Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи кібербезпеки перешкодостійкого кодування для відеоконференцзв’язку у бездротових мережах”

Виконав здобувач вищої освіти
IV курсу, групи КБ-21
ОПП «Кібербезпека»
спеціальності 125 «Кібербезпека»
_____ Іушин М.О.
«____» _____ 2025 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Смірнов С.А.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет
Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *бакалавр*
Галузь знань ._12 _____*“*Інформаційні технології*”*_____
Спеціальність *125 “*Кібербезпека*”*
Освітньо-професійна (освітньо-наукова) програма _*“*Кібербезпека*”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2025 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Іушину Максиму Олександровичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи кібербезпеки перешкодостійкого кодування для відеоконференцзв’язку у бездротових мережах*

2. Керівник роботи *Смірнов Сергій Анатолійович, канд. техн. наук, доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 57-02 від 17.01.2025 року

3. Строк подання студентом роботи до захисту *23.05.2025 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи кібербезпеки перешкодостійкого кодування для відеоконференцзв’язку у бездротових мережах*

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

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи кібербезпеки в промислову експлуатацію.
6. Висновки

6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Структурна схема системи кібербезпеки	*1 аркуш*
Функціональна схема системи кібербезпеки	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

7. Дата видачі завдання « 17 » січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2025 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2025 р.	
3.	Розробка моделі компонента	20.03.2025 р.	
4.	Розробка структур даних	25.03.2025 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2025 р.	
6.	Програмування алгоритмів	10.04.2025 р.	
7.	Оформлення ПЗ	17.04.2025 р.	
8.	Попередній захист роботи	23.05.2025 р.	

Дата видачі завдання
« 17 » січня 2025 р.

Підпис керівника
_____ Смірнов С.А.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2025 р.

Підпис здобувача
_____ Іушин М.О.
(прізвище та ініціали)

АНОТАЦІЯ

Іушин М.О. Програмне забезпечення системи кібербезпеки перешкодостійкого кодування для відеоконференцзв'язку у бездротових мережах. 125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи кібербезпеки перешкодостійкого кодування для відеоконференцзв'язку у бездротових мережах.

Метою розробки є програмне забезпечення системи кібербезпеки перешкодостійкого кодування для відеоконференцзв'язку у бездротових мережах.

Результат роботи – програмна реалізація системи кібербезпеки перешкодостійкого кодування для відеоконференцзв'язку у бездротових мережах.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ з ОС Windows 10/11.

Програму розроблено в середовищі Visual C#.

Ключові слова: кібербезпека, перешкодостійке кодування, відеоконференцзв'язок

ABSTRACT

Iushin M.O. Software for a cybersecurity system of interference-resistant coding for videoconferencing in wireless networks. 125 Cybersecurity. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the first (bachelor's) level of higher education, software has been developed, which is intended for a cybersecurity system of interference-resistant coding for videoconferencing in wireless networks.

The purpose of the development is software for a cybersecurity system of interference-resistant coding for videoconferencing in wireless networks.

The result of the work is a software implementation of a cybersecurity system of interference-resistant coding for videoconferencing in wireless networks.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software tools are provided.

The program can be used on PCs with Windows 10/11.

The program is developed in the Visual C# environment.

Keywords: cybersecurity, tamper-resistant coding, video conferencing

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

АБГШ	–	аддітивний білий гаусів шум
ВКЗ	–	відеоконференцзв'язок
ЕОМ	–	електронно-обчислювальна машина
ІМС	–	інтегральна мікросхема
ARQ	–	автоматичний запит повторної передачі
CD-DA	–	Compact Disc Digital Audio
CIRC	–	Cross Interleaved Reed Solomon Code
EAB	–	Embedded Array Block, блок зосередженої пам'яті
ECC	–	error-correcting code, код корекції помилок
FEC	–	метод прямої корекції помилок
LDPC	–	коди Галлагера
NAK	–	негативне підтвердження

ВСТУП

Актуальність теми. На сучасному рівні розвитку техніки існує проблема забезпечення високої якості відео-конференц-зв'язку в мережах з бездротовим доступом в Інтернет при недостатній продуктивності мобільних пристроїв. Рішення даної проблеми багато в чому пов'язане з вибором оптимальних алгоритмів кодування.

Споживчі властивості Інтернету визначаються в першу чергу – обсягом інформаційних ресурсів і послуг, у другу – характеристиками пристрою доступу до Мережі: рівень продуктивності, параметри екрана, програмне забезпечення, зручність використання. З погляду фахівця, відповідь буде неповним, якщо не враховувати якість каналу, за допомогою якого інтернет-сервіс надається користувачеві.

Безумовно, є безліч інтернет-послуг і протоколів, для яких якість каналу не має першорядного значення – наприклад, щоб забезпечити функціонування електронної пошти (звичайна переписка, без передачі «важких» файлів), досить установити з'єднання для завантажування інформації.

Для таких дій, як перегляд Web-сторінок, відеороликів, завантаження файлів і багато хто інші, важлива пропускна здатність каналу. А для онлайн-перегляду HD-відео пристрій повинне мати відповідну продуктивність: якщо ресурсів процесора недостатньо для декодування й відображення, якісного показу очікувати не доводиться.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи кібербезпеки перешкодостійкого кодування для відеоконференцзв’язку у бездротових мережах.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем перешкодостійкого кодування для відеоконференцзв’язку у бездротових мережах.

– Дослідження системи кібербезпеки перешкодостійкого кодування для відеоконференцзв’язку у бездротових мережах.

– Програмна реалізація системи кібербезпеки перешкодостійкого кодування для відеоконференцзв’язку у бездротових мережах.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі перешкодостійкого кодування для відеоконференцзв’язку у бездротових мережах.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки перешкодостійкого кодування для відеоконференцзв’язку у бездротових мережах, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

На зорі розвитку інтернет-сервісів ВКЗ, коли основні пристрої доступу в Інтернет – десктопи й лептопи – були не занадто потужними, а інтернет-канали низькошвидкістними, почалася розробка движків для ВКЗ-клієнтів, які повинні були вирішити обидві проблеми: забезпечити прийнятну якість відеозв'язку при поганих параметрах мережі й недостачі ресурсів комп'ютера.

Пізніше, коли мобільні бездротові гаджети ще не були поширені так широко, як зараз, але комп'ютери вже стали набагато могутніше, а мережа – досить продуктивної, з'явилися й досить успішно використовуються дотепер програмні клієнти ВКЗ, у яких реалізовані прості засоби контролю якості мережі й ресурсів. Однак у цьому полягає і їхній недолік: якщо подаваний потік аудіо- і відеопакетів випробовує навіть незначні пульсації й групові втрати, відео відтворюється в поганій якості: зображення «розвалюється», картинка й звук виявляються розсинхронізованими, до того ж останній передається з перебоями. При більше низьких параметрах каналу відео вже не прорисовується й з'єднання розривається. При цьому середнє значення пропускної здатності мережі може бути досить більшим. Це говорить про те, що для відео-конференц-зв'язку важлива не тільки пропускна здатність, але й інші характеристики каналу.

1.2 Область застосування

Бездротовий доступ для ВКЗ – не кращий варіант, тому що для нього характерні фонові групові втрати, тривалі плаваючі затримки при передачі пакетів і варіації часу надходження пакетів, тобто в наявності ті ж проблеми, які істотно знижують якість сеансу зв'язку ВКЗ. Крім того, при переході між зонами

точок доступу передача може надовго припинятися – аж до розривів з'єднань. При цьому невідповідність якості каналу потребам сервісу не може бути компенсовано можливостями встаткування, що забезпечує доступ до мережі: продуктивність більшості сучасних мобільних пристроїв, на які стрімко переміщається ВКЗ, поки недостатньо для обробки якісного відео з мінімальною затримкою.

Ситуація виглядає так, начебто для ВКЗ-клієнтів повернулися колишні часи. У результаті алгоритми ВКЗ для боротьби з названими вище явищами знову стали затребувані. Як приклад можна привести алгоритми мобільних ВКЗ-клієнтів для запобігання втрат пакетів у бездротових мережах.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки перешкодостійкого кодування для відеоконференцзв'язку у бездротових мережах, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

У цьому розділі ми розглянемо представлені на українському ринку моделі групових систем ВКЗ і виділимо найбільш перспективні. Для аналізу ми вибрали по дві групові системи початкового й бізнес рівня п'яти виробників: Polycom, Tandberg, Aethra, Sony і AddPac. Системи одного класу мають схожі технічні характеристики, але різною ціною, тому однієї із ключових завдань порівняння є виявлення моделей з найкращим співвідношенням «ціна/функціональні можливості».

Aethra Vega X3

Всі групові системи Aethra працюють абсолютно безшумно через відсутність вентиляторів. Система компактна й надійна, підтримує всі необхідні кодеки й стандарти, сумісна з устаткуванням інших виробників. Недавно з'явився убудований MCU підтримує до чотирьох абонентів (відеокодек H.263). Що стосується ціни, те це найдешевша модель серед розглянутих нами. З додаткових можливостей можна виділити інтеграцію із сервісами NetMeeting. Aethra Vega X3 має невелике число відеовходів і відеовиходів, однак у ряді систем початкового рівня лише вона підтримує підключення через DVI.

Sony PCS-1P

Дана модель сама функціональна, причому деякі можливості відрізняються від стандартних. Зокрема, є два входи XGA (опціонально), підтримується кодек MPEG-4 (на жаль, його можна використовувати тільки із системами ВКЗ від Sony), убудований MCU передбачає підключення до шести абонентів (відеокодек H.263), з декількох систем можна організувати каскад. При

досить конкурентоспроможній ціні PCS-1P має краще співвідношення «ціна/функціональні можливості» у своїй «вагарній» категорії. На жаль, незначні недоліки псують всю картину. І головні з них – невибагливий зовнішній вигляд і порівняно високий рівень шуму.

Polycom VSX 6000

Групова система VSX 6000 дозволяє зберегти зроблені вкладення в довгостроковій перспективі. За допомогою конференц-телефону Polycom SoundStation VTX1000 можна побудувати систему аудіоконференц-зв'язку, а потім розширити її за допомогою VSX 6000 до повноцінної системи ВКЗ. При цьому конференц-телефон інтегрується в структуру ВКЗ і служить як зовнішній мікрофон. З інших особливостей можна відзначити підтримку фірмового аудіокодеку Polycom StereoSurround (використовується тільки в системах Polycom). Головний мінус системи – надмірно висока ціна. При цьому модель не виділяється ні високими швидкостями з'єднання по IP, ні більшим числом відеовходів (немає навіть входу XGA для підключення ПК). Більше того, VSX 6000 не відрізняється стабільністю в роботі.

Tandberg 550 MXP

Як і всі системи ВКЗ Tandberg, групова система 550 MXP досить надійна в експлуатації. Але це, мабуть, єдине достоїнство даної моделі, тому що при мінімальному функціоналі вона має досить високу вартість. Як наслідок, Tandberg 550 MXP має найгірше співвідношення «ціна/функціональні можливості». Наявність такої опції, як порт USB, нічого не міняє.

AddPac VC-1000

Компанія AddPac – новий гравець на ринку групових систем відеоконференц-зв’язку. Система початкового рівня AddPac VC-1000 пропонує цікаві (у чомусь трохи дивні) можливості. Зокрема, вона забезпечує найвищу серед аналогів швидкість з'єднання по IP – до 4 Мбіт/с, при цьому підтримує не стандартний дозвіл 4CIF (704 x576), а комп'ютерне VGA (640 x 480). Функція DualVideo (стандарт H.239) не реалізована, але уведені два нових, не сумісних з

іншими, відеокодека. З аудіокодеків підтримуються тільки самі популярні різновиди. Порт USB придатний для підключення не тільки адаптера WiFi (як у конкурентів), але й різноманітної периферії: від клавіатури й миші до зовнішнього HDD. Все це наводить на думку, що при розробці VC-1000 виробник вирішив зробити акцент на сумісність у першу чергу із власними системами ВКЗ.

Aethra Vega X5

Як і модель початкового рівня, Vega X5 відрізняє компактність, безшумність і надійність. На наш погляд, ця групова система ВКЗ є самої стильною (до її розробки залучалися італійські дизайнери). Вона має найбільшу швидкість з'єднання по IP (до 4 Мбіт/с) і убудованим сервером MCU на вісім абонентів. Крім того, в Aethra Vega X5 реалізована функція автоматичного наведення камери й унікальна опція вибору розкладки екрана. Остання передбачає 12 варіантів розміщення «вікон» на екрані, у яких відображаються віддалені абоненти. За своєю ціною ця система найбільш доступна у своєму класі й має краще співвідношення ціна/функціональні можливості.

Sony PCS-G70 VP

У порівнянні з моделлю початкового рівня, Sony PCS-G70 VP має більше привабливий дизайн, надійністю в роботі, максимальної (4 Мбіт/с) швидкістю з'єднання по IP. Вона має дев'ять відеовиходів (на всі випадки життя й ще три-чотири про запас). MCU підтримує ті ж шість абонентів, але зате додана функція автоматичного наведення камери. У цілому відмінності від PCS-1P невеликі – головним чином, вони складаються в усуненні недоліків молодшої моделі. При цьому вартість виросла майже вдвічі, що зробило PCS-G70 VP однієї з найдорожчих систем бізнесу-класу.

Polycom VSX 7000s

Компанія Polycom є лідером ринку ВКЗ, однак можливості цієї моделі не вражають: швидкість з'єднання по IP до 2 Мбіт/с, убудований MCU до чотирьох абонентів і власний аудіокодек Polycom StereoSurround. Середньому функціоналові відповідає середня ціна.

Tandberg 990 MXP

Як і модель початкового рівня, Tandberg 990 MX характеризується високою надійністю й посередніми можливостями, як в Polycom VSX 7000s (MCU до чотирьох абонентів, швидкість з'єднання по IP до 2 Мбіт/с). Зовнішній вигляд копіює молодшу модель 550 MXP. При цьому вона виявилася самою дорогою серед своїх аналогів, а тому має найнижче співвідношення «ціна/функціональні можливості».

AddPac VC-2000

Групова система VC-2000 незначно відрізняється від моделі початкового рівня. Крім додавання MCU на чотири абонентів було збільшене число відеовходів/відеовиходів, швидкість з'єднання по IP як і раніше гранична для систем такого рівня – до 4 Мбіт/с. Таким чином, AddPac VC-2000 потенційно може стати найпривабливішою (з погляду «ціна/функціональні можливості») груповою системою ВКЗ на українському ринку.

2.2 Обґрунтування вибору засобів для побудови системи кібербезпеки та мови програмування

Visual C# – це об’єктноорієнтована мова програмування високого рівня загального призначення з відкритим кодом. Це визначення може бути важким для новачків, тому розглянемо кожну характеристику окремо, щоб зрозуміти, що вона означає:

– Відкритий вихідний код: це безкоштовно та доступно для подальших покращень, таких як додавання корисних функцій або виправлення помилок.

– Об’єктноорієнтована: заснована не на функціях, але в об’єктах з певними атрибутами й методами.

– Високий рівень: зручний для людини, а не для комп’ютера.

– Загальне призначення: можна використовувати для створення будь-яких програм.

Ця мова використовується в будь-якому програмному забезпеченні, про яке ви тільки можете подумати. Ви можете використовувати його для створення вебсайтів, штучного інтелекту, серверів, програмного забезпечення для бізнесу та багато іншого. Також застосовується в науці про дані, аналізі даних, машинному навчанні, інженерії даних, веброзробці, розробці програмного забезпечення та інших галузях.

Переваги та недоліки Visual C#

Переваги:

– Її легко читати, вчити та писати. Це мова програмування високого рівня з англійським синтаксисом. Це полегшує читання та розуміння коду. Її дійсно легко зрозуміти і вивчити, тому багато людей рекомендують Visual C# новачкам. Вам потрібно менше рядків коду для виконання того ж завдання в порівнянні з іншими основними мовами, такими як C/C++ та Java.

– Підвищує продуктивність. Це дуже продуктивна мова. Завдяки її простоті розробники можуть зосередитися на розв'язанні проблеми. Їм не потрібно витрачати багато часу на розуміння синтаксису або поведінку мови програмування. Ви пишете менше коду та виконуєте більше завдань.

– Інтерпретована мова. Visual C# мова, що інтерпретується, а це означає, що вона безпосередньо виконує код по рядку. Якщо сталася помилка, вона зупиняє подальше виконання та повідомляє про її виникнення. Вона показує лише одну помилку, навіть якщо у програмі їх кілька. Це спрощує налагодження.

– Динамічно типізована. Visual C# не визначає тип змінної, доки ми не запустимо код. Вона автоматично надає тип даних, коли відбувається процес виконання. Фахівець може не турбуватися про оголошення змінних та типи даних.

– Безкоштовна та з відкритим вихідним кодом. Ця мова постачається під схваленою OSI ліцензією з відкритим вихідним кодом. Це робить його безкоштовним для використання та розповсюдження. Ви можете завантажити

вихідний код, змінити його та навіть розповсюджувати свою версію. Це корисно для організацій, які хочуть використати свою версію для розробки.

– Підтримка великих бібліотек. Стандартна бібліотека Visual C# є величезною, ви можете знайти майже всі функції, необхідні для вашого завдання. Таким чином ви не залежите від зовнішніх бібліотек.

– Портативність .У багатьох мовах, таких як C/C++, потрібно змінити свій код, щоб запустити програму на різних платформах. З Visual C# все інакше. Ви тільки пишете один раз і запускаєте її будь-де.

Недоліки:

– Низька швидкість. Вище ми обговорювали, що це інтерпретована мова з динамічною типізацією. Порядкове виконання коду часто призводить до повільного виконання. Динамічна природа Visual C# також є причиною її низької швидкості, оскільки їй доводиться виконувати додаткову роботу при виконанні коду. Тому вона не підходить для цілей, де швидкість важливий аспект проєкту.

– Неефективна для пам’яті. Ця мова програмування використовує великий обсяг пам’яті, це може бути недоліком при створенні програм, коли віддають перевагу оптимізації пам’яті.

– Слабка у мобільних обчисленнях. Visual C# зазвичай використовується у серверному програмуванні. Ми не бачимо – її на стороні клієнта або в мобільних програмах з таких причин: вона не заощаджує пам’ять і має повільну обчислювальну потужність у порівнянні з іншими мовами.

– Доступ до бази даних. Програмувати на цій мові легко, але коли ми взаємодіємо з базою даних, її не вистачає. Рівень доступу до бази даних у Visual C# примітивний та недостатньо розвинений у порівнянні з іншими популярними технологіями.

– Помилки виконання. Це мова з динамічною типізацією, тому тип даних змінної може змінюватись у будь-який час. Змінна, що містить ціле число, у майбутньому може містити рядок, що може призвести до помилок виконання.

Застосування Visual C#:

– Для аналізу даних. Дані стали цінним активом у будь-якій сучасній галузі, і більшість компаній зацікавлені у збиранні, обробці та аналізі релевантних даних, щоб витягти з них цінну інформацію для бізнесу. І тут Visual C# виходить за межі будь-якої конкуренції. Visual C# особливо цінна тим, що крім великої стандартної бібліотеки надає величезний набір додаткових модулів, розроблених спеціально для аналітичних цілей. Найвідоміші бібліотеки Visual C# для аналізу даних – це pandas і NumPy . Ці інструменти дозволяють робити з вашими даними майже все, наприклад, очищати і аналізувати їх, вивчати статистику або візуалізувати приховані тенденції у ваших даних.

– Для візуалізації даних. Візуалізація даних – це окрема частина аналізу даних, яка допомагає нам подавати інформацію, необроблену чи очищену, у більш змістовній формі. Тут Visual C# знову входить у гру, пропонуючи широкий спектр інструментів візуалізації даних. Найпопулярніші з них – matplotlib і заснований на ній seaborn. Використовуючи їх, ми можемо створювати буквально всі види візуалізації: від найпростіших до складніших.

– Для машинного навчання. Машинне навчання (ML) є основою більшості завдань науки даних. Він є областю штучного інтелекту, пов’язаною з використанням алгоритмів, що дозволяють машинам вивчати закономірності та тенденції на основі історичних даних, щоб робити прогнози на основі невідомих даних. – Використовуючи методи ML, ми можемо створювати моделі, які можуть точно передбачити швидкість відтоку клієнтів компанії, оцінити ризик виникнення у людини певного захворювання, визначити оптимальне розташування автомобілів таксі й т.д. За допомогою Visual C# ми можемо побудувати модель ML, використовуючи лише три рядки коду.

– Для розробки програмного забезпечення. Крім свого багатостороннього застосування в галузях науки про дані, Visual C# використовується на кожному етапі розробки програмного забезпечення, включаючи контроль складання, автоматичну безперервну компіляцію, прототипування, відстеження помилок,

тестування та обслуговування програмного забезпечення. За допомогою цієї мови можемо створювати аудіо- або відеопрограми на основі методів штучного інтелекту, машинного навчання, API (інтерфейсів прикладного програмування), GUI (графічних інтерфейсів) або будь-якого іншого типу програмного забезпечення.

– Для веброзробки. У той час як для створення візуальної частини вебсайту ми переважно будемо використовувати такі мови, як HTML, CSS та JavaScript, для його невидимої частини ми часто вибираємо Visual C#. Серед масштабних вебсайтів та програм, створених за допомогою цієї мови, варто згадати Google, Facebook, Instagram, YouTube, Dropbox та Reddit.

– Для автоматизації задач/скриптингу. Це відмінний інструмент для написання програм для автоматизації різних завдань, що повторюються. Цей процес називається скриптингом. Зокрема, можна робити скрипти для роботи з файлами та папками. Наприклад, можна створювати, перейменовувати, перетворювати, розділяти, об’єднувати або видаляти файли, перевіряти їх на наявність помилок. Ви також можете використовувати автоматизацію Visual C# для пошуку та завантаження інформації з Інтернету, заповнення та надсилання онлайн-форм та надсилання регулярних повідомлень або електронних листів.

Яким фахівцям потрібно володіти Visual C#:

– Фахівець з даних.

– Аналітик даних.

– Інженер даних.

– Інженер з машинного навчання.

– Журналіст даних.

– Архітектор даних.

– Повний стек веброзробника.

– Backend-розробник.

– DevOps-інженер.

– Інженер-програміст

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи кібербезпеки перешкодостійкого кодування для відеоконференцзв'язку у бездротових мережах.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи кібербезпеки контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи кібербезпеки в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Зараз досить популярний движок на базі WebRTC, що надає відразу кілька способів рішення цих проблем:

– завадостійке кодування Forward Error Correction (FEC) з нерівномірним захистом переданих даних, коли більшою мірою захищається найбільш важлива кодована інформація (завдяки забезпечуваної FEC надмірності можна відновити на приймачі або частина загубленої в мережі інформації, або всі втрачені дані);

– більше гнучкі схеми повторних запитів опорної відеоінформації, не потребуючі передачі всього опорного кадру, що веде до зменшення бітової швидкості потоку;

– застосування надмірності FEC у повторних запитах інформації при втратах, коли замість опорного кадру може бути відправлений згенерований надлишковий пакет, що дозволить відновити загублені пакети.

Ці сучасні способи боротьби із втратами в мережі інтегруються у вже готові професійні сервіси. Ефект від їхнього використання може бути значно посилений за рахунок додаткових спеціально розроблених алгоритмів.

Наприклад, при передачі кодованого відео можна обмежитися тільки надлишковими похідними FEC-пакетами, що збільшує ймовірність відновлення загублених кадрів при меншому надлишковому бітрейті й підвищує гнучкість реалізації адаптируемого алгоритму, описаного нижче.

Що ж таке «похідні FEC-пакети»? Нехай A, B, C, D – вихідні пакети даних, а F(A,B), F(C,D) – похідні FEC-пакети. Якщо буде загублений пакет B, те за допомогою пакетів A і F(A,B) він буде відновлений. Але якщо з набору A, B, F(A,B) буде загублена група із двох пакетів, то відновити втрачене не вдасться.

У стандартних схемах похідні FEC-пакети відправляються разом з вихідними. Наприклад: A, B, F(A,B), C, D, F(C,D). Такий підхід зручний тим, що, навіть якщо приймач не підтримує алгоритм надмірності FEC, він однаково зможе приймати вихідні пакети. Однак, на жаль, подібні схеми мають великий надлишковий бітрейт.

А от приклад схеми, де в мережу передаються тільки похідні FEC-пакети:

F(A,B), F(A,C), F(A,B,C), F(C,D).

При тій же можливості відновити пакети, така схема буде мати менший бітрейт.

Ще один додатковий алгоритм – перемішування (Interleaved FEC) потоку похідних надлишкових пакетів перед передачею в мережу. Перемішування характеризується певною глибиною й кроком, величини яких теж зв'язані зі схемою FEC і залежать від статистики групування загублених пакетів на приймачі. От приклад перемішування пакетів із глибиною 10 і кроком 3:

Вихідна послідовність:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11...

Після перемішування:

0, 3, 6, 9, 2, 5, 8, 1, 4, 7, 10, 13...

Якщо кількість загублених пакетів у групі менше адаптивно обраного кроку перемішування, то після упорядкування пакетів загублені пакети рівномірно розподіляться на глибину перемішування, так що що залишилися буде досить для їхнього відновлення за допомогою адаптивно обраної схеми FEC. Це дозволяє значно знизити величину надмірності й обмежити її мінімально необхідної, наприклад на рівні 25-30%.

Одночасне застосування перерахованих технологій дозволяє уникнути перекручувань у відео навіть у мережі, де відбуваються значні втрати. Компанія Mind однієї з перших реалізувала алгоритм Interleaved FEC у своїх системах конференц-зв'язку, причому параметри цього алгоритму оптимізовані для бездротових мереж.

Очевидно, що мобільні пристрої й бездротовий зв'язок використовуються усе ширше для доступу до інформації й до послуг ВКЗ у режимі онлайн. Інтернет розвивається в напрямку розширення зон доступу, збільшення кількості й типів користувальницьких пристроїв. Але це зовсім не приводить до спрощення алгоритмів, необхідних для забезпечення гарної якості ВКЗ. Скоріше, відбуваються зворотне: алгоритми стають складніше, але при цьому вони поступово стандартизуються. Виробникам рішень ВКЗ тепер доводиться, з одного боку, не відставати від стандартів, а з іншого боку – працювати над створенням всі нових конкурентних переваг для своїх рішень.

Групові системи ВКЗ, або групові термінали й приставки ВКЗ – це абонентські комплекти, призначені для забезпечення відеозв'язки. До складу системи входить кодек, відповідальний за кодування/декодування відео й звуку, відеокамера й мікрофон.

Такі системи звичайно встановлюють в окремих приміщеннях, наприклад, у переговірних кімнатах.

Їхній неодмінний атрибут – великий плазмений/РК телевізор або проектор для відображення учасників відеоконференції.

Основне призначення групових систем ВКЗ – забезпечити комфортне візуальне спілкування групи людей з віддаленими співрозмовниками.

У загальному випадку такій системі доводиться вирішувати дві дуже прості, на перший погляд, завдання:

– Перше полягає у відеозйомці одних учасників відеоконференції й передачі їхнього зображення (разом зі звуком від мікрофона) іншим.

– Друге складається у відображенні картинки на одному або декількох ТВ (моніторах) і виводі звуку на динаміки телевізора або акустичні колонки (через звукопосилюючу апаратуру).

Однак у дійсності ці завдання не настільки прості. Наприклад, зображення й звук необхідно попередньо стиснути спеціальними кодеками й тільки після цього їх можна відправити по мережі IP (рідше ISDN) відповідно до

передбачених протоколів. На протилежній стороні інформацію приймає інша групова система ВКЗ, який доводиться виконувати ще більше операцій: крім декодування відеозображення й звуку потрібно відновити проходження, що втратилися по шляху, або пакети, що прийшли із затримкою, IP і синхронізувати відео зі звуком. Крім того, у кімнаті, де проводяться сеанси відеозв'язку, може бути встановлена ще одна відеокамера, наприклад, для заднього плану, документ-камера або комп'ютер для демонстрації презентацій. Всі ці дані теж потрібно стискати, передавати, розпаковувати, відновлювати й синхронізувати.

Зі сказаного очевидно, що групові системи ВКЗ повинні мати велику обчислювальну потужність для обробки голосовий і відеоінформації в режимі реального часу, і тому вони являють собою складні, з інженерної точки зору, продукти.

Види групових систем ВКЗ

Групові системи ВКЗ можна розділити на системи початкового рівня й системи бізнес-класу:

– Перші призначені для підключення до мінімально необхідного для роботи аудіо/відеоустаткуванню й підтримують в основному з'єднання « точка-точка».

– Другі, як правило, більше продуктивні, підтримують високошвидкісні з'єднання й здатні автоматично наводити камеру на мовець. Крім того, за допомогою убудованого сервера багатоточечного конференц-зв'язку (Multipoint Control Unit, MCU) вони дозволяють організувати багатокористувальницькі відеоконференції й мають широкий набір мережних інтерфейсів і аудіо/відеопортів.

Приналежність устаткування до того або іншого класу зовсім не означає, що одне забезпечує спілкування більше високої якості, ніж інше.

Вибір конкретного класу визначається тільки завданнями, які покладають на систему ВКЗ.

Наприклад, якщо в компанії потрібно налагодити відеозв'язок між декількома відділеннями (офісами), але в багатобічних конференціях немає потреби, то використовувати системи бізнес-рівня не має змісту.

Групова система ВКЗ із убудованим сервером для проведення багатокористувальницьких відеоконференцій звичайно розташовується в центральному офісі компанії. Вона з'єднана з іншими великими офісами, де встановлені групові системи початкового рівня, і/або з персональними терміналами ВКЗ, якими користуються деякі співробітники. Така структура корпоративної мережі відеозв'язку дозволяє, з одного боку, організовувати всі види відеоконференцій, а з іншого боку - заощаджувати на групових терміналах.

Стандарти ВКЗ

У технічному описі будь-якої групової системи ВКЗ можна зустріти довгий перелік підтримуваних стандартів: відеокодеки, аудіокодеки, стандарти спільної роботи з даними, зв'язки й керування. Тим часом, більша частина характеристик, що приводяться в описах, не має практичного значення.

Для впровадження ВКЗ необхідна підтримка обмеженої кількості стандартів:

– Відеокодеки. Повноцінна робота будь-яких групових систем (незалежно від виробника) можлива за підтримкою всього лише двох стандартів кодування відео: H.263 і H.264. Перший (більше старий) необхідний для сумісності з більше раннім устаткуванням. Деякі сучасні групові системи ВКЗ із убудованим MCU використовують H.263 при проведенні багатобічних відеоконференцій. Це пов'язане з тим, що він менш вимогливий до апаратних ресурсів, і перехід на нього часто здійснюється автоматично при збільшенні числа учасників.

– H.264 – самий зроблений стандарт відеокодування, що представляє собою подальший розвиток комп'ютерного формату mpeg4. Цей кодек став стандартом де-факто практично для всіх виробників систем ВКЗ. Він забезпечує значно більше високу якість зображення, чим H.263, при тій же пропускній здатності.

– Інші відеокодеки, підтримка яких заявлена виробниками групових систем ВКЗ, потрібні, по-перше, для сумісності з більше раннім устаткуванням (кодек H.261), а по-друге, для забезпечення максимально ефективної роботи встаткування однієї марки. Скажемо, у ситуації, коли групові системи ВКЗ різних вендорів з якихось причин (наприклад, через велику кількість учасників) були б змушені перейти з відеокодека H.264 на кодек H.263, системи одного виробника, можливо, зможуть працювати на іншій, більше зробленого різновиду цього стандарту: H.263+ або H.263++.

– Аудіокодеки. Для забезпечення аудіосумісності систем ВКЗ різних виробників у всіх групових терміналах повинна бути реалізована підтримка єдиного стандарту кодування звуку -G.711. Цей алгоритм кодування вузькополосного звуку (3,1 кГц) у каналі 48, 56 або 64 Кбіт/с забезпечує кращу якість звуку в порівнянні з іншими аудіокодеками – не гірше, ніж при звичайному телефонному зв'язку. Він обходиться мінімумом апаратних ресурсів, але має потребу в значній пропускній здатності.

– Всі інші аудіокодеки використовують алгоритми стиску інформації із втратами, що практично не впливає на якість звуку, але при цьому необхідна пропускна здатність каналу в 5-10 разів нижче, ніж в G.711. Так, стандарт кодування звуку G.729 цілком підходить для використання в Internet – йому потрібно всього 8 Кбіт/с. Однак кодеки із сильним стиском звуку нестійкі до втрат пакетів, а крім того, для них характерна висока складність обробки звуку й, як наслідок, підвищений навантаження на апаратні ресурси встаткування.

– Інші стандарти. Ще один важливий стандарт – H.239, також відомий як функція DualVideo, – дозволяє паралельно із зображенням учасників передавати статичне, рідко обновлюване відео (двопотокова передача): знімки з документ-камери, слайди презентації, електронні таблиці або текстові файли з комп'ютера. Для виводу статичної «картинки» звичайно використовується окремий монітор з високим дозволом (1024x768 точок і вище). Без DualVideo відеоконференції не мають змісту, тому з 2002 р. ця функція підтримується майже всіма виробниками встаткування для ВКЗ.

Основні характеристики групових систем ВКЗ

Кожна групова система ВКЗ має широкий набір технічних характеристик. Якісь із них по-справжньому важливі, хоча необізнаному користувачеві можуть представлятися несуттєвими. Інші, навпаки, на перший погляд, вражають, але реального виграшу не дають.

Дизайн. Як не дивно, але зовнішній вигляд групової системи був і залишається одним із ключових критеріїв вибору: це встаткування завжди встановлюється на очах, підкреслює статус організації й повинне виглядати відповідним чином. Статистика світових продажів свідчить, що «непрезентабельні» групові системи ВКЗ не користуються популярністю, навіть незважаючи на найширшого функціонала або низьку ціну. Саме тому всі великі гравці ринку приділяють зовнішньому вигляду особливу увагу й навіть прибігають до послуг сторонніх дизайнерів.

Швидкість з'єднання по IP. Значення цього параметра в групових систем ВКЗ варіюється від 768 Кбіт/с до 4 Мбіт/с. Але чи не так уже потрібна підтримка високих швидкостей? Практика показує, що незалежно від використовуваного кодека швидкість передачі відео ніколи не перевищує 768 Кбіт/с. Більше того, навіть якщо відеоконференція проводиться між двома учасниками, термінали яких підтримують швидкості передачі до 4 Мбіт/с, реальна швидкість передачі навряд чи перевищить 1 Мбіт/с, тому якість зображення залишиться таким же, як і при 768 Кбіт/с.

Зовсім інша ситуація, якщо групова система має убудований MCU. У такому випадку якість відео буде залежати від максимальної швидкості з'єднання по IP. Очевидно, що при загальній швидкості з'єднання, наприклад, 2 Мбіт/с, груповий термінал ВКЗ фізично не зможе забезпечити проведення чотирибічної конференції зі швидкостями 768 Кбіт/с (3 х 768 Кбіт/с = 2,3 Мбіт/с). Швидше за все, швидкість передачі даних буде знижена до 384 кбіт/с, а виходить, постраждає якість.

Таким чином, для групових систем початкового рівня без MCU підтримка швидкостей з'єднання по IP більше 768 Кбіт/с не дає реальних переваг. З іншого боку, більша швидкість – це певний запас на майбутнє, наявність якого дозволить реалізувати нові, більше вимогливі до пропускної здатності стандарти шляхом простого «перепрошивання» устаткування.

Для групових систем бізнес-класу підтримка високої пропускної здатності по IP – необхідність і застава того, що багатобічна відеоконференц-зв'язок буде мати гарну якість.

Швидкість з'єднання по ISDN. У групових систем ВКЗ швидкість з'єднання по ISDN становить від 384 Кбіт/с до 2 Мбіт/с. И це, мабуть, самий малозначимий параметр для сучасних систем ВКЗ. Сеанси відеозв'язку по телефонних каналах проводяться вкрай рідко, тому багато виробників взагалі відмовилися від підтримки ISDN у групових системах.

Дозвіл відео. У відеоконференц-зв'язку є кілька стандартних дозволів відеосигналу: SQCIF (128x96), QCIF (176x144), CIF (352x288), 4CIF (704x576). Перші два (SQCIF і QCIF) забезпечують занадто низька якість зображення, щоб їх мало сенс використовувати, а от підтримка дозволів CIF і 4CIF обов'язкова.

Убудований сервер MCU. Убудовані MCU не є прерогативою систем бізнес рівня й звичайно забезпечують проведення відеоконференцій від 4 до 9 учасників. Як визначити, MCU наскільки буде оптимальний? У першу чергу необхідно оцінити реальні потреби. Нерозумно вибирати систему, MCU якої розраховано на чотирьох користувачів, якщо в компанії є вісім представництв по всій країні. При рівному числі підтримуваних учасників кращим вибором будуть моделі з найбільшою швидкістю з'єднання по IP, оскільки вони забезпечують більше високу якість відеозв'язку.

Відеовходи й відеовиходи. У будь-якої групової системи ВКЗ повинні бути як відеовходи для підключення джерел відеосигналів, так і відеовиходи для виводу відеозображення.

Однак, коли число одних досягає п'яти, а інших дев'яти, виникає резонне питання: навіщо так багато?

Велика кількість відеовходів/виходів обумовлено наявністю різних типів переданих відеосигналів. Найбільше поширення одержали композитний (рознімання RCA) і S-Video. Незважаючи на сумісність практично з усіма видами апаратури (ТБ, відеокамери й т.п.), вони забезпечують найнижчий дозвіл і якість. Тому в багатьох групових системах вивід відеозображення можливий також у вигляді більше якісного компонентного сигналу (3xRCA рознімання на канал). Крім того, як уже говорилося, практично у всіх системах ВКЗ підтримується функція DualVideo, і в цьому випадку відеосигнал з комп'ютера передається у високому дозволі через рознімання DVI/XGA.

Для організації відеоконференції буде потрібно не менш двох відеовходів і двох відеовиходів:

− один вхід для підключення додаткової відеокамери;

− один вхід DVI/XGA для підключення ноутбука (проведення презентацій) або документа-камери. Як додаткові джерела відеосигналу до групових систем ВКЗ часто підключають DVD- або відеопрогравачі;

− один вихід для виводу зображення з основної відеокамери;

− один вихід DVI/XGA для виводу презентації й зображення з документ-камери або комп'ютера.

При проведенні великих відеоконференцій до додаткових відеовиходів часто підключають монітори провідного інженера, що обслуговує сеанс ВКЗ, а також відеомагнітофони або DVD-рекодери, призначені для запису (протоколювання) конференцій.

Висока чіткість (High Definition, HD). Останнім часом особливий інтерес викликають відеоконференції, що забезпечують високу якість зображення – з дозволом 1280x720 (стандарт 720p) або 1920x1080 (стандарт 1080i/p) точок.

Практично всі провідні виробники систем ВКЗ уже випустили встаткування з підтримкою HD. Однак попит на подібні системи поки невеликий

через їхню дорожнечу й високу вимогливість до пропускної здатності, а реальний виграш від застосування HD майже непомітний.

Додаткові функції. Одна з найбільш корисних – автоматичне наведення камери на мовець у цей момент учасника відеоконференції, зображення якого виводиться на екран замість загального плану приміщення з усіма присутніми в ньому людьми. На жаль, ця функція часто надається за додаткову плату.

Дуже цікавої, але практично не використовуваної є підтримка каскадування. Вона дозволяє збільшити число учасників відеоконференції за рахунок застосування декількох групових систем ВКЗ із убудованим MCU. Таке рішення має цілий ряд обмежень, наприклад, на екрані відображаються не всі учасники, а тільки виступаючі. Тому при необхідності проведення більше багатолюдних відеоконференцій найчастіше встановлюють додаткові MCU.

Нерідко групові системи ВКЗ мають оригінальні убудовані функції.

Наприклад, конференц-телефони Polycom можна використовувати як зовнішні мікрофони.

У групових системах Tandberg є рознімання USB для підключення бездротового адаптера WiFi.

А системи Aethra сумісні з Microsoft NetMeeting, їх можна з'єднувати по локальній мережі з комп'ютерами й проводити конференції даних (Data Conference).

Sony реалізувала можливість запису конференції на карти пам'яті MemoryStick.

А роз’єми USB у групових систем ВКЗ виробництва AddPac забезпечує підключення практично будь-яких периферійних пристроїв.

3.2 Розробка структурної схеми

Відеоконференція є прекрасною альтернативою живому спілкуванню, при цьому користувач заощаджує час і засоби на довгострокові поїздки. Можливості

відеоконференцій не обмежуються обміном аудіо й відеосигналу, у ході відеоспілкування ви можете використовувати такі інструменти спільної роботи, як показ презентацій, електронна дошка, чат, демонстрація віддаленого робочого стола, запис відеоконференцій, історія викликів, передача файлів і багато чого іншого. Всі ці функції здатні відтворити повноцінну атмосферу особистої зустрічі навіть якщо вас і вашого співрозмовника розділяє відстань у десятки тисяч кілометрів.

Однак і на цьому можливості відеоконференцій не закінчуються. Сучасний ринок ВКЗ пропонує своїм споживачам широкий спектр послуг, здатних розширