високу надійність і зручність для кінцевих користувачів.

Функціональні вимоги програмного засобу включають:

− шифрування даних –захист даних від НСД, для забезпечення конфіденційності переданої інформації;

− обробка помилок – безпомилкове виявлення та виправлення спотворень за допомогою алгоритмів корекції помилок;

− декодування – коректне відновлення отриманих даних до їх початкового вигляду;

− збереження та завантаження даних - можливість збереження даних, а також доступ до них при необхідності;

− інтерфейс користувача – зручний вибір режиму роботи, завантаження повідомлень і файлів, а також можливість збереження зашифрованих та розшифрованих даних.

Нефункціональні вимоги програмного засобу включають:

− надійність - засіб повинен гарантувати високий рівень

надійності передачі даних, зокрема за рахунок ефективної корекції помилок;

− швидкість роботи - шифрування та дешифрування повинні виконуватися швидко, з урахуванням обмежень ресурсів, особливо при роботі з великими обсягами даних;

− безпека - засіб повинен використовувати алгоритми, які забезпечують високий рівень безпеки, щоб запобігти НСД до даних;

− масштабованість - засіб повинен працювати з різними розмірами даних, від невеликих повідомлень до великих файлів;

− сумісність - програмний засіб повинен бути сумісний з різними операційними системами та платформами, підтримувати різні розміри файлів для шифрування/декодування.

2. ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАСОБУ

2.1 Дослідження властивостей коригуючих кодів

Зі зростанням обсягу та важливості передаваних даних, забезпечення їхньої безпеки стає критичним завданням. Атаки на інформаційні системи стають дедалі складнішими та численнішими, що вимагає впровадження надійних методів захисту. Застосування криптографічних перетворень дозволяє значно підвищити надійність і стійкість інформаційних систем, зменшуючи ризик втрати або спотворення даних. Коди з корекцією помилок та криптографічні алгоритми є основними інструментами для забезпечення цілісності, конфіденційності та автентичності даних під час їхньої передачі.

Коди з корекцією помилок (ККП), які також називають завадостійкими, є важливою складовою сучасних комунікаційних систем [23]. Їх класифікація є складною та може бути виконана за різними критеріями, наприклад за типом кодування, методом виявлення помилок, здатністю до їх виправлення, структурою коду, складністю декодування тощо. На рисунку 2.1 наведена узагальнена класифікація ККП.

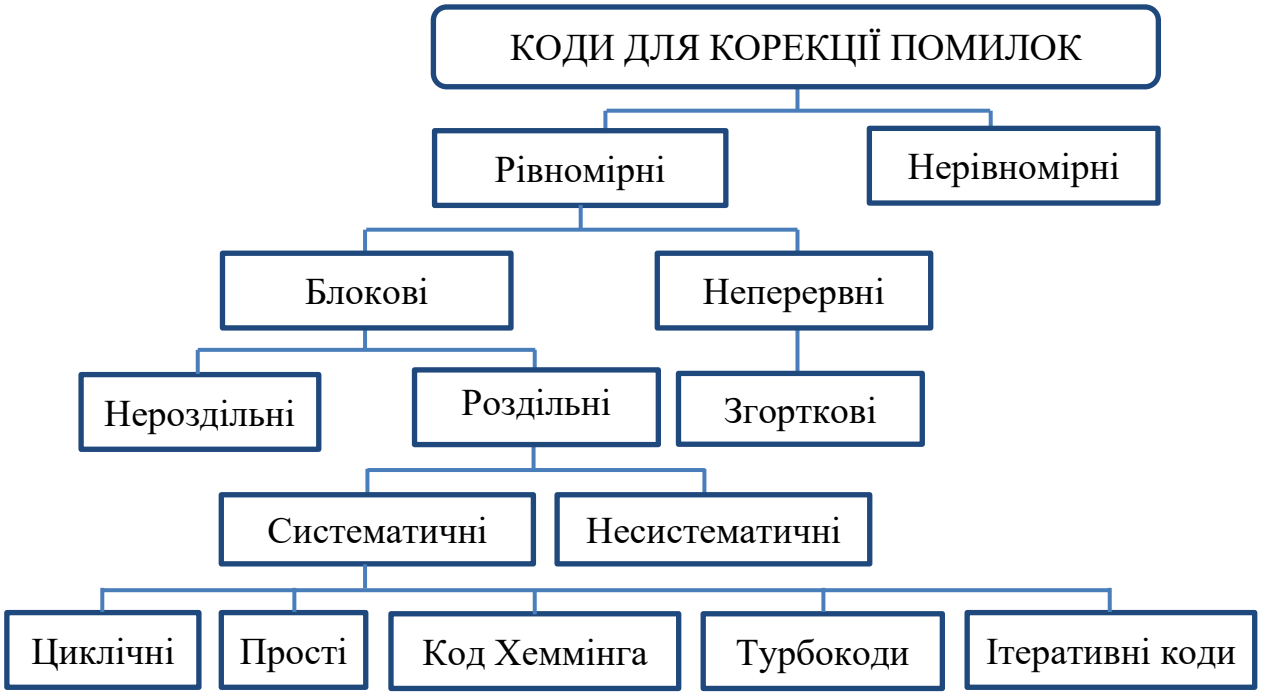

Рисунок 2.1 - Класифікація коригуючих кодів

ККП широко застосовуються в різних галузях для забезпечення надійності передачі даних через мережі, мінімізації впливу перешкод і шумів каналів передачі, а також для збереження цілісності даних на дисках і флеш-накопичувачах. Це обумовлене наступними причинами:

1. Надійність передачі даних. В умовах шуму та перешкод, які характерні для більшості КЗ (радіозв'язок, інтернет, мобільні мережі), ККП дозволяють виявляти та виправляти помилки, що виникають під час передачі.

2. Забезпечення цілісності даних. Під час зберігання даних на різних носіях можуть виникати помилки зчитування. ККП допомагають виявити та виправити ці помилки.

3. Підвищення ефективності систем. Використання ККП дозволяє зменшити потребу у повторній передачі даних у разі виникнення помилок, що підвищує загальну пропускну здатність систем передачі даних.

4. Забезпечення безпеки даних. ККП можуть також використовуватись для забезпечення безпеки даних, виявляючи несанкціоновані зміни або пошкодження даних.

5. Економія ресурсів. Замість того, щоб дублювати дані для забезпечення їх надійності, використання ККП дозволяє досягти тієї ж мети з меншою витратою ресурсів, що є важливим для систем з обмеженими ресурсами, наприклад, вбудованих систем, пристроїв IoT.

6. Адаптивність до різних умов. Різні ККП можуть бути адаптовані до специфічних умов та вимог конкретних систем, що робить їх універсальним інструментом для забезпечення надійності даних.

Здатність ККП виявляти та виправляти помилки базується на додаванні надлишкової інформації до вихідних даних. Ці додаткові дані зазвичай називають перевірочними або паритетними бітами. Кількість додаткових бітів або символів залежить від типу коду, а також від кількості помилок, які він дозволяє виправити чи виявити.

Вибір типу ККП зумовлений їх властивостями та ефективністю застосування [22, 23]. Серед найбільш поширених ККП є:

− Коди Хеммінга - використовуються для виявлення та виправлення одиничних помилок у даних в пам'яті комп'ютерів, телекомунікаційних системах та при передачі даних у надійних КЗ.

− Коди Гоппа - використовується у застосуваннях, де потрібна висока стійкість до помилок, зокрема в криптографії. Мають здатність виправлення кількох помилок, але можуть бути складними у реалізації та вимагати значних обчислювальних ресурсів.

− BCH коди - клас лінійних ККП, які можуть виявляти і виправляти декілька помилок, що широко використовуються у системах супутникового зв'язку та оптоволоконних мережах.

− Коди Ріда-Соломона – ефективно застосовуються для корекції групових помилок, зокрема в CD, DVD, QR-кодах та цифровому телебаченні.

− Коди з низькою щільністю перевірки парності (LDPC) – забезпечують оптимальну продуктивність при дуже низьких рівнях шуму, використовуються в сучасних системах зв'язку, наприклад Wi-Fi та 5G.

− Turbo-коди - застосовуються в стандартних протоколах зв'язку 3G та 4G для забезпечення високої продуктивності завдяки ітеративним алгоритмам декодування.

Для того, щоб вибрати код для конкретної задачі чи ситуації ККП можна порівнювати за такими характеристиками, як:

− швидкодія - час, необхідний для кодування та декодування повідомлень. Деякі коди можуть вимагати більше обчислювальних ресурсів, що збільшує часові витрати;

− споживання ресурсів - кількість пам'яті та обчислювальних потужностей, необхідних для кодування та декодування;

− ефективність передачі даних - співвідношення між корисною інформацією та загальним обсягом переданих даних, з урахуванням контрольних символів;

− складність реалізації - складність алгоритмів кодування та декодування, що включає як апаратні, так і програмні вимоги.

В таблиці 2.1 наведено порівняння ККП за даними характеристиками.

Таблиця 2.1 – Порівняльні характеристики коригуючих кодів

	Швидкодія	Споживання ресурсів	Ефективність передачі даних	Складність реалізації
Коди Хеммінга	Висока	Низьке	Середня	Низька
LDPC	Середня	Високе	Висока	Висока
BCH	Середня	Середнє	Середня	Середня
Коди Ріда-Соломона	Середня	Високе	Середня	Висока
Коди Гоппа	Середня	Високе	Середня	Висока
Turbo-коди	Середня	Дуже високе	Висока	Дуже висока

Дані таблиці 2.1 дозволяють зробити висновки щодо ефективності застосування. Коди Хеммінга демонструють високу швидкодію та низькі вимоги до ресурсів. Вони використовують алгоритми кодування та декодування, що легко реалізуються. Мають просту структуру і не потребують великої кількості додаткових символів, що дозволяє їх використання для реалізації в системах, де важливо зберегти баланс між ефективністю та складністю.

LDPC коди, хоча і забезпечують високу ефективність і корекцію помилок при невеликій кількості паритетних символів, потребують великої кількості обчислювальних ресурсів та об’ємів пам'яті. Висока складність алгоритмів декодування обмежує їх застосування, особливо в системах де важлива швидкодія.

BCH коди мають середні показники швидкодії та ефективності. Вони можуть бути оптимальними у випадку використання великих блоків даних. Однак складність їх реалізації вища, ніж у кодів Хеммінга, що робить їх менш привабливими для простих і швидких систем.

Застосування кодів Ріда-Соломона є доцільним у системах, де необхідна висока надійність. Незважаючи на значну обчислювальну складність алгоритмів кодування та декодування, ці коди забезпечують ефективну корекцію помилок, що робить їх оптимальними для використання в спеціалізованих застосуваннях.

Коди Гоппа потребують значних обчислювальних ресурсів і використовують складніші алгоритми порівняно з кодами Хеммінга, що обмежує їх застосування у загальних задачах. Вони забезпечують виправлення великої кількості помилок, проте можуть вимагати значної кількості контрольних бітів.

Turbo-коди забезпечують високу ефективність передачі даних і корекцію помилок, однак мають дуже високе споживання ресурсів та складність реалізації. Їх зазвичай використовують у високопродуктивних системах, де є можливість обробляти великі обсяги даних.

Для реалізації програмного засобу безпечного обміну даними найбільш доцільним є використання коду Хеммінга. Його висока швидкодія, низькі вимоги до ресурсів та проста реалізація дозволяють ефективно забезпечити корекцію помилок у умовах обмежених ресурсів, що є важливим для реальних застосувань, таких як передачі даних через мережі з низькими вимогами до складності алгоритмів.

Крім того, реалізація алгоритмів кодування та декодування дозволяє їх реалізацію у системах числення з основою відмінною від двійкової [24] сприяє зручній інтеграції коду Хеммінга в систему безпечного обміну даними з помірними вимогами до ефективності передачі даних.

2.2 Дослідження механізмів виявлення та корекції помилок коду Хеммінга в двійковій системі числення

Коди Хеммінга, розроблені Річардом Хеммінгом у 1950-х роках, є одним з найбільш відомих і найпростіших типом ККП. Вони широко використовуються у сучасних електронних системах кодування,

інформаційних системах та системах зв‘язку, а на їх основі розроблено багато варіантів та розширень.

Однією з основних характеристик коду є мінімальна відстань d_{min}, яка визначає завадостійкість коду. Її називають Хеммінгова відстань - найменша кількість бітів, які потрібно змінити, щоб перетворити одне кодове слово в інше. Існують певні співвідношення між d_{min} та кількістю помилок, які може виявити чи виправити код [22]:

1. Для коду з виявленням помилок повинна виконуватись умова:

$$d_{min} \geq t + 1, \tag{2.1}$$

де t – кількість помилок, які можуть бути виявлені.

2. Для коду з виправленням помилок повинна виконуватись умова:

$$d_{min} \geq 2 \cdot \sigma + 1, \tag{2.2}$$

де σ – кількість помилок, які можуть бути виправлені.

3. Для коду, що забезпечує і виявлення, і виправлення помилок, повинна виконуватись умова:

$$d_{min} \geq t + \sigma + 1. \tag{2.3}$$

Чим більшою є мінімальна відстань, тим більше помилок може бути виявлено та виправлено, оскільки з більшим значенням d_{min} код здатний коректувати більшу кількість помилок без втрати точності.

Залежно від $d_{min} \geq$, код має певні властивості:

- якщо $d_{min} = 3$, то код може виявляти до 2 помилок і виправляти 1 помилку;
- якщо $d_{min} = 4$, то код може виявляти до 3 помилок і виправляти 2 помилки.

Відповідно до наведених співвідношень (2.1)-(2.3) для кодів Хеммінга $d_{min} = 3$. Це означає, що між будь-якими двома кодовими словами в коді Хеммінга є щонайменше 3 відмінні біти. Тому він здатний виправити одну і виявити дві помилки в переданих даних.

Розрядність коду Хеммінга n визначається з умови:

$$2^k \leq \frac{2^n}{1+n}, \quad (2.4)$$

що у логарифмічній формі має вигляд:

$$k \leq n - log_2(1+n), \quad (2.5)$$

де k - кількість інформаційних бітів;

r – кількість контрольних бітів;

n – кількість бітів кодового слова.

В практичному застосуванні довжина коду n зазвичай виражається як:

$$n = 2^r - 1. \quad (2.6)$$

Коди Хеммінга, у порівняні з іншими лінійними ЕСС, характеризуються наступними перевагами.

1. Простота та ефективність алгоритму формування.

2. Баланс між кількістю контрольних та інформаційних бітів дозволяє знизити надлишковість при збереженні необхідного рівня завадостійкості.

Хоча алгоритм створення коду Хеммінга (кодування) досить простий, він включає наступні кроки:

1. Визначення кількості контрольних символів.

2. Створення шаблону для коду Хеммінга.

3. Обчислення контрольних сум (біти парності).

4. Формування кодового повідомлення.

Для зручності розглянемо приклад створення коду Хеммінга для вихідного повідомлення $m = 1001\ 0110$.

Кількість контрольних символів коду Хеммінга r визначається таким чином, щоб виконувалася умова:

$$2^r \geq k + r + 1, \quad (2.7)$$

Їх кількість повинна бути достатньою для того, щоб можна було записати в двійковій формі номер будь-якої з позицій в отриманому коді. При цьому їх не повинно бути більше, ніж це необхідно.

Наприклад, для $k = 8$ буде недостатньо 3 контрольних символів. Для того, щоб визначити тільки позиції інформаційних символів k, потрібно 4

розряди, оскільки двійкове представлення числа 8 це 1000. Але потрібно ще врахувати додаткові контрольні символи. Якщо кількість контрольних символів r=4, то довжина коду складе $n = k + r = 8 + 4 = 12$, що дозволяє представити номер будь-якої із позицій коду в двійковій формі.

Перевіримо умову (2.7) для $r = 4$. В результаті отримаємо:

$$2^4 \geq 8 + 4 + 1,$$

$$16 \geq 13.$$

Отже, 4 контрольні символи є достатніми.

Контрольні символи r у коді Хеммінга розташовуються на позиціях, які відповідають степеням числа 2, тобто 1, 2, 4, 8 і т.д. Інформаційні символи k розміщуються на всіх інших позиціях, а їх порядок у коді залишається таким самим, як у натуральному двійковому представленні вихідного повідомлення, тобто вони не змінюються між собою.

Для 8-бітного повідомлення шаблон коду матиме вигляд, що наведений на рисунку 2.2.

№ позиції	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}	n_{11}	n_{12}
Значення	r_1	r_2	k_1	r_3	k_2	k_3	k_4	r_4	k_5	k_6	k_7	k_8

Рисунок 2.2 - Ілюстрація принципу формування коду Хеммінга

де $r_1 - r_4$ - контрольні символи,

$k_1 - k_8$ - інформаційні символи.

Підставивши бітові значення вихідного повідомлення $m = 1001\ 0110$ у шаблон, наведений на рисунку 2.2 отримаємо наступний вигляд створеного коду (рисунок 2.3).

№ позиції	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}	n_{11}	n_{12}
Значення	r_1	r_2	1	r_3	0	0	1	r_4	0	1	1	0

Рисунок 2.3 - Ілюстрація принципу формування коду Хеммінга

Спосіб формування контрольних сум S_i коду Хеммінга ілюструє таблиця 2.2, де наведене двійкове представлення чисел від 1 до 12, що

відповідають позиціям в коді, який створюється. Для кожної контрольної суми S_i, необхідно проаналізувати відповідний рядок і включити до розрахунку ті номери позицій, двійкове представлення яких містить значення «1». Ці позиції відповідатимуть позиціям, які використовуються для формування конкретної контрольної суми.

Таблиця 2.2 – Принцип обчислення контрольних сум

№ позиції	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}	n_{11}	n_{12}
S_4	0	0	0	0	0	0	0	1	1	1	1	1
S_3	0	0	0	1	1	1	1	0	0	0	0	1
S_2	0	1	1	0	0	1	1	0	0	1	1	0
S_1	1	0	1	0	1	0	1	0	1	0	1	0

На основі даних таблиці 2.2 отримаємо вирази для обчислення контрольних сум:

$$\begin{gathered} S_1 = n_1 \oplus n_3 \oplus n_5 \oplus n_7 \oplus n_9 \oplus n_{11}; \\ S_2 = n_2 \oplus n_3 \oplus n_6 \oplus n_7 \oplus n_{10} \oplus n_{11}; \\ S_3 = n_4 \oplus n_5 \oplus n_6 \oplus n_7 \oplus n_{12}; \\ S_4 = n_8 \oplus n_9 \oplus n_{10} \oplus n_{11} \oplus n_{12}. \end{gathered} \quad (2.8)$$

Обчислення контрольних сум відбувається за допомогою операції XOR логічне додавання по модулю 2. Підставивши відомі значення інформаційних символів та позначення контрольних символів в (2.8) отримаємо:

$$\begin{gathered} S_1 = r_1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1; \\ S_2 = r_2 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1; \\ S_3 = r_3 \oplus 0 \oplus 0 \oplus 1 \oplus 0; \\ S_4 = r_4 \oplus 0 \oplus 1 \oplus 1 \oplus 0. \end{gathered}$$

Контрольні суми обчислюються таким чином, щоб загальна кількість одиниць у всіх позиціях, що беруть участь в розрахунках, була парною. Оскільки контрольні суми мають бути рівні нулю, щоб кількість одиниць у кожній S_i була парною, потрібно прирівняти кожну з них до нуля. Розв’язавши

рівняння отримаємо значення контрольних символів:

$$r_1 = 1;$$

$$r_2 = 0;$$

$$r_3 = 1;$$

$$r_4 = 0.$$

Після обчислення контрольних символів отримаємо кодову послідовність, що наведена на рисунку 2.4.

№ позиції	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}	n_{11}	n_{12}
Значення	1	0	1	1	0	0	1	0	0	1	1	0

Рисунок 2.4 - Ілюстрація принципу формування коду Хеммінга

В результаті для вихідного повідомлення

$$m = 1001\ 0110$$

створений код Хеммінга

$$x = 1011\ 0010\ 0110.$$

Алгоритм корекції помилок включає наступні кроки:

1. Приймання кодового слова.

2. Визначення позиції помилки.

3. Виправлення помилки.

Припустимо, що під час передачі через КЗ кодованого повідомлення

$$x = 1011\ 0010\ 0110$$

відбувалася помилка і один з символів змінив своє значення.

В результаті на приймаючій стороні отримане спотворене повідомлення:

$$x' = 1011\ 0110\ 0110$$

Для виявлення помилки необхідно виконати перерахунок контрольних сум S_i. На рисунку 2.5 наведено отримане повідомлення з помилкою.

№ позиції	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}	n_{11}	n_{12}
Значення	1	0	1	1	0	1	1	0	0	1	1	0

Рисунок 2.5 - Ілюстрація принципу виявлення помилки

Підставивши у вирази (2.8) відповідні значення x' отримаємо:

$$S_1 = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1;$$
$$S_2 = 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1;$$
$$S_3 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0;$$
$$S_4 = 0 \oplus 0 \oplus 1 \oplus 1 \oplus 0.$$

В результаті отримаємо, що

$$S_1 = 0;$$
$$S_2 = 1;$$
$$S_3 = 1;$$
$$S_4 = 0.$$

Якщо записати отримані результати обчислення S_i в двійковому форматі, починаючи з першої контрольної суми S_1 отримаємо:

$$1001_2 = 6_{10}.$$

В результаті аналізу контрольних сум визначено позицію помилки в коді. Виправляємо помилку, змінюючи символ в позиції n_6 на протилежний, і отримуємо коректне кодове слово

$$x = 1011\ 0010\ 0110.$$

Алгоритм декодування включає наступні кроки:

1. Видалення контрольних символів з виправленого коду.

2. Отримання оригінального повідомлення.

Оскільки відомо на яких позиціях розташовані контрольні символи, зокрема це позиції n_1, n_2, n_4, та n_8, видаляємо їх, щоб отримати тільки інформаційні символи. Після видалення контрольних символів, залишаються лише інформаційні символи (рисунок 2.6).

№позиції	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}	n_{11}	n_{12}
значення			1		0	1	1		0	1	1	0

Рисунок 2.6 - Ілюстрація принципу декодування коду Хеммінга

В даному прикладі після видалення контрольних символів залишаються символи на позиціях n_3, n_5, n_6, n_7, n_9, n_{10}, n_{11}, n_{12}, що відповідають вихідному повідомленню:

$$m = 1001\ 0110.$$

Схема алгоритму формування коду Хеммінга при класичній реалізації в двійковій системі числення наведена на рисунку 2.7 [22].

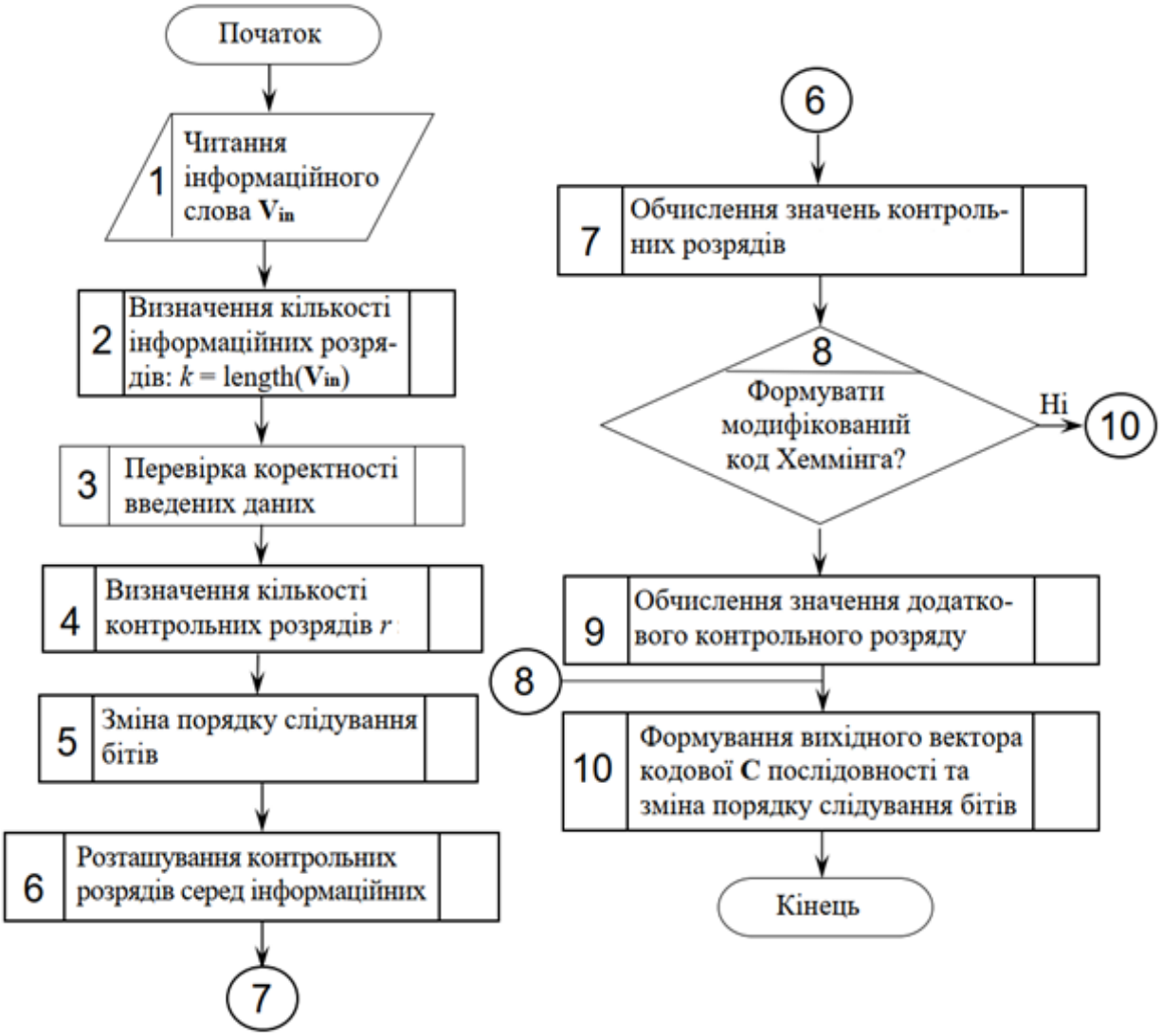

Рисунок 2.7 – Алгоритм формування коду Хеммінга

На рисунку 2.8 наведена схема алгоритму декодування послідовностей коду Хеммінга з використанням традиційного підходу до формування коду, де застосовуються двійкові операції для розрахунку контрольних символів і формування коду в бінарному вигляді [22].

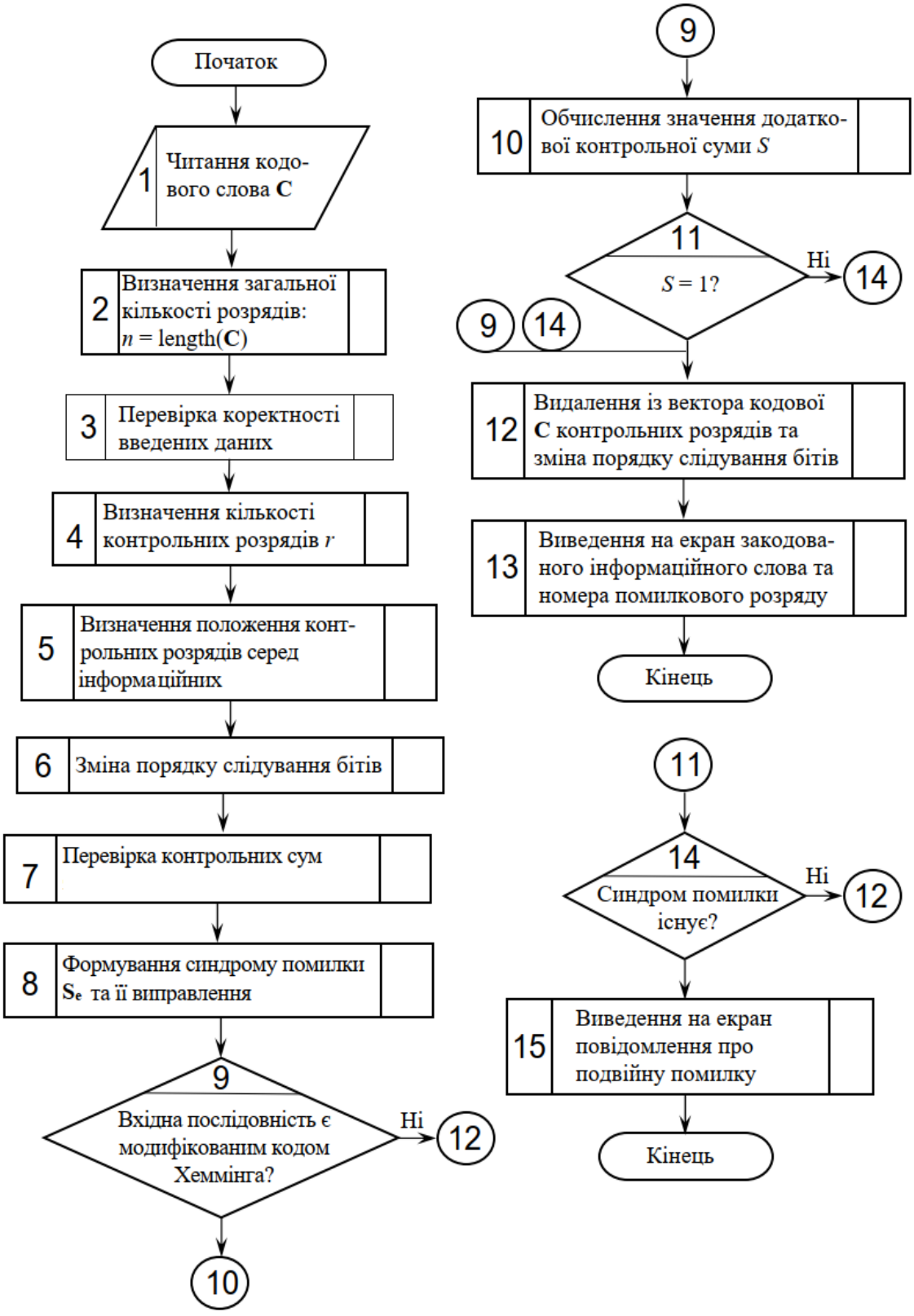

Рисунок 2.8 - Схема алгоритму декодування послідовностей коду Хеммінга

Код Хеммінга в двійковій системі числення дозволяє ефективно виявляти та виправляти помилки в переданих даних.

2.3 Дослідження створення коду Хеммінга для систем числення з довільною основою

При класичному створенні коду Хеммінга використовується бінарна системи числення де дані представлені бітами (0 та 1), а контрольні символи розраховуються за допомогою операцій додавання за модулем 2. В такому випадку словник складається лише з двох елементів $\{0,1\}$ і всі операції виконуються в межах цієї множини. Основною перевагою такого підходу є простота реалізації та корекції помилок, оскільки кожен символ має лише два стани, а помилка може полягати в зміні 0 на 1 або навпаки.

Однак використання системи числення з іншою основою, наприклад, простих чисел, дозволяє працювати з ширшим діапазоном значень. При переході до систем числення з основою p, де $\mathrm{p} > 2$, використовується арифметика за модулем p [24]. Це дозволяє розширити словник символів значення якого можуть бути в межах від 0 до $p - 1$. Наприклад, у трійковій системі $p = 3$ використовуються символи $\{0, 1, 2\}$, при $p = 5$ словник містить елементи $\{0, 1, 2, 3, 4\}$ і так далі.

Основні етапи алгоритму кодування та декодування залишаються незмінними, незалежно від основи системи числення. Для ілюстрації створення коду Хеммінга в системі числення з основою $\mathrm{p} = 7$, де словник складатиметься з елементів від 0 до 6, вихідне повідомлення матиме вигляд

$$\mathrm{m} = 1\ 3\ 5\ 0\ 6\ 2\ 4\ 1\ .$$

Як і при класичній реалізації коду, відбувається додавання контрольних символів до інформаційних таким чином, щоб можна було виявити та виправити помилки, тобто у позиці, що відповідають степеням 2.

Підставивши значення вихідного повідомлення у шаблон, отримаємо вигляд коду, що наведений на рисунку 2.9.

№ позиції	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}	n_{11}	n_{12}
значення	r_1	r_2	1	r_3	3	5	0	r_4	6	2	4	1

Рисунок 2.9 - Ілюстрація принципу формування коду

Для обчислення контрольних символів використовуються значення інформаційних символів, відповідно та таблиці 2.2, з використанням операції додавання за модулем p.

$$r_1 = (n_3 + n_5 + n_7 + n_9 + n_{11})\ mod(p),$$
$$r_2 = (n_3 + n_6 + n_7 + n_{10} + n_{11})\ mod(p), \quad (2.9)$$
$$r_3 = (n_5 + n_6 + n_7 + n_{12})\ mod(p),$$
$$r_4 = (n_9 + n_{10} + n_{11} + n_{12})\ mod(p).$$

Підставивши відповідні значення (рисунок 2.9) у вирази 2.9 отримаємо:

$$r_1 = (1 + 3 + 0 + 6 + 4)\ mod(7),$$
$$r_2 = (1 + 5 + 0 + 2 + 4)\ mod(7),$$
$$r_3 = (3 + 5 + 0 + 1)\ mod(7),$$
$$r_4 = (6 + 2 + 4 + 1) mod(7).$$

Виконавши обчислення отримаємо значення контрольних символів:

$$r_1 = 0, r_2 = 5, r_3 = 2, r_4 = 6.$$

Після внесення отриманих даних в шаблон коду отримаємо кодоване повідомлення $x = 0\ 5\ 1\ 2\ 3\ 5\ 0\ 6\ 6\ 2\ 4\ 1$.

Для визначення позиції помилки необхідно виконати повторне обчислення контрольних символів r'_nвідповідно до 2.9 використовуючи отримані значення інформаційних символів. Наприклад під час передачі повідомлення відбулася помилка і отримано повідомлення

$$x' = 0\ 5\ 1\ 2\ 3\ 5\ 0\ 6\ 6\ 4\ 4\ 1.$$

В результаті обчислень отримуємо наступні значення:

$$r'_1 = 0,$$
$$r'_2 = 0,$$
$$r'_3 = 2,$$
$$r'_4 = 1.$$

Обчислені значення відрізняються від отриманих значень контрольних символів, що свідчить про наявність помилки. Для визначення позиції помилки у кодованому повідомленні необхідно порівняти отримані значення r_n та обчислені значення r'_nконтрольних символів (таблиця 2.3).

Таблиця 2.3 – Порівняння значень контрольних символів

r_n	0	5	2	6
r'_n	0	0	2	1
	0	1	0	1

В результаті порівняння отримано значення 1010_2, що відповідає 10_{10}. Отже помилка виникла у позиції n_{10}.

На відміну від бінарної, у системі числення з основою p, помилка в може мати $p-1$ можливих значень. Наприклад, якщо модуль $p=3$, символ, що первісно мав значення 1, може бути змінений на 0 або 2 через помилку. Таким чином, виникає необхідність не лише виявити, що помилка сталася, але й визначити її розмір q, тобто на скільки символ відрізняється від свого початкового значення.

Для обчислення розміру помилки використовується обчислення контрольних сум S_i (2.8) з використанням операції додавання за модулем p.

$$\begin{gathered} S_1 = (n_1 + n_3 + n_5 + n_7 + n_9 + n_{11})\ mod\ (p); \\ S_2 = (n_2 + n_3 + n_6 + n_7 + n_{10} + n_{11})\ mod\ (p); \\ S_3 = (n_4 + n_5 + n_6 + n_7 + n_{12})\ mod\ (p); \\ S_4 = (n_8 + n_9 + n_{10} + n_{11} + n_{12}) mod\ (p). \end{gathered} \tag{2.10}$$

За допомогою (2.10) можна виразити q_n, наприклад при помилці, що знаходиться в 10 позиції отримаємо:

$$\begin{gathered} q_{10} = (r_2 - n_3 + n_6 + n_7 + n_{11})\ mod(p), \\ q_{10} = (r_4 - n_9 + n_{11} + n_{12})\ mod(p). \end{gathered} \tag{2.11}$$

Виконавши обчислення отримаємо правильне значення спотвореного символу $q_{10}=2$. Після виправлення помилки отримуємо кодоване повідомлення

$$x = 0\ 5\ 1\ 2\ 3\ 5\ 0\ 6\ 6\ 2\ 4\ 1.$$

Декодування передбачає видалення контрольних символів з кодованого повідомлення та отримання вихідного значення

$$\mathrm{m} = 1\ 3\ 5\ 0\ 6\ 2\ 4\ 1.$$

Схема алгоритму кодування та декодування коду Хеммінга з використанням системи числення з основою p наведений на рисунку 2.10 [24].

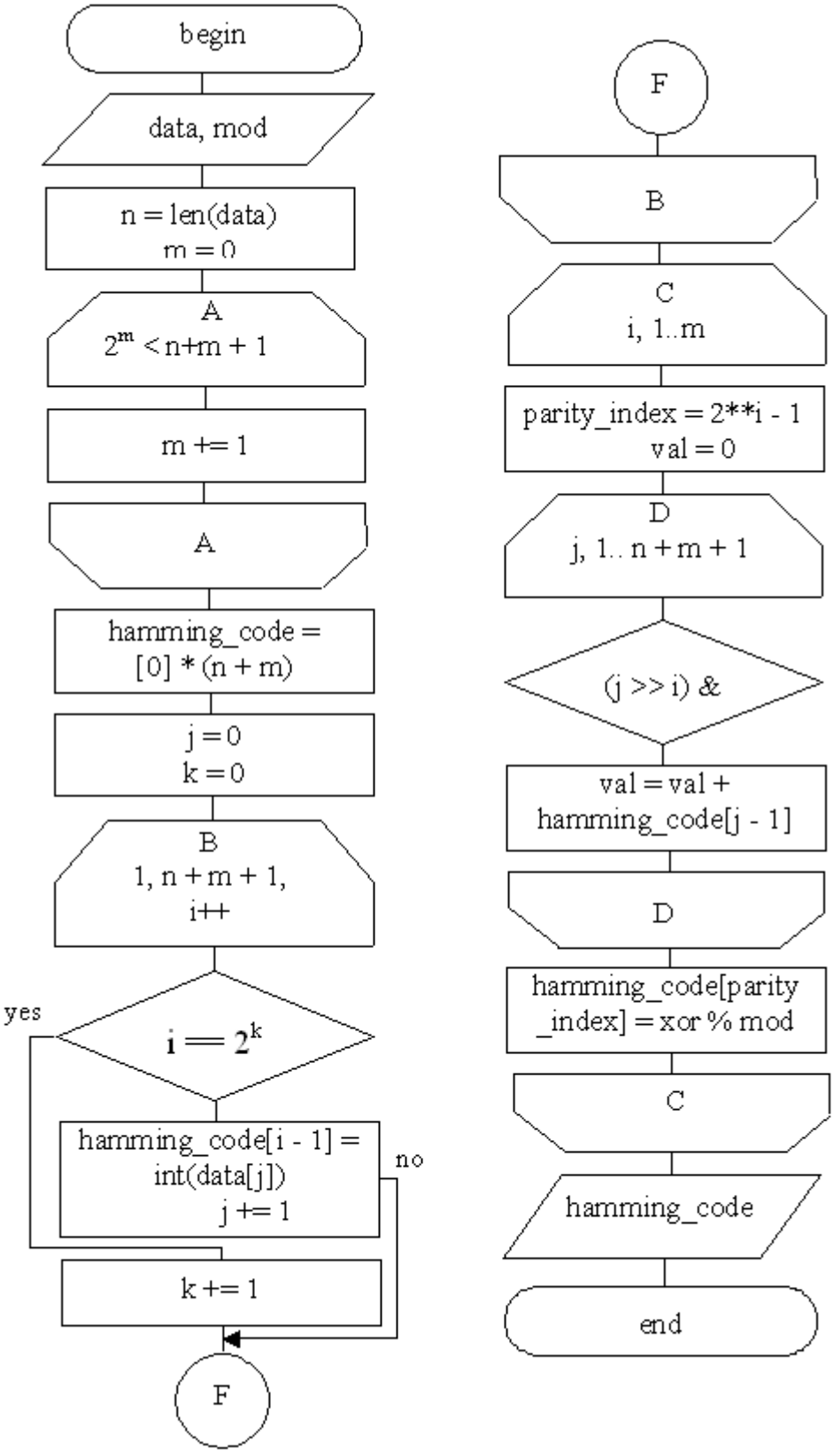

Рисунок 2.10 – Схема алгоритму створення розширеного коду Хеммінга

Створення коду Хеммінга з використанням модульної арифметики, розширює можливості традиційного бінарного коду Хеммінга, що дозволяє використовувати більш широке коло систем числення. У цьому випадку, послідовність даних $m_1, m_2, m_3, \dots, m_n$ де кожен елемент m_i належить множині $\{0, 1, 2, \dots, p-1\}$, використовує основу системи числення p як модуль для виконання операцій кодування інформаційних та контрольних символів повідомлення. Такий підхід дозволяє адаптувати код Хеммінга до систем числення з будь-якою основою, не обмежуючись лише двійковою системою числення.

Використання системи числення з основою $\mathrm{p} > 2$ має наступні переваги:

− гнучкість у виборі параметрів - можна адаптувати створення коду до різних систем числення;

− підвищена стійкість до зламу - розширення діапазону словника ускладнює аналіз і відновлення вихідних даних для потенційного зловмисника;

− ефективне використання ресурсів - оскільки інформаційні та контрольні символи можуть приймати більше значень, при фіксованій довжині кодованого повідомлення можна передати більше інформації, зберігаючи здатність виявлення та виправлення помилок.

Таким чином, використання модульної арифметики де $\mathrm{p} > 2$ дозволяє розширити можливості класичного коду Хеммінга без зміни основних принципів його створення, а також підвищує криптографічну стійкість.

2.4 Розробка алгоритму роботи програмного засобу

При розробці програмного засобу було визначено необхідність забезпечення безпеки та надійності передачі даних. Для забезпечення цілісності даних запропоновано використання коригуючих властивостей коду Хеммінга, який дозволяє виявляти та виправляти помилки, що можуть виникнути під час передачі даних, забезпечуючи їх точність і правильність. Для

забезпечення конфіденційності даних запропоновано застосування перетворення в ASCII-код, яке дозволяє представити текстові дані у числовій формі, що є зручним для подальшої математичної обробки [25].

Перетворення в ASCII також забезпечує універсальність проєктованого програмного засобу, адже дозволяє підтримувати передачу як числових, так і текстових даних. Оскільки код Хеммінга працює з числовими значеннями, перед кодуванням текстові дані потрібно перевести в числовий формат. За допомогою ASCII-перетворення текстове повідомлення представляється у вигляді послідовності чисел, що дає змогу обробляти їх відповідно до алгоритму.

Також ASCII є загальноприйнятим стандартом, що дозволяє без додаткових перетворень працювати з числовими значеннями в програмному середовищі. Завдяки цьому після декодування та корекції помилок вихідний текст можна безпомилково відновити.

Проєктований програмний засіб призначений для безпечного обміну даними, використовуючи алгоритм шифрування на основі ASCII-перетворення та коду Хеммінга в системі числення з довільною основою p.

Алгоритм роботи програмного засобу включає етапи шифрування та дещифрування даних.

Основні кроки роботи алгоритму шифрування даних включають:

1. Отримання вихідних даних.

2. Перетворення вхідних даних у числовий формат.

3. Створення коду Хеммінга.

4. Модифікація кодованого повідомлення.

5. Виведення кодованого повідомлення.

Наприклад вихідними даними є текстове повідомлення

$$m = Hello.$$

Для виконання операції шифрування використовуємо систему числення з основою $p = 131$.

В результаті перетворення вихідних даних у числовий

формат отримаємо

$$m' = 72,\ 101,\ 108,\ 108,\ 111.$$

Далі на основі отриманих чисел будується код Хеммінга з використанням арифметики за модулем p. Зокрема відбувається формування інформаційних бітів, додавання контрольних бітів, обчислених за правилами коду Хеммінга у системі числення з основою p.

В результаті формування остаточного кодованого повідомлення отримаємо

$$\mathrm{x} = 130, 26, 72, 55, 101, 108, 108, 111, 111.$$

Для підвищення захисту даних, що передаються до кожен символ отриманого кодованого повідомлення додатково модифікується шляхом додавання модуля p.

Отримане повідомлення має вигляд:

$$h = 261, 157, 203, 186, 232, 239, 239, 242, 242.$$

Після того як код Хеммінга був створений та модифікований, отримані зашифровані дані виводяться на екран.

Зашифроване дані можуть бути передані через КЗ або зберігатися у файл.

На рисунку 2.11 наведено приклад виконання даного алгоритму, реалізований з використанням мови програмування Python.

```
Enter your choice: 1
Enter the message to encrypt: Hello
Stage 1 - Input message: Hello
Enter the modulus value: 131
Stage 2 - Data: [72, 101, 108, 108, 111]
Stage 3 - Hamming_code: [130, 26, 72, 55, 101, 108, 108, 111, 111]
Stage 4 - Protected_code: [261, 157, 203, 186, 232, 239, 239, 242, 242]
Stage 5 - Encrypted message: [261, 157, 203, 186, 232, 239, 239, 242, 242]
Do you want to save the encrypted message? (y/n): n
```

Рисунок 2.11 – Приклад реалізації шифрування даних

Алгоритм дешифрування включає наступні кроки:

1. Отримання зашифрованих даних.

2. Відновлення коду Хеммінга.

3. Виявлення та корекція помилки.

4. Відновлення вихідного повідомлення.

5. Перетворення даних у вихідний формат.

6. Виведення даних.

Наприклад, при передачі зашифрованих даних відбулася помилка і на приймаючій стороні отримано спотворене повідомлення

$$h' = 261, 157, 23, 186, 232, 239, 239, 242, 242.$$

Для відновлення початкової форми коду Хеммінга для подальшої обробки необхідно видалити із кожного елемента отриманих даних значення модуля p. В результаті отримуємо

$$x' = 130, 26, -108, 55, 101, 108, 108, 111, 111.$$

Використовуючи алгоритм корекції помилок для коду Хеммінга, відбувається перевірка наявності помилок у переданих даних. Якщо виявляються помилки, то вони виправляються відповідно до алгоритму коду Хеммінга, забезпечуючи точність даних (рисунок 2.12).

```
Enter your choice: 2
Enter the encrypted message: 261,157,23,186,232,239,239,242,242
Stage 1 - Send message: 261,157,23,186,232,239,239,242,242
Enter the modulus value: 131
Stage 2 - Hamming_code: [261, 157, 23, 186, 232, 239, 239, 242, 242]
Stage 3 - Ontected_code: [130, 26, -108, 55, 101, 108, 108, 111, 111]
original_parity_bit_value: 130
Error correction
False value: -108
Correct value: 72
Wrong bit at position: 3, is corrected.
Stage 4 - Corrected_code: [130, 26, 72, 55, 101, 108, 108, 111, 111]
Stage 5 - Data_bits: [72, 101, 108, 108, 111]
Stage 6 - Decrypted message: Hello
Do you want to save the decrypted message? (y/n):
```

Рисунок 2.12 – Ілюстрація виявлення та корекції помилок

Після корекції помилок із коду Хеммінга вибираються лише інформаційні символи, оскільки коригуючі необхідні лише для виявлення та виправлення можливої помилки

$$m' = 72, 101, 108, 108, 111.$$

Вони перетворюються знову в текстовий формат, що дає можливість відновити оригінальне повідомлення, яке було зашифроване.

На рисунку 2.13 наведено реалізовані алгоритми шифрування та дешифрування даних для програмного засобу безпечної передачі даних.

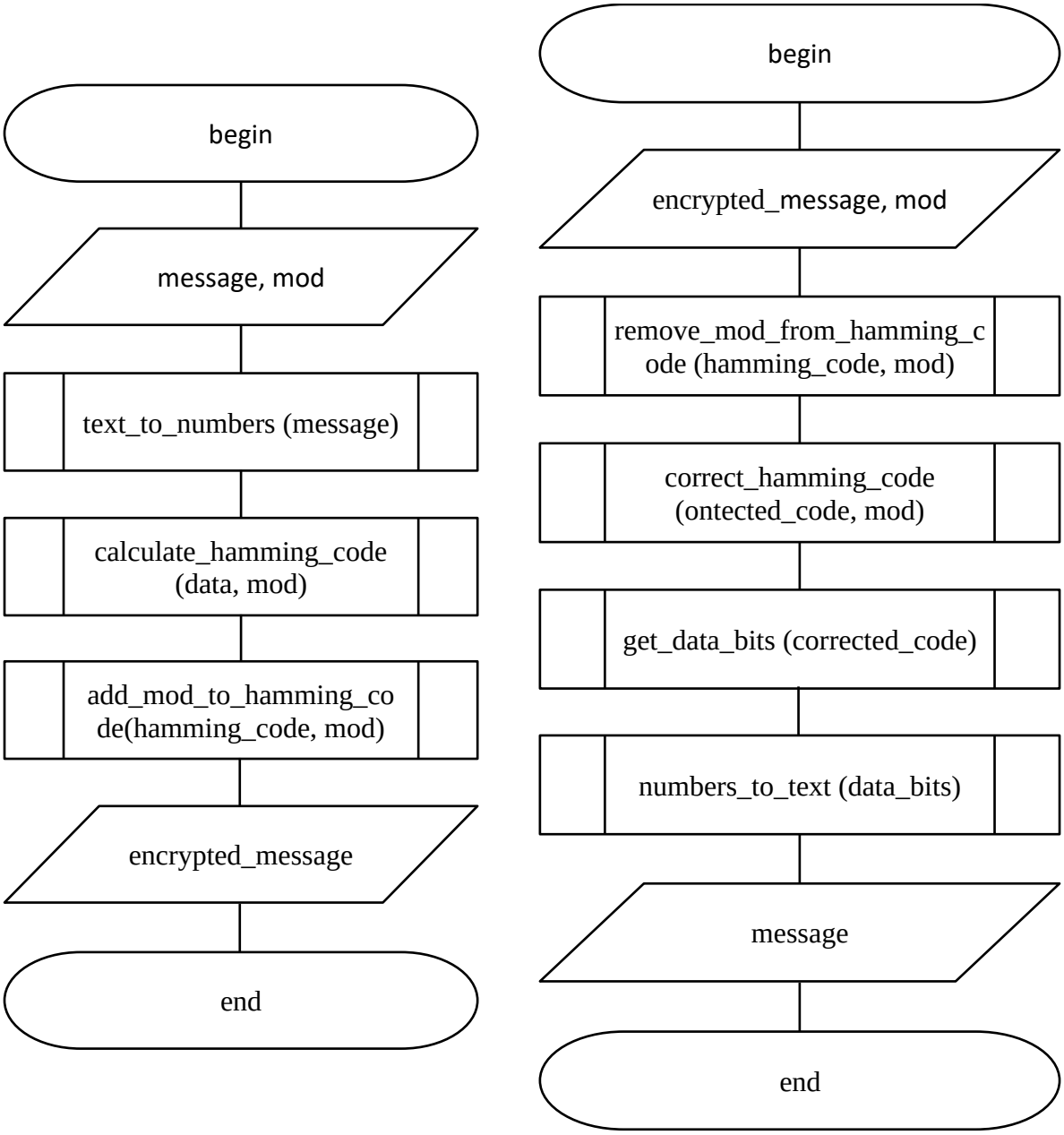

Рисунок 2.13 – Алгоритм роботи програмного засобу на етапах шифрування та дешифрування даних

Наведений алгоритм поєднує перетворення вхідних даних у числовий формат за допомогою ASCII-кодування, створення коду Хеммінга в системі числення з довільною основою та використання операцій додавання за

модулем для виявлення та виправлення помилок. Він дозволяє забезпечити необхідний рівень захисту для переданих даних, включаючи можливість виправлення помилок завдяки використанню коду Хеммінга та додаткового модулярного захисту.

Для коректного виправлення помилок важливо, щоб значення модуля p було не меншим ніж діапазон значень, які використовуються. Наприклад для ASCII-кодів, які знаходяться в діапазоні від 0 до 127, необхідною є виконання умови $p > 128$. Це гарантує, що всі значення ASCII будуть правильно зашифровані і відновлені без втрати інформації.

Програмний засіб дозволить вирішити проблему забезпечення безпеки та надійності передачі даних у сучасних інформаційних системах, забезпечуючи надійний і швидкий обмін інформацією, навіть при нестабільних та зашумлених каналах передачі даних.

3. РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСОБУ

3.1 Реалізація користувацького інтерфейсу

При проєктуванні інтерфейсу програмного засобу були визначені основні вимоги, зокрема зручність використання, простота і ефективність, що зробить застосунок придатним для широкого кола користувачів. Інтерфейс повинен бути інтуїтивно зрозумілий, забезпечувати швидкий доступ до основних функцій, підтримувати гнучкі налаштування та передбачити підтримку різних режимів роботи, які легко перемикати.

Головне вікно програмного засобу (рисунку 3.1) містить елементи для вибору параметрів, необхідних для коректної роботи. Інтерфейс дозволяє користувачу вводити текстові або файлові дані, шифрувати та дешифрувати їх, а також вибирати спосіб відображення результату.

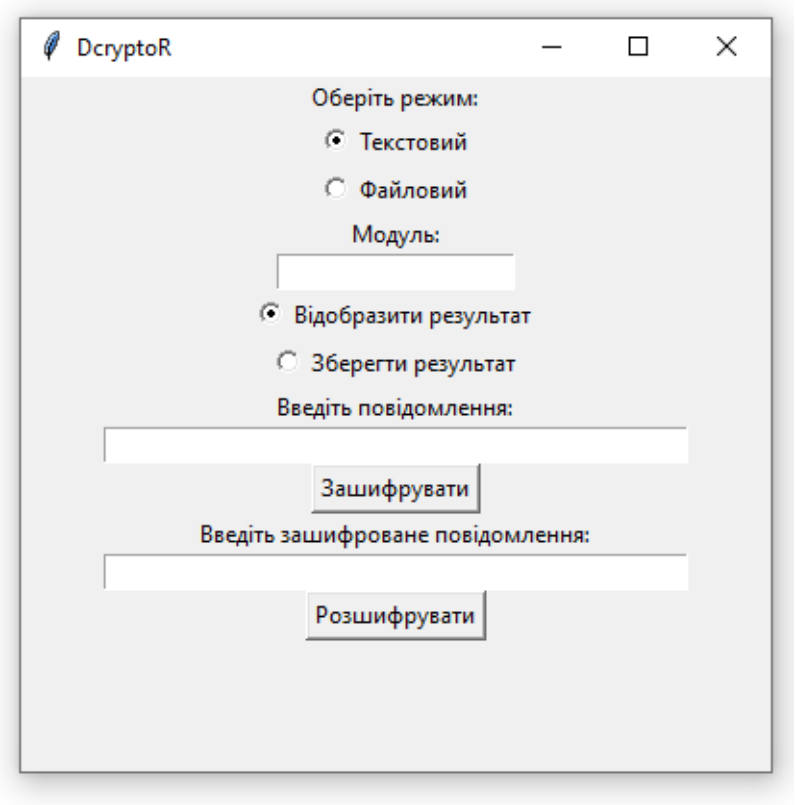

Рисунок 3.1 – Головне вікно програмного засобу на етапі розробки

Усі основні функції згруповані у логічно впорядковані блоки, що спрощує навігацію та використання програмного засобу. При зміні режиму

текстовий чи файловий автоматично приховуються або відображаються відповідні поля вводу, кнопки та текстові підказки. Це забезпечує зручність використання і мінімізує зайві елементи в інтерфейсі.

Графічна частина реалізована за допомогою мови програмування Python та бібліотеки Tkinter, що дозволяє створювати легкий та швидкий інтерфейс, який не перевантажує систему і не потребує складної конфігурації. Tkinter використано як стандартну бібліотеку для створення GUI у Python, оскільки вона забезпечує мінімальні залежності, простоту інтеграції та кросплатформенність, тобто роботу у Windows, Linux та macOS без додаткових налаштувань. Використання Python та Tkinter дозволить легко модифікувати та розширити ітресфейс за потреби.

Для зручності використання кінцевими користувачами програмний засіб було зібрано у виконуваний файл формату .exe (рисунок 3.2) за допомогою інструмента PyInstaller, що дозволяє запускати програму без необхідності встановлення середовища Python.

Рисунок 3.2 - Вигляд ярлика програмного засобу DcryptoR

Це забезпечує швидкий доступ до функціоналу програмного засобу та спрощує його поширення серед користувачів як готовий до використання застосунок, без необхідності встановлення додаткових компонентів.

3.2 Реалізація функцій

Проєктування основних функцій програмного засобу було спрямоване на забезпечення ефективного та надійного процесу шифрування та дешифрування даних з використанням коду Хеммінга для корекції помилок, що дозволяє користувачам зручно та безпечно обмінюватися інформацією.

Функція process_encryption () реалізує шифрування вихідних даних. Вона включає кілька етапів:

− отримання тексту повідомлення з введеного поля;

− перетворення отриманих даних у числовий формат за допомогою функції text_to_numbers ();

− обчислення коду Хеммінга для отриманих даних з урахуванням заданого модуля через функцію calculate_hamming_code ();

− додавання модульного значення до коду Хеммінга за допомогою add_mod_to_hamming_code ();

− виведення результату або збереження його у файл, залежно від вибору користувача. Якщо обрано "зберегти", результат записується в файл за допомогою save_file ().

Функція process_decryption () реалізує виявлення й виправлення спотворень та дешифрування зашифрованих даних. Процес включає такі етапи:

− отримання зашифрованого повідомлення (коду Хеммінга) з введеного поля;

− виділення числових значень із рядка і перетворення їх на список цілих чисел;

− видалення модульного значення з коду Хеммінга за допомогою функції remove_mod_from_hamming_code ();

− виправлення помилок у коді Хеммінга за допомогою функції correct_hamming_code ();

− вибір інформаційних бітів з виправленого коду за допомогою

get_information_symbols ();

– перетворення отриманих бітів назад у текстове повідомлення через функцію numbers_to_text ();

– виведення відновленого повідомлення або збереження його у файл, залежно від вибору користувача. Якщо обрано "зберегти", результат записується в файл за допомогою save_file ().

Ці функції містять обробку помилок, наприклад, у разі введення некоректних даних, через використання конструкції try-except для виведення відповідних повідомлень користувачу через messagebox.showerror.

Процес шифрування та дешифрування також адаптований для роботи з файлами. На рисунку 3.3 наведено фрагмент коду на етапі розробки, що описує функцію encrypt_file(). Вона має схожу структуру до process_encryption (), але працює з файлами замість текстових полів інтерфейсу.

Користувач вибирає файл, з якого буде зчитано текст. Далі текст з файлу конвертується у числовий формат. Відбувається створення коду Хеммінга для шифрування та його модифікація. В результаті користувач може зберегти зашифровані дані у файл або відобразити їх на екрані.

```
def encrypt_file():
    try:
        file_path = select_file()
        if not file_path:
            return
        with open(file_path, 'r', encoding='utf-8') as file:
            file_content = file.read()
        mod = int(entry_mod.get())
        data = text_to_numbers(file_content)
        hamming_code = calculate_hamming_code(data, mod)
        protected_code = add_mod_to_hamming_code(hamming_code, mod)
        if var_output_type.get() == 'save':
            save_file(','.join(map(str, protected_code)), "text")
        else:
            messagebox.showinfo("Результат", f"Зашифроване повідомлення: {','.join(map(str, protected_code))}")
    except ValueError:
        messagebox.showerror("Помилка", "Перевірте коректність введених даних")
```

Рисунок 3.3 – Функція шифрування даних з файлу

Функція decrypt_file () забезпечує користувачу можливість вибору файлу, який містить зашифровані дані. Далі відбувається зчитування вмісту файлу, з модифікованого коду видаляється значення модуля відбувається корекція помилок. Після цього дані відновлюються і перетворюються в текст. Користувач може вивести відновлене повідомлення на екран або зберегти його

у файл. На рисунку 3.4 наведено фрагмент коду на етапі розробки, що описує функцію decrypt_file ().

```
def decrypt_file():
    try:
        file_path = select_file()
        if not file_path:
            return
        with open(file_path, 'r', encoding='utf-8') as file:
            hamming_code_str = file.read()
        mod = int(entry_mod.get())
        hamming_code = [int(x) for x in hamming_code_str.split(',')]
        ontected_code = remove_mod_from_hamming_code(hamming_code, mod)
        corrected_code = correct_hamming_code(ontected_code, mod)
        data_symbols = get_information_symbols(corrected_code)
        decrypted_message = numbers_to_text(data_symbols)
        if var_output_type.get() == 'save':
            save_file(decrypted_message, "text")
        else:
            messagebox.showinfo("Результат", f"Відновлене повідомлення: {decrypted_message}")
        messagebox.showinfo("Корекція", f"Виправлене зашифроване повідомлення: {','.join(map(str, corrected_code))}")
    except ValueError:
        messagebox.showerror("Помилка", "Перевірте коректність введених даних")
```

Рисунок 3.4 – Функція дешифрування даних з файлу

Реалізовано функції які дозволяють отримувати дані з файлу та зберігати їх у файл. Функція select_file() використовує filedialog.askopenfilename() для вибору файлу через діалогове вікно. Вона обмежує вибір текстових файлів, формат *.txt та повертає шлях до обраного файлу.

На риунку 3.5 наведено фрагмент коду на етапі розробки, що реалізує вибір файлу.

```
def select_file():
    return filedialog.askopenfilename(filetypes=[("Text files", "*.txt")])
```

Рисунок 3.5 – Функція вибору файлу з даними

Функція save_file(content, file_type="text") забезпечує збереження вмісту в файл. Якщо тип файлу "text", діалогове вікно дозволяє зберегти текстовий файл *.txt. Якщо тип файлу "binary", зберігається бінарний файл формат *.bin. Після збереження виводиться повідомлення з підтвердженням успішного збереження файлу.

На рисунку 3.6 наведено фрагмент коду на етапі розробки, що реалізує збереження даних у файл.

```
def save_file(content, file_type="text"):
    if file_type == "text":
        file_path = filedialog.asksaveasfilename(defaultextension=".txt", filetypes=[("Text files", "*.txt")])
        if file_path:
            with open(file_path, 'w', encoding='utf-8') as file:
                file.write(content)
            messagebox.showinfo("Результат", f"Файл збережено як: {file_path}")
    elif file_type == "binary":
        file_path = filedialog.asksaveasfilename(defaultextension=".bin", filetypes=[("Binary files", "*.bin")])
        if file_path:
            with open(file_path, 'wb') as file:
                file.write(content)
            messagebox.showinfo("Результат", f"Файл збережено як: {file_path}")
```

Рисунок 3.6 – Функція збереження даних у файлу

Наведені функції використовують діалогові вікна для вибору та збереження файлів, що зручно для користувача.

Ключовими функціями для шифрування та дешифрування є наступні. Функція calculate_hamming_code (рисунок 3.7) створює код Хеммінга для переданих даних data та використовує заданий модуль mod для обчислення контрольних символів. Визначається кількість коригуючих символів r, необхідних для створення коду, залежно від довжини вхідних даних n. Створюється масив, який заповнюється значеннями даних, пропускаючи позиції, які будуть використовуватися для r. Далі обчислюються значення r для кожної відповідної позиції за допомогою операції XOR над відповідними символами. Результатом є код Хеммінга з інформаційними та контрольними символами.

```
def calculate_hamming_code(data, mod):
    n = len(data)
    r = 0
    while 2**r < n + r + 1:
        r += 1
    hamming_code = [0] * (n + r)
    j = 0
    k = 0
    for i in range(1, n + r + 1):
        if i == 2**k:
            k += 1
        else:
            hamming_code[i - 1] = int(data[j])
            j += 1
    for i in range(r):
        parity_index = 2**i - 1
        xor = 0
        for j in range(1, n + r + 1):
            if (j >> i) & 1:
                xor = xor + hamming_code[j - 1]
        hamming_code[parity_index] = xor % mod
    return hamming_code
```

Рисунок 3.7 - Функція створення коду Хеммінга

Функція correct_hamming_code (рисунок 3.8) виправляє помилки в зашифрованих наних encoded_data з використанням значення модуля mod. Спочатку вибираються інформаційні символи та обчислюються нові контрольні символи. Далі лтримані та обчислені значення порівнюються для виявлення помилки. Якщо помилка виявлена, символ коригується, відповідно до позиції помилки.

```
def correct_hamming_code(encoded_data, mod):
    information_symbols = get_information_symbols(encoded_data)
    new_control_symbols = calculate_control_symbol(information_symbols, mod)
    prior_symbols_from_data = get_control_symbols(encoded_data)
    compare_byte_arr = []
    for i in range(len(prior_symbols_from_data)):
        compare_byte_arr.append(prior_symbols_from_data[i] != new_control_symbols[i])
    index_e = bool_list_to_int(compare_byte_arr[::-1]) - 1
    if index_e >= 0 and index_e < len(encoded_data):
        control_symbol_position = 1
        for i in range(len(encoded_data)):
            if is_power_of_two(i + 1) and (i + 1) & (index_e + 1) != 0:
                control_symbol_position = i
                break
        original_control_symbol_value = encoded_data[control_symbol_position]
        data_sum = 0
        for i in range(len(encoded_data)):
            if (i + 1) & (1 << control_symbol_position) != 0 and not is_power_of_two(i + 1):
                data_sum += encoded_data[i]
        corrected_value = (original_control_symbol_value - (data_sum - encoded_data[index_e]) % mod)
        encoded_data[index_e] = corrected_value
        messagebox.showinfo("Корекція", f"Помилковий біт на позиції: {index_e + 1}, виправлено.")
    else:
        messagebox.showinfo("Корекція", "Помилок не виявлено.")
    return encoded_data
```

Рисунок 3.8 - Функція виявлення та корекції помилки

Функції control_symbols () та information_symbols () забезпечують отримання списків контрольних та інформаційних символів з кодованого повідомлення encoded_data, відповідно рисунок 3.9 та 3.10.

```
def get_control_symbols(encoded_data):
    control_symbols = []
    for i in range(len(encoded_data)):
        if is_power_of_two(i + 1):
            control_symbols.append(encoded_data[i])
    return control_symbols
```

Рисунок 3.9 – Функція для отримання списку контрольних символів

```
def get_information_symbols(encoded_data):
    information_symbols = []
    for i in range(1, len(encoded_data)):
        if not is_power_of_two(i + 1):
            information_symbols.append(encoded_data[i])
    return information_symbols
```

Рисунок 3.10 – Функція для отримання списку інформаційних символів

Проєктований програмний засіб включає ряд функціональних модулів:

− кодування - відповідає за шифрування даних за допомогою використання модульних перетворень;

− виявлення помилок - перевіряє дані на наявність помилок після декодування та застосовує методи корекції помилок;

− корекції помилок - виконує кодування та декодування даних з використанням властивостей коригуючих кодів Хеммінга;

− декодування - відповідає за відновлення вхідних даних після корекції помилок;

− введення/виведення даних - забезпечує інтерфейс для користувачів для завантаження файлів або введення текстових повідомлень для шифрування та збереження результатів;

− управління інтерфейсом - забезпечує користувачеві зручний графічний інтерфейс для роботи з програмним засобом.

Реалізація такого підходу дозволяє підвищити надійність передачі інформації, зменшити ймовірність викривлення даних та забезпечити стабільну роботу навіть у складних умовах, зокрема при нестабільних або зашумлених КЗ.

3.3 Тестування програмного засобу

Тестування програмного засобу є важливим етапом у процесі його розробки, оскільки дозволяє виявити помилки та недоліки, усунення яких забезпечить роботу застосунку відповідно до поставлених вимог. Це процес перевірки функціональності, стабільності та ефективності програмного засобу. Він дозволяє гарантувати її готовність для використання кінцевим користувачем.

Функціональне тестування програмного засобу для безпечної передачі даних підтвердило, що програма працює відповідно до вимог. Тестування охоплювало перевірку шифрування та дешифрування даних, сумісність з операційними системами Windows 10 і 11, а також ефективність використання

ресурсів. Результати підтверджують її сумісність з різними платформами.

Цільова аудиторія продукту включає широке коло користувачів, від приватних осіб до організацій, що працюють з конфіденційними даними. Програмний засіб спрямований на шифрування та дешифрування текстових файлів з використанням різних криптографічних методів, таких як алгоритми шифрування з виявленням та виправленням помилок.

Під час запуску відкривається головний інтерфейс, що відповідає вимогам та задачам користувача, забезпечуючи ефективну та безпечну передачу даних. Програма займає 9,83 Мб, що є оптимальним розміром і не впливає на ефективність роботи інших програм. Під час юзабіліті-тестування було виявлено, що інтерфейс програми зручний для користувачів, кнопки мають оптимальний розмір та розташовані в зручних місцях на екрані. Текст у програмі чіткий і легко сприймається, а підписи до кнопок прості й зрозумілі. Користувач може вибрати шифрування чи дешифрування тексту, імпортувати та експортувати текстові файли.

На рисунку 3.11 наведено головне вікно засобу в режимі роботи з файлами.

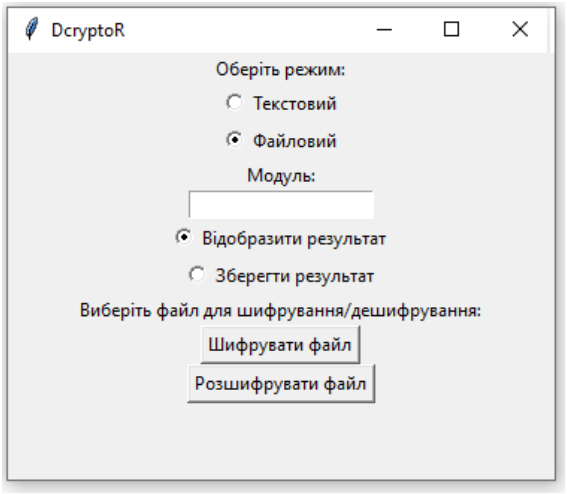

Рисунок 3.11 – Режим роботи з файлами

Для шифрування даних потрібно ввести значення модуля та обрати необхідний файл (рисунок 3.12).

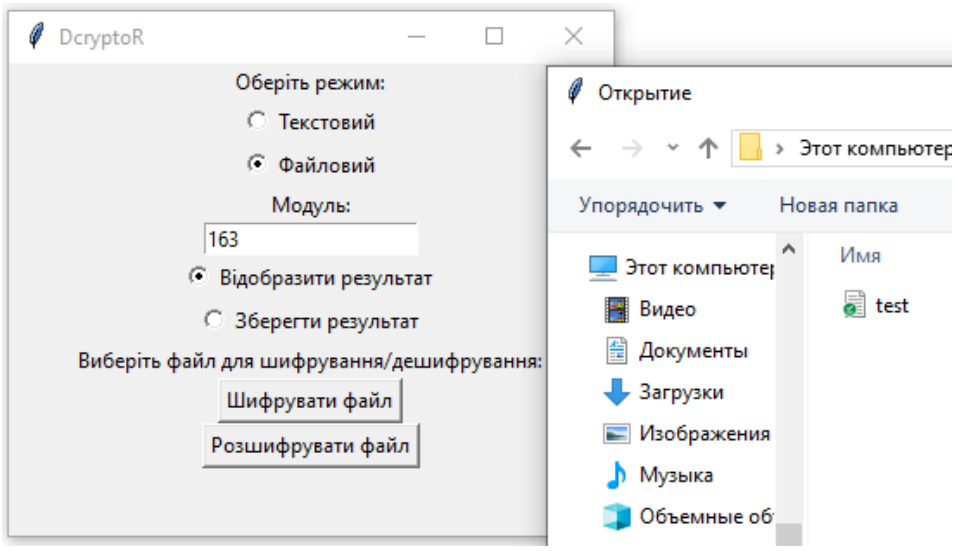

Рисунок 3.12 – Вибір файлу для шифрування

Після цього програма виконує необхідну операцію, яка залежить від вибраного модуля шифрування та обраного файлу. Оскільки було обрано режим відображення зашифрованих даних виводиться відповідне повідомлення (рисунок 3.13), що дає змогу користувачу побачити результат операції в реальному часі.

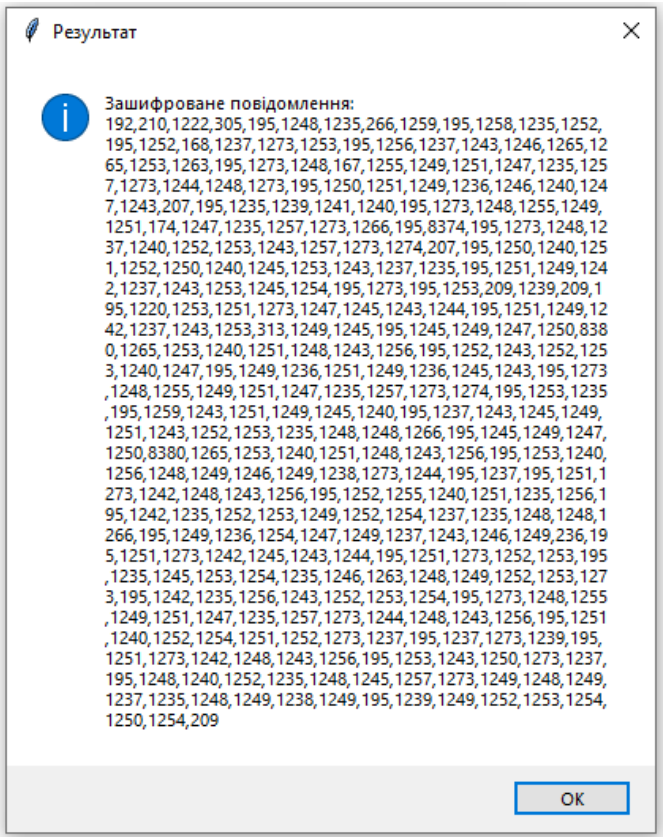

Рисунок 3.13 – Результат шифрування

Для забезпечення коректності функції зберігання зашифрованих даних, тестування передбачає перевірку можливості збереження результату в файл. Після виконання шифрування користувач може вказати шлях та ім'я файлу для збереження результату (рисунок 3.14).

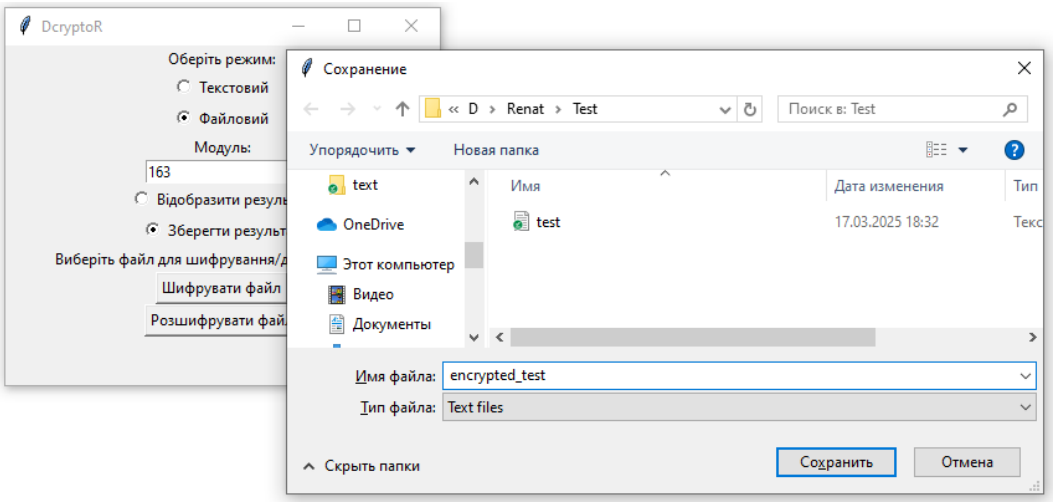

Рисунок 3.14 – Зберігання результату шифрування у файл

Якщо зберігання було виконано успішно, з'являється відповідне повідомлення з інформацією про шлях до файлу (рисунок 3.15).

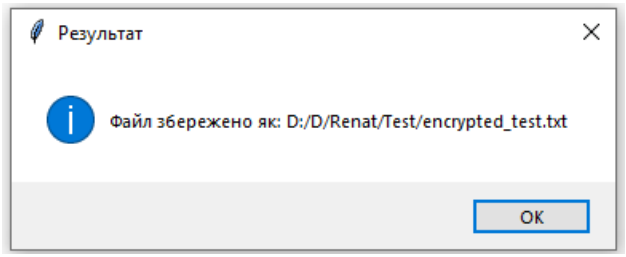

Рисунок 3.15 – Результат зберігання файлу

Переглянувши вміст збереженого файлу, можна підтвердити, що дані були записані правильно (рисунок 3.16).

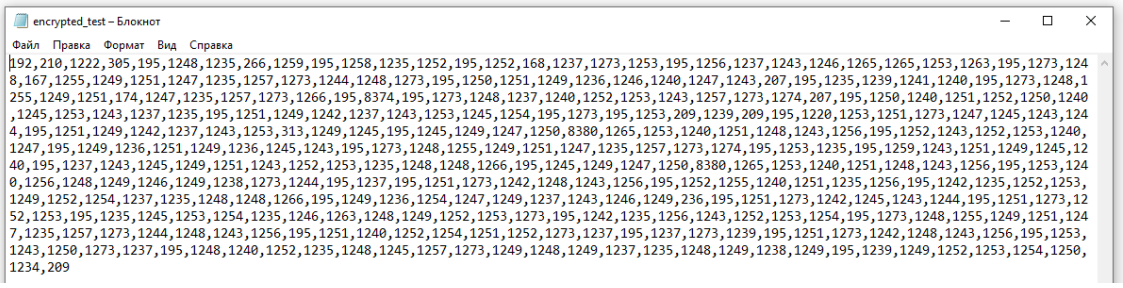

Рисунок 3.16 – Вміст збереженого файлу

Для тестування функціональності дешифрування необхідно вибрати відповідний режим роботи програмного засобу та файл з зашифрованими даними (рисунок 3.17).

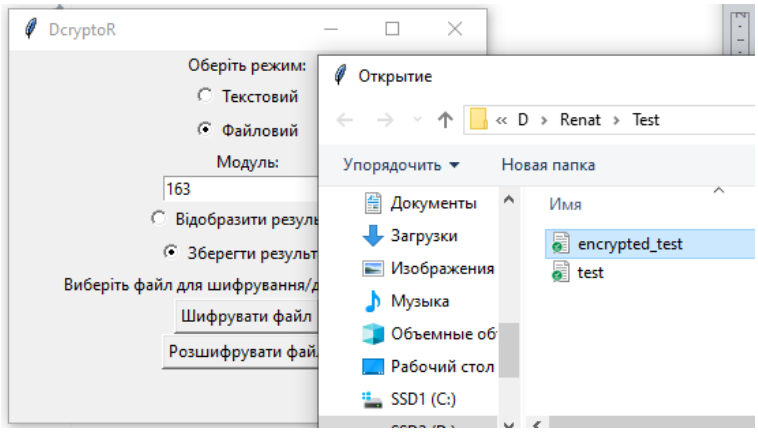

Рисунок 3.17 – Вибір файлу для дешифрування

У разі відсутності помилок у зашифрованих даних програма виводить повідомлення, що підтверджує коректність результату (рисунок 3.18).

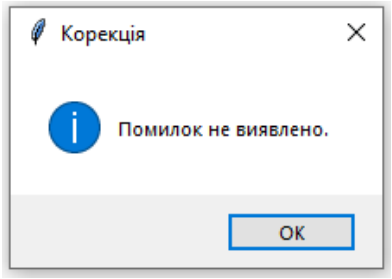

Рисунок 3.18 – Результат виявлення помилки

Результат дешифрування може бути збережений у файл (рисунок 3.19).

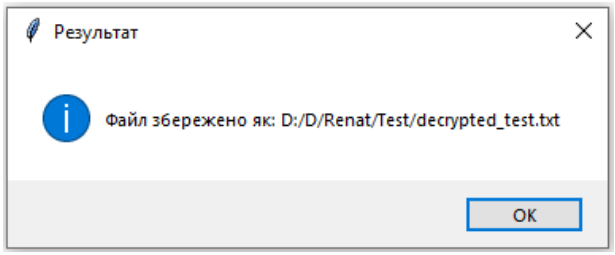

Рисунок 3.19 – Результат збереження дешифрованих даних

Після цього для підтвердження коректності дешифрування можна відобразити результат на екрані (рисунок 3.20).

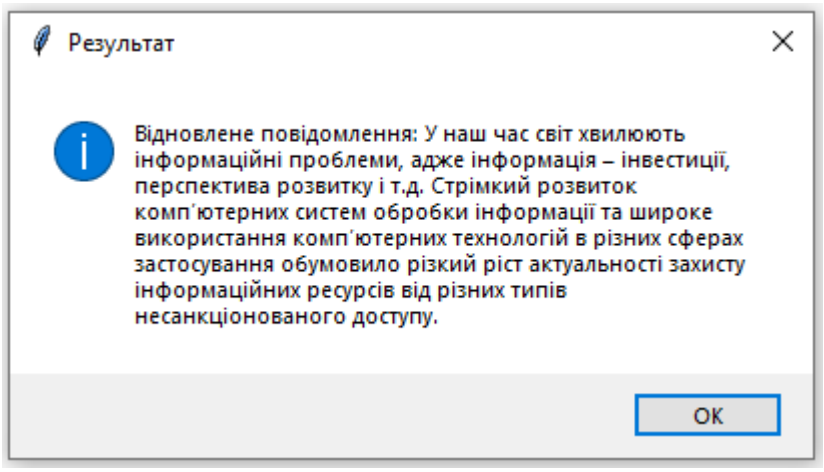

Рисунок 3.20 – Результат дешифрування

Порівнявши вихідні та дешифровані (рисунок 3.21) дані, можна переконатися у правильності виконаних операцій шифрування та дешифрування файлів. Це підтверджує, що засіб працює належним чином і не порушує цілісність даних при обробці файлів.

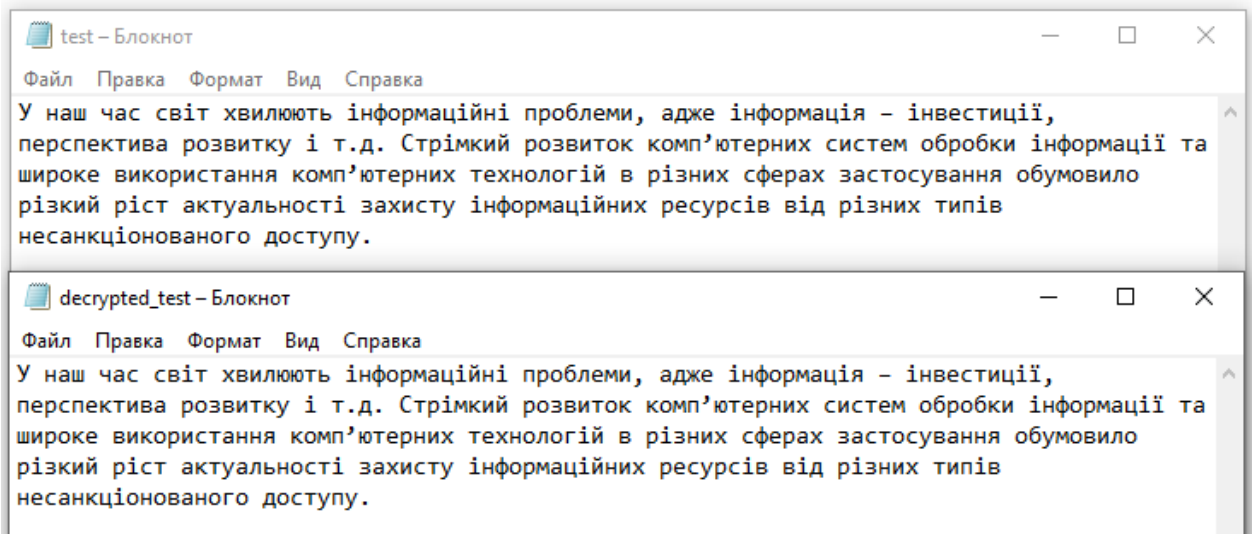

Рисунок 3.21 – Порівняння результатів

Для шифрування коротких повідомлень передбачено використання текстового режиму. Користувач може ввести текст безпосередньо у відповідне поле для шифрування. Після цього користувач може зашифрувати введений текст і побачити результат (рисунок 3.22).

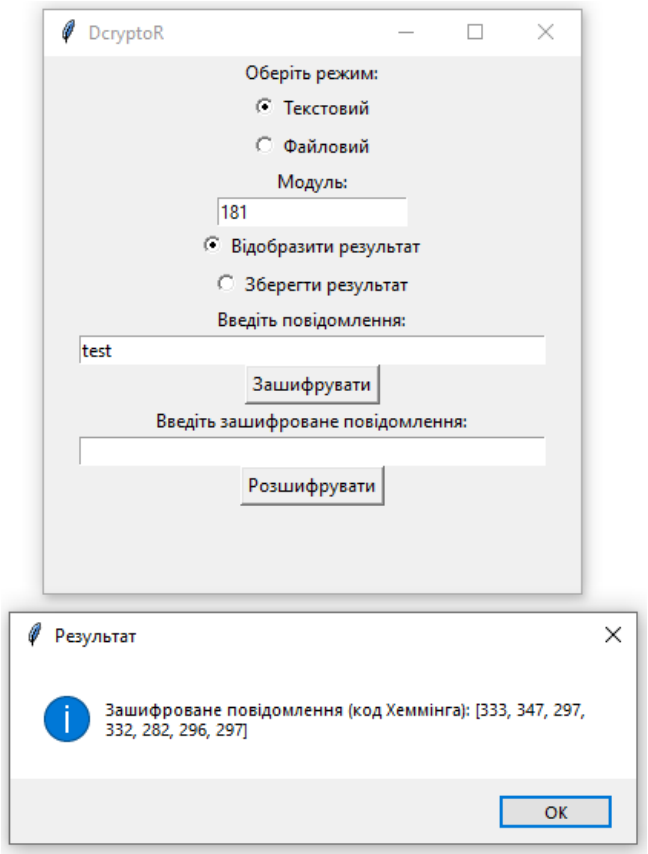

Рисунок 3.22 – Результат шифрування короткого повідомлення

Для тестування коректності роботи функцій виправлення помилок при дешифрування введемо дані з навмисною помилкою (рисунок 3.23).

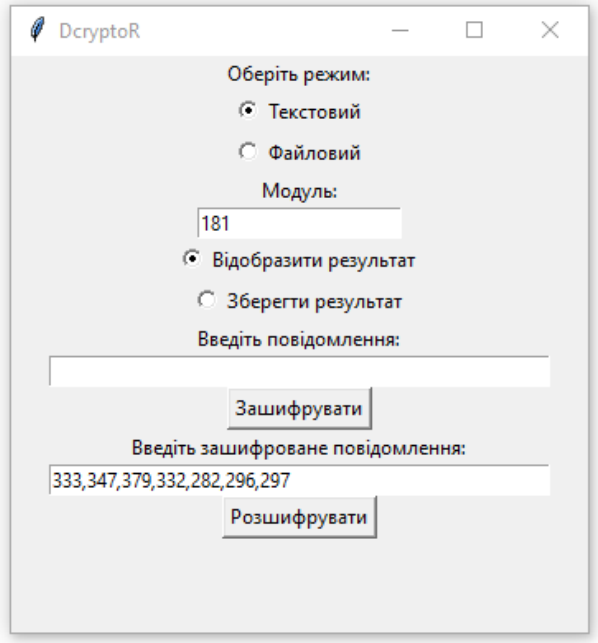

Рисунок 3.23 – Введення даних з помилкою

Результат показав, що функція виправлення помилок працює коректно і здатна відновлювати правильні дані навіть у випадку їх пошкодження під час передачі (рисунок 3.24).

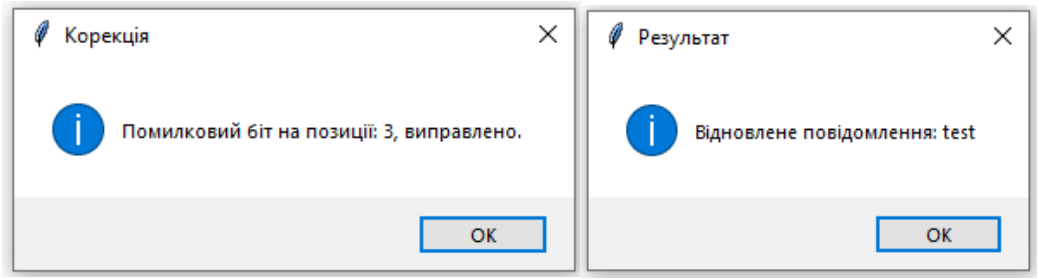

Рисунок 3.24 – Результат виявлення та виправлення помилки

Однією з важливих особливостей програми є використання значення модуля p або основи системи числення, яка застосовується для виконання шифрування як ключа для шифрування. Це забезпечує додатковий рівень безпеки, оскільки якщо це значення невідоме то неможливо здійснити правильне дешифрування даних. У даному тестуванні було перевірено, як програма поводиться, якщо для дешифрування введено неправильне значення модуля (рисунок 3.25).

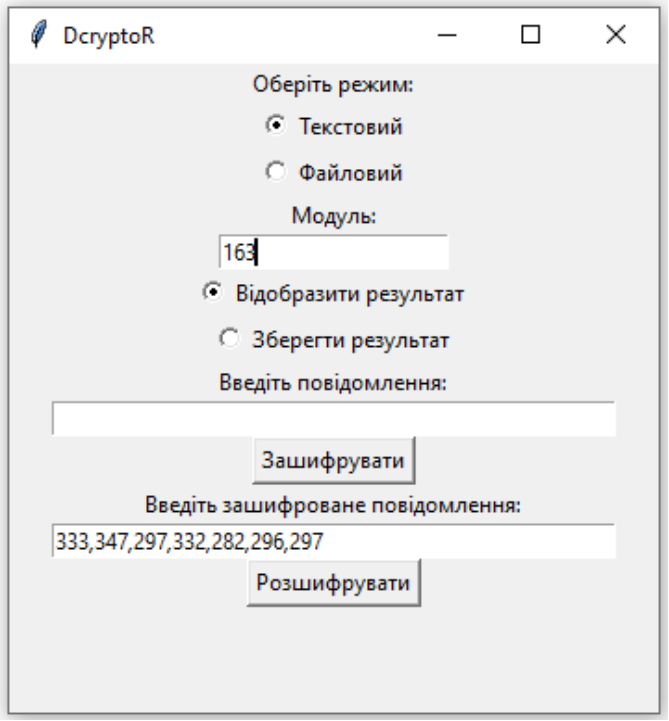

Рисунок 3.25 – Спроба дешифрувати дані з неправильним значенням модуля

Очікується, що при відсутності правильного значення модуля програма не зможе виконати дешифрування, а результат буде некоректним.

Після напискання кнопки «Розшифрувати» програма сприймає дані як спотворені і намагається визначити позиції помилки в даних. Хоча зашифровані дані було введено без помилок.

Рисунок 3.26 ілюструє наявність хибно визначеної помилки в позиції.

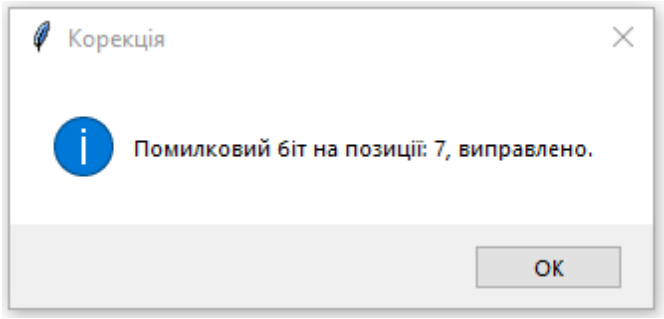

Рисунок 3.26 – Хибно визначена позиція помилки

При подальшому дешифруванні даних результат виявляється некоректним. Рисунок 3.27 демонструє некоректний результат дешифрування, що вказує на неможливість відновлення даних без точного значення модуля.

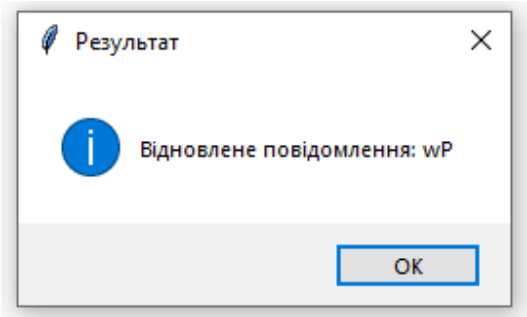

Рисунок 3.27 - Некоректний результат дешифрування

Як очікувалося, внаслідок введення неправильного модуля, дані не можуть бути успішно відновлені до оригінального вигляду. Це підтверджується в процесі тестування.

Під час тестування було підтверджено, що без введення правильного

модуля програма не може здійснити операцію дешифрування, що забезпечує захист даних. Це робить програму надійною для використання у ситуаціях, де важливий високий рівень конфіденційності. Коли значення модуля вводиться правильно, шифрування та дешифрування працюють без помилок. Це підтверджує, що модуль використовується належним чином як ключ і що без нього безпечне відновлення даних неможливе.

Всі основні функції програмного засобу були перевірені та протестовані на наявність помилок.

Програма успішно шифрує та дешифрує дані, зберігає результат у файли, правильно обробляє помилки і забезпечує користувачам зручний інтерфейс для роботи з даними. Тестування підтвердило правильність виконання всіх основних операцій та забезпечило високий рівень надійності програмного засобу.

4 ТЕХНІКО-ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ

4.1 Аналіз ринку

Програмне забезпечення для безпечного обміну інформацією та збереження цілісності даних займає важливе місце в сучасних інформаційних технологіях. Основною метою таких програм є запобігання НСД та втратам інформації під час передачі.

На ринку існує широке коло продуктів, які реалізують методи шифрування та контролю цілісності даних. Однак більшість сучасних рішень орієнтовані або лише на криптографічний захист або на корекцію помилок у комунікаційних системах. Поєднання цих підходів у рамках єдиного програмного засобу забезпечує баланс між безпекою та виправленням можливих спотворень, що є особливістю представленої розробки. Розроблюваний засіб використовує комбінацію ASCII-перетворення та коду Хеммінга для шифрування та виправлення помилок у переданих повідомленнях. Це дозволяє не тільки забезпечити захист даних, але й підвищити їх надійність при передачі через нестабільні КЗ.

Ринок інформаційної безпеки постійно змінюється, і для збереження актуальності програмних засобів необхідно здійснювати регулярні оновлення та вдосконалення алгоритмів. Відсутність оновлень може призвести до зниження рівня захисту даних та втрати довіри з боку користувачів. Тому запропонований програмний засіб має конкурентні переваги, оскільки поєднує методи захисту конфіденційності та корекції помилок, що робить його ефективним рішенням для забезпечення безпечного та надійного обміну даними.

4.2 Розрахунок витрат на розробку проєкту

Розробка та реалізація проєкту є складним процесом, що передбачає врахування ряду економічних факторів. Перед початком проєктування було проведено аналіз ринку, включаючи оцінку попиту, динаміку його розвитку та

актуальність програмних рішень для безпечного обміну даними. Результати дослідження підтвердили, що ринок має високий попит на інноваційні методи шифрування та корекції помилок. Відкрита конкуренція на цьому ринку сприяє необхідності створення економічно ефективного продукту, що забезпечуватиме безпеку даних із мінімальними витратами на обслуговування та впровадження.

Формування бюджету розробки $С_{розр}$ базується на ключових статтях витрат та визначаються як

$$С_{розр} = С_{зп} + с_{вн} + С_{ел} + С_{пр} + С_{ін},$$

де $С_{зп}$ - затрати на заробітну плату розробникам;

$с_{вн}$ - внески до дофндів (ЄСВ - єдиний соціальний внесок);

$С_{ел}$ - затрати на електричну енергію;

$С_{пр}$ - затрати на придбання чи оренду ПЗ й технічних засобів;

$С_{ін}$ - інші затрати.

Заробітна плата розраховується для кожного з учасників проєкту, відповідно до виразу

$$С_{зп} = Т_{розр} \cdot Р_{год},$$

де $Т_{розр}$ – тривалість розробки проєкту (год);

$Р_{год}$ – оплата праці (погодинна).

Для оцінки витрат на розробку програмного засобу для надійного і швидкого обміну інформацією з корекцією помилок було здійснено розрахунок заробітної плати основних виконавців проєкту:

- програміст Junior - відповідальний за реалізацію базової логіки, модулів кодування/декодування, інтерфейс користувача.
- програміст Middle - виконує проєктування архітектури, реалізацію алгоритмів корекції помилок, оптимізацію продуктивності.
- тестувальник - перевіряє правильність роботи програми, тестує обробку помилок та стійкість до збоїв.

Розрахунок затрат на зарплату наведено в таблиці 4.1.

Таблиця 4.1 –Розрахунок заробітньої платні

Посада виконавця	Оплата праці	Тривалість розробки, год	Кількість чоловік	Сума з/п, грн
Програміст - juior	150	80	1	12000
Програміст - middle	250	60	1	15000
Тестувальник	180	40	1	7200
Разом				34200

Отримані результати розрахунків дають можливість зробити висновок про доцільність розробки та впровадження проєкту.

Для розрахунку витрат внесків до ЄСВ використовуємо формулу:

$$с_{вн} = с_{зп} \cdot К_{есв},$$

де $К_{есв}$ – коефіцієнт ЄСВ, що є стандартною ставкою для більшості платників, включаючи роботодавців та фізичних осіб-підприємців і складає 22% від нарахованої заробітної плати.

$$с_{вн} = 34200 \cdot 0{,}22 = 7524 \text{ грн.}$$

Відповідно загальна сума затрат для оплати праці складає

$$с_{праця} = с_{зп} + с_{вн} = 34200 + 7524 = 41724 \text{ грн.}$$

Для визначення витрат на електроенергію використовується така формула:

$$С_{ел} = Р \cdot Т \cdot С_{кВт},$$

де Р – потужність обладнання, кВт;

Т - тривалість роботи обладнання для реалізації проєкту, год;

$С_{кВт}$ – тариф, зокрема 6,9 грн/кВт·год для непобутових споживачів.

Кожен виконавець використовував окремий комп'ютер протягом свого робочого часу. Розрахунок витрат на електроенергію наведено в таблиці 4.2.

Таблиця 4.2 –Розрахунок витрат на електричну енергію

Пристрій	P	T	$C_{ел}$
ПК 1	0,2	80	110,4
ПК 2	0,25	60	103,5
ПК 3	0,15	40	41,4
Монітор	0,04	180	49,68
Принтер	0,1	6	4,14

Загальні витрати на електроенетгію складають

$$C_{ел} = 110,4 + 103,5 + 41,4 + 49,68 + 4,14 = 309,12 грн.$$

Розрахунок затрат для придбання чи оренду програмних та технічних засобів здійснюється за формулами

$$C_{пз} = C \cdot N + C_{підп}; \qquad C_{об} = \sum_{i=0}^{n} C_i \cdot N_i,$$

де $C_{пз}$ – загальні затрати на ПЗ;

$C_{об}$ – загальні затрати на технічні засоби;

C – вартість одного засобу,

N – кількість;

$C_{підп}$ – витрати на підписку;

В процесі розробки програмного засобу використано ПЗ наведене в таблиці 4.3

Таблиця 4.3 – Програмне забезпечення

Назва	Кількість	Вартість	Сума
Python	3	0	0
VS Code / PyCharm Community	2	0	0
Git+ GitHub	3	0	0

У процесі розробки програмного засобу для надійного і швидкого обміну інформацією з корекцією помилок придбання ліцензійного

програмного забезпечення не передбачалося. Для розробки використовувалася мова програмування Python та супутні бібліотеки з відкритим вихідним кодом, які знаходяться у вільному доступі і не вимагають ліцензійних платежів.

Для розробки програмного засобу використовувалося наявне обладнання, яке забезпечує достатній рівень обчислювальних ресурсів для виконання поставлених завдань. Тому оренда або закупівля додаткових серверів чи іншого обладнання не проводилася. Витрати проєкту включають амортизаційні витрати на обладнання за період розробки за період розробки - 3 місяці буде використано пропорційну частку від річної норми амортизації. Розрахунок наведено в таблиці 4.4.

Таблиця 4.4 – Технічні засоби

Обладнання	Кількість, шт	Вартість, грн	Амортизація, %	Сума, грн
ПК	1	30000	5	1500
ПК	1	35000	5	1750
ПК	1	18000	5	900
Монітор	3	7000	2,5	525
Принтер	1	10000	2,5	250

Відповідно

$$C_{об} = 1500 + 1750 + 900 + 525 + 250 = 4925 \text{ грн.}$$

Загальна сума затрат складе

$$C_{пр} = C_{пз} + C_{об} = 0 + 4925 = 4925 \text{ грн.}$$

До інших витрат можна віднести канцелярські витрати:

- папір для документації: 100 грн;
- ручки, маркери: 50 грн.

$$C_{канц} = \sum_{i=0}^{n} C_i \cdot N_i = 100 + 50 = 150 \text{ грн.}$$

До цієї категорії відносяться також витрати на зв'язок:

- інтернет (3 місяці): 900 грн;
- мобільний зв'язок: 450 грн.

$$C_{зв'яз} = C_{моб} + C_{інтернет} + C_{інші} = 900 + 450 = 1350 грн.$$

Транспортні витрати включають поїздки на зустрічі та консультації, відповідно $C_{транс} = 300$ грн.

$$C_{ін} = C_{канц} + C_{зв'яз} + C_{транс} = 150 + 1350 + 300 = 1800 \text{ грн.}$$

Розрахунок загальної суми витрат на реалізацію проєкту (таблиця 4.5) враховує не лише основні витрати на оплату праці, а й супутні витрати, що формують реальну вартість розробки, включно з інфраструктурою, програмним забезпеченням і матеріальними ресурсами.

Таблиця 4.5 – Загальні витрати на розробку

Стаття витрат	Сума, грн
Заробітна плата	34200
ЄСВ	7524
Витрати на електроенергію	309,12
Витрати на ПЗ та технічні засоби	4925
Інші витрати	1800
Разом	50458,12

Загальні витрати на розробку програмного засобу для надійного і швидкого обміну інформацією з корекцією помилок складають $C_{розр} = 50458{,}12$ грн.

Найбільшу частку витрат – 67,8% загальної вартості розробки становлять витрати на оплату праці з нарахуванням, що підтверджує значний внесок трудових ресурсів у проєкт, 14,9% є обов'язковими соціальними платежами. Витрати на електроенергію 0,6% та на програмне забезпечення і технічні засоби 9,8% є відносно невеликими, але необхідними для забезпечення нормального процесу розробки. Інші витрати складають 3,6% від

загальної суми враховують додаткові або допоміжні витрати, пов'язані з реалізацією проєкту. Структура витрат є характерною для проєктів розробки програмного забезпечення та вказує на ефективний розподіл ресурсів із пріоритетом на оплату праці, що забезпечує якісну розробку програмного продукту.

Для розрахунку економічного ефекту від впровадження програмного засобу для надійного і швидкого обміну інформацією з корекцією помилок доцільно врахувати наступні оцінки:

1. Економія часу працівників $E_{\text{час}}$ завдяки автоматизації процесів шифрування, дешифрування та корекції помилок, що значно скорочує час обробки кожної операції передачі даних складе

$$E_{\text{час}} = \left(\text{T}_{\text{ручн}} - \text{T}_{\text{авт}}\right) \cdot \text{P}_{\text{год}} \cdot N,$$

де $\text{T}_{\text{ручн}}$ – час (в годинах), який витрачався на виконання задачі вручну;

$\text{T}_{\text{авт}}$ - час (в годинах), який витрачається на виконання тієї ж задачі після автоматизації;

$\text{P}_{\text{год}}$ - середня вартість години праці (112,5 грн/год);

N - кількість повторень задачі за обраний період (500 операцій/місяць).

В таблиці 4.6 наведені дані для розрахунку ефекту через економію часу працівників.

Таблиця 4.6 – Витрати часу на обробку даних

Операція	Тривалість, год	
	Поточний стан	Після впровадження
Час на підготовку даних для безпечної передачі	0,33	0,03
Час на перевірку цілісності отриманих даних	0,25	0,02
Час на виявлення та виправлення помилок при передачі	0,17	0,02
Загальний час виконання	0,75	0,07

Економічний ефект складе

$$E_{\text{час}} = (0{,}75 - 0{,}07) \cdot 112{,}5 \cdot 500 * 12 = 459000 \text{ грн/рік.}$$

2. Зменшення кількості помилок $E_{\text{помилки}}$ за рахунок використання коду хеммінга для автоматичного виявлення та виправлення помилок при передачі даних мінімізує кількість збоїв та необхідність повторної передачі становить

$$E_{\text{помилки}} = N_{\text{пом}} \cdot \text{С}_{\text{одн}},$$

де $N_{\text{пом}}$ – кількість помилок, які вдається усунути за рік;

$\text{С}_{\text{одн}}$ – вартість корекції однієї помилки, грн.

Дані для розрахунку наведено в таблиці 4.7.

Таблиця 4.7 - Аналіз помилок при передачі даних

Показник	Поточний стан	Після впровадження
$N_{\text{пом}}$	60 помилок/рік (1% від операцій)	6 помилок/рік (0,1% від операцій)
$\text{С}_{\text{одн}}$	1200	1200

Розрахунок економії від зменшення помилок

$$E_{\text{помилки}} = (60 -\cdot 6) \cdot 1200 = 64800 \text{ грн/рік.}$$

3. Підвищення безпеки даних $E_{\text{безпека}}$ досязається щляхом застосування надійних алгоритмів шифрування, що знижують ризик витоку конфіденційної інформації та пов'язані з цим фінансові втрати.

$$E_{\text{безпека}} = \left(P_{\text{пс}} - P_{\text{впр}}\right) \cdot \text{С}_{\text{інц}} \cdot N.$$

До впровадження програмного засобу надійного та безпечного обміну даними з корекцією помилок ймовірність витоку інформації $P_{\text{пс}}$ становить 2% на рік. Це означає, що з кожних 100 випадків передачі даних у середньому два завершувалися інцидентами, пов’язаними з компрометацією інформації. Після впровадження засобу ймовірність витоку $P_{\text{впр}}$ знизиться до 0,1% на рік, 1 інцидент витоку на 1000 випадків, що свідчитиме про підвищення рівня інформаційної безпеки. При цьому вартість одного інциденту безпеки $\text{С}_{\text{інц}}$

(включаючи втрати від витоку конфіденційної інформації, витрати на відновлення, штрафи, зниження репутації тощо) залишається незмінною та становить 150000 грн.

В результаті розрахунку економучного ефекту від впровадження розробленого програмного засобу отримаємо

$$E_{\text{безпека}} = (0{,}02 -\cdot 0{,}001) \cdot 15000 \cdot 6000 = 1710000 \text{ грн/рік.}$$

Сумарний економічний ефект розраховується як:

$$E_{\text{загальний}} = E_{\text{час}} + E_{\text{помилки}} + E_{\text{безпека}}$$

і складе

$$E_{\text{загальний}} = 459000 + 64800 + 1710000 = 2233800 \text{ грн/рік.}$$

Термін окупності $T_{\text{ок}}$ проєкту визначається як відношення загальних витрат на розробку$\text{C}_{\text{розр}}$ до економічного ефекту E, тобто показує, за який період часу витрати на впровадження програмного засобу повністю компенсуються за рахунок отриманої економії або прибутку.

$$T_{\text{ок}} = \frac{\text{C}_{\text{розр}}}{\text{E}} = \frac{50458{,}12}{2233800} = 0{,}0266 \text{ років.}$$

Перевівши отримані дані отримаємо 8 днів, що свідчить про високу ефективність і доцільність впровадження розробленого програмного засобу.

Коефіцієнт ефективності $K_{\text{еф}} = \frac{\text{E}}{\text{C}_{\text{розр}}} = 44{,}27$. Це означає, що на кожну вкладену гривню розробки проєкт приносить понад 44 гривні економічного ефекту. Це свідчить про високу рентабельність проєкту.

4.3 Обгрунтування необхідності розробки

В умовах цифрової трансформації все більше організацій та приватних користувачів стикаються з проблемами НСД, витоку конфіденційної інформації та пошкодження даних під час передачі через незахищені комунікаційні канали. Перелічені проблеми є актуальними для сучасного інформаційного середовища, де збереження цілісності та конфіденційності даних стає пріоритетом. Це вимагає використання ефективних рішень для захисту інформації.

На ринку інформаційних технологій представлені засоби, які реалізують методи шифрування та контролю цілісності даних. Однак більшість із них орієнтовані на забезпечення лише однієї з двох функцій. Ннаприклад, алгоритми AES, RSA фокусуються на криптографічному захисті, а ККП - на корекції помилок у комунікаційних системах. Поєднання цих підходів у запропонованому рішенні дозволяє досягти високої стійкості до атак і помилок під час передачі. Запропоноване рішення не лише забезпечує безпечний обмін даними, але й надає можливості його адаптації та подальшої інтеграції сучасних алгоритмів шифрування та корекції помилок.

Впровадження цього програмного засобу в сучасні інформаційні системи дозволить організаціям та приватним користувачам забезпечити надійний захист своїх даних. Розробка і реалізація програмного засобу є доцільною та економічно вигідною, оскільки він відповідає вимогам сучасної інформаційної безпеки та забезпечує конкурентні переваги на ринку технологій захисту даних.

ВИСНОВКИ

1. Проведено аналіз методів захисту від НСД дозволив визначити, що для забезпечення конфіденційності даних найбільш ефективними є алгоритми шифрування даних, що забезпечують високий рівень захисту переданої інформації.

2. Проаналізовано механізми забезпечення цілісності даних під час їх передачі ненадійними КЗ. Встановлено, що використання коригуючих кодів для виявлення та виправлення помилок, які виникають через збої у комунікаційних каналах, значно підвищує надійність переданих даних.

3. Проведено дослідження властивостей ККП показали, що для реалізації проєктованого програмного засобу оптимальним є використання кодів Хеммінга. Це зумовлено простотою реалізації, ефективним використанням обчислювальних ресурсів та здатністю забезпечити гарантоване виправлення помилок при передачі даних.

4. Досліджено алгоритми кодування та декодування, а також механізми виявлення та корекції помилок коду Хеммінга при класичній бінарній реалізації та з використанням систем числення з основою p>2. У результаті було визначено можливості їх застосування для забезпечення безпечного обміну даними в проєктованому програмному засобі.

5. На основі проведених досліджень розроблено алгоритми кодування та декодування даних, які поєднують перетворення вихідних даних за допомогою криптографічних операцій та модульної арифметики й корекцію помилок, що забезпечує швидкий та надійний обмін інформацією без ризику її пошкодження чи втрати.

6. Створено користувацький інтерфейс, який забезпечує ефективність роботи з програмним засобом і підвищує його доступність для кінцевого користувача.

7. Проведено тестування програмного засобу з використанням різних сценаріїв використання, яке підтвердило його ефективність та надійність у

забезпеченні безпеки та цілісності даних.

8. Проведено техніко-економічне обґрунтування, яке підтвердило економічну доцільність розробки програмного засобу. Впровадження проєктованого продукту дозволить підвищити рівень безпеки та знизити ризики витоку чи пошкодження даних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. SSL/TLS simplified. URL: https://medium.com/@jayampathiadhikari/ssl-tls-simplified-c3c1f08051b2 (дата звернення: 22.11.2024).

2. Virtual Private Network (VPN) URL: https://cyberhoot.com/cybrary/virtual-private-network-vpn/ (дата звернення: 22.11.2024).

3. What is Symmetric Encryption? Symmetric-Key Algorithms. URL: https://www.clickssl.net/blog/what-is-symmetric-encryption (дата звернення: 28.11.2024).

4. What is Asymmetric Encryption and How it Works? URL: https://www.clickssl.net/blog/what-is-asymmetric-encryption (дата звернення: 28.11.2024).

5. Advanced Encryption Standard (AES). URL: https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard (дата звернення: 16.12.2025).

6. Advanced Encryption Standard (AES). URL: https://www.geeksforgeeks.org/advanced-encryption-standard-aes/ (дата звернення: 16.12.2024).

7. Miah M. S. Introduction to Cryptography and Advanced Encryption Standard. *International Journal of Research and Scientific Innovation*. 2024. Vol. XI, no. IX. P. 776–783. URL: https://doi.org/10.51244/ijrsi.2024.1109065 (дата звернення: 17.12.2024).

8. Data Encryption Standard. URL: https://www.tutorialspoint.com/cryptography/data_encryption_standard.htm (дата звернення: 17.12.2024).

9. Paar C., Pelzl J., Güneysu T. The Data Encryption Standard (DES) and Alternatives. *Understanding Cryptography*. Berlin, Heidelberg, 2024. P. 73–110. URL: https://doi.org/10.1007/978-3-662-69007-9_3 (дата звернення: 18.12.2024).

10. DES Data Encryption Standard. URL: https://jethrojeff.com/ (дата

звернення: 18.12.2024).

11. RSA Algorithm in Cryptography: Rivest Shamir Adleman Explained. URL: https://www.splunk.com/en_us/blog/learn/rsa-algorithm-cryptography.html (дата звернення: 24.12.2024).

12. Koç Ç. K., Özdemir F., Ödemiş Özger Z. Rivest-Shamir-Adleman Algorithm. *Partially Homomorphic Encryption*. Cham, 2021. P. 37–41. URL: https://doi.org/10.1007/978-3-030-87629-6_3 (дата звернення: 24.12.2024).

13. Rivest-Shamir-Adleman (RSA) in Cryptography. URL: https://www.includehelp.com/cryptography/rivest-shamir-adleman-rsa-in-cryptography.aspx (дата звернення: 25.12.2024).

14. Purnama I., Suryadi S., Watrianthos R., Irmayani D., Nasution M. Android-Based Text Message Security Application With Rivest Method, Shamir, Adleman (RSA). *International journal of scientific & technology research*. Vol. 8, Issue 05, 2019. P. 41-43 URL: https://www.ijstr.org/final-print/may2019/Android-based-Text-Message-Security-Application-With-Rivest-Method-Shamir-Adleman-rsa.pdf (дата звернення: 25.12.2024).

15. Yan Y. The Overview of Elliptic Curve Cryptography (ECC). *Journal of Physics: Conference Series*. 2022. Vol. 2386, no. 1. P. 012019. URL: https://doi.org/10.1088/1742-6596/2386/1/012019 (дата звернення: 25.12.2024).

16. Laassiri J., Berguig Y., Hassi S. Elliptic Curve Cryptography (ECC) for a Lightweight Public Key Infrastructure (PKI). *Distributed Sensing and Intelligent Systems*. 2022. P. 1–12. URL: https://doi.org/10.1007/978-3-030-64258-7_1 (дата звернення: 25.12.2024).

17. Mishra A., Singh M. Elliptic Curve Cryptography (ECC) for Security in wireless Sensor Network, *International journal of engineering research & technology* (IJERT) 2012. Vol. 01, Issue 03 1. 2278-0181. URL: https://www.ijert.org/research/elliptic-curve-cryptography-ecc-for-security-in-wireless-sensor-network-IJERTV1IS3046.pdf (дата звернення: 25.12.2024).

18. What is PGP? URL: https://www.wallarm.com/what/what-is-pgp-encryption-everything-you-need-to-know (дата звернення: 16.01.2025).

19. Demystifying OpenSSL: How It Works to Secure Your Data. URL: https://medium.com/@MakeComputerScienceGreatAgain/demystifying-openssl-how-it-works-to-secure-your-data-ba2792474e1e (дата звернення: 16.01.2025).

20. Understanding zlib. URL: https://www.euccas.me/zlib/ (дата звернення: 17.01.2025).

21. Wireshark. URL: https://www.wireshark.org (дата звернення: 17.01.2025).

22. Кодування сигналів в електронних системах. Частина 3. Способи кодування сигналів: Том 1. Натуральні, ефективні та лінійні коди / С.В. Денбновецький, І.В. Мельник, Л.Д. Писаренко ; КПІ ім. Ігоря Сікорського. – Київ : КПІ ім. Ігоря Сікорського, 2021. – 470с.

23. Кодування сигналів в електронних системах. Частина 3. Способи кодування сигналів. Том 2. Групові, ітеративні та згорткові коди. / С.В. Денбновецький, І.В. Мельник, Л.Д. Писаренко ; КПІ ім. Ігоря Сікорського. – Київ : КПІ ім. Ігоря Сікорського, 2021. – 634с.

24. Давлетова А. Побудова кодів хеммінга в скінченних полях галуа. *Herald of Khmelnytskyi National University. Technical sciences*. 2024. Т. 333, № 2. С. 28–34. URL: https://doi.org/10.31891/2307-5732-2024-333-2-4 (дата звернення: 05.02.2025).

25. Давлетов Р., Кузик В. (2024) Програмний засіб для захищеного обміну даними з корекцією помилок. Збірник матеріалів науково-практичної конференції молодих вчених, аспірантів та студентів «Кібербезпека та комп’ютерно-інтегровані технології» (КБКІТ - 2024), Тернопіль, 2024. - с. 74-77.

Додатки

Додаток А

Скрипт програмного засобу

```
import tkinter as tk
from tkinter import messagebox, filedialog
def is_power_of_two(n):
    return n & (n - 1) == 0 and n != 0

def calculate_hamming_code(data, mod):
    n = len(data)
    r = 0
    while 2**r < n + r + 1:
        r += 1
    hamming_code = [0] * (n + r)
    j = 0
    k = 0
    for i in range(1, n + r + 1):
        if i == 2**k:
            k += 1
        else:
            hamming_code[i - 1] = int(data[j])
            j += 1
    for i in range(r):
        parity_index = 2**i - 1
        xor = 0
        for j in range(1, n + r + 1):
            if (j >> i) & 1:
                xor = xor + hamming_code[j - 1]
        hamming_code[parity_index] = xor % mod
```

```
    return hamming_code
def add_mod_to_hamming_code(hamming_code, mod):
    return [x + mod for x in hamming_code]
def remove_mod_from_hamming_code(hamming_code, mod):
    return [x - mod for x in hamming_code]

def get_control_symbols(encoded_data):
    control_symbols = []
    for i in range(len(encoded_data)):
        if is_power_of_two(i + 1):
            control_symbols.append(encoded_data[i])
    return control_symbols
def get_information_symbols(encoded_data):
    information_symbols = []
    for i in range(1, len(encoded_data)):
        if not is_power_of_two(i + 1):
            information_symbols.append(encoded_data[i])
    return information_symbols

def correct_hamming_code(encoded_data, mod):
    information_symbols = get_information_symbols(encoded_data)
    new_control_symbols = calculate_control_symbol(information_symbols, mod)
    prior_symbols_from_data = get_control_symbols(encoded_data)
    compare_byte_arr = []
    for i in range(len(prior_symbols_from_data)):
        compare_byte_arr.append(prior_symbols_from_data[i]                    !=
new_control_symbols[i])
    index_e = bool_list_to_int(compare_byte_arr[::-1]) - 1
    if index_e >= 0 and index_e < len(encoded_data):
        control_symbol_position = 1
```

```
        for i in range(len(encoded_data)):
            if is_power_of_two(i + 1) and (i + 1) & (index_e + 1) != 0:
                control_symbol_position = i
                break
        original_control_symbol_value = encoded_data[control_symbol_position]
        data_sum = 0
        for i in range(len(encoded_data)):
            if (i + 1) & (1 << control_symbol_position) != 0 and not is_power_of_two(i
+ 1):
                data_sum += encoded_data[i]
        corrected_value = (original_control_symbol_value - (data_sum -
encoded_data[index_e]) % mod)
        encoded_data[index_e] = corrected_value
        messagebox.showinfo("Корекція", f"Помилковий біт на позиції: {index_e +
1}, виправлено.")
    else:
        messagebox.showinfo("Корекція", "Помилок не виявлено.")
    return encoded_data

def calculate_control_symbol(data, mod):
    n = len(data)
    r = 0
    while 2**r < n + r + 1:
        r += 1
    hamming_code = [0] * (n + r)
    j = 0
    k = 0
    for i in range(1, n + r + 1):
        if i == 2**k:
            k += 1
```

```
            else:
                hamming_code[i - 1] = int(data[j])
                j += 1
        control_symbol = []
        for i in range(r):
            parity_index = 2**i - 1
            xor = 0
            for j in range(1, n + r + 1):
                if (j >> i) & 1:
                    xor = xor + hamming_code[j - 1]
            control_symbol.append(xor % mod)
        return control_symbol

def bool_list_to_int(bool_list):
    int_list = [int(x) for x in bool_list]
    number = 0
    for bit in int_list:
        number = (number << 1) | bit
    return number

def text_to_numbers(text):
    return [ord(c) for c in text]
def numbers_to_text(numbers):
    return ''.join([chr(num) for num in numbers])

def select_file():
    return filedialog.askopenfilename(filetypes=[("Text files", "*.txt")])
def save_file(content, file_type="text"):
    if file_type == "text":
        file_path = filedialog.asksaveasfilename(defaultextension=".txt",
```

```
filetypes=[("Text files", "*.txt")])
        if file_path:
            with open(file_path, 'w', encoding='utf-8') as file:
                file.write(content)
            messagebox.showinfo("Результат", f"Файл збережено як: {file_path}")
    elif file_type == "binary":
        file_path = filedialog.asksaveasfilename(defaultextension=".bin",
filetypes=[("Binary files", "*.bin")])
        if file_path:
            with open(file_path, 'wb') as file:
                file.write(content)
            messagebox.showinfo("Результат", f"Файл збережено як: {file_path}")
def encrypt_file():
    try:
        file_path = select_file()
        if not file_path:
            return
        with open(file_path, 'r', encoding='utf-8') as file:
            file_content = file.read()
        mod = int(entry_mod.get())
        data = text_to_numbers(file_content)
        hamming_code = calculate_hamming_code(data, mod)
        protected_code = add_mod_to_hamming_code(hamming_code, mod)
        if var_output_type.get() == 'save':
            save_file(','.join(map(str, protected_code)), "text")
        else:
            messagebox.showinfo("Результат", f"Зашифроване повідомлення:
{','.join(map(str, protected_code))}")
    except ValueError:
        messagebox.showerror("Помилка", "Перевірте коректність введених
```

```
даних")
def decrypt_file():
    try:
        file_path = select_file()
        if not file_path:
            return
        with open(file_path, 'r', encoding='utf-8') as file:
            hamming_code_str = file.read()
        mod = int(entry_mod.get())
        hamming_code = [int(x) for x in hamming_code_str.split(',')]
        ontected_code = remove_mod_from_hamming_code(hamming_code, mod)
        corrected_code = correct_hamming_code(ontected_code, mod)
        data_symbols = get_information_symbols(corrected_code)
        decrypted_message = numbers_to_text(data_symbols)
        if var_output_type.get() == 'save':
            save_file(decrypted_message, "text")
        else:
            messagebox.showinfo("Результат", f"Відновлене повідомлення:
{decrypted_message}")
    except ValueError:
        messagebox.showerror("Помилка", "Перевірте коректність введених
даних")
def process_encryption():
    try:
        text_message = entry_message.get()
        mod = int(entry_mod.get())
        data = text_to_numbers(text_message)
        hamming_code = calculate_hamming_code(data, mod)
        protected_code = add_mod_to_hamming_code(hamming_code, mod)
        if var_output_type.get() == 'save':
```

```
            save_file(','.join(map(str, protected_code)), "text")
        else:
            messagebox.showinfo("Результат", f"Зашифроване повідомлення (код Хеммінга): {protected_code}")
    except ValueError:
        messagebox.showerror("Помилка", "Перевірте коректність введених даних")
def process_decryption():
    try:
        hamming_code_str = entry_encrypted.get()
        mod = int(entry_mod.get())
        hamming_code = [int(x) for x in hamming_code_str.split(',')]
        ontected_code = remove_mod_from_hamming_code(hamming_code, mod)
        corrected_code = correct_hamming_code(ontected_code, mod)
        information_symbols = get_information_symbols(corrected_code)
        decrypted_message = numbers_to_text(information_symbols)
        if var_output_type.get() == 'save':
            save_file(decrypted_message, "text")
        else:
            messagebox.showinfo("Результат", f"Відновлене повідомлення: {decrypted_message}")
    except ValueError:
        messagebox.showerror("Помилка", "Перевірте коректність введених даних")

def switch_mode(mode):
    if mode == 'file':
        label_message.pack_forget()
        entry_message.pack_forget()
        button_process.pack_forget()
```

```
        label_encrypted.pack_forget()
        entry_encrypted.pack_forget()
        button_decrypt.pack_forget()
        label_file.pack()
        button_encrypt.pack()
        button_decrypt_file.pack()
    elif mode == 'text':
        label_file.pack_forget()
        button_encrypt.pack_forget()
        button_decrypt_file.pack_forget()
        label_message.pack()
        entry_message.pack()
        button_process.pack()
        label_encrypted.pack()
        entry_encrypted.pack()
        button_decrypt.pack()
def on_mode_change():
    selected_mode = var_mode.get()
    switch_mode(selected_mode)
root = tk.Tk()
root.title("DcryptoR")
root.minsize(350, 350)
var_mode = tk.StringVar(value='text')
label_mode = tk.Label(root, text="Оберіть режим:")
radio_text = tk.Radiobutton(root, text="Текстовий", variable=var_mode,
value='text', command=on_mode_change)
radio_file = tk.Radiobutton(root, text="Файловий", variable=var_mode,
value='file', command=on_mode_change)
label_mod = tk.Label(root, text="Модуль:")
entry_mod = tk.Entry(root)
```

```
label_message = tk.Label(root, text="Введіть повідомлення:")
entry_message = tk.Entry(root, width=50)
button_process = tk.Button(root, text="Зашифрувати", command=process_encryption)
label_encrypted = tk.Label(root, text="Введіть зашифроване повідомлення:")
entry_encrypted = tk.Entry(root, width=50)
button_decrypt = tk.Button(root, text="Розшифрувати", command=process_decryption)
label_file = tk.Label(root, text="Виберіть файл для шифрування/дешифрування:")
button_encrypt = tk.Button(root, text="Шифрувати файл", command=encrypt_file)
button_decrypt_file = tk.Button(root, text="Розшифрувати файл", command=decrypt_file)
var_output_type = tk.StringVar(value='display')
radio_display = tk.Radiobutton(root, text="Відобразити результат", variable=var_output_type, value='display')
radio_save = tk.Radiobutton(root, text="Зберегти результат", variable=var_output_type, value='save')
label_mode.pack()
radio_text.pack()
radio_file.pack()
label_mod.pack()
entry_mod.pack()
radio_display.pack()
radio_save.pack()
switch_mode(var_mode.get())
root.mainloop()
```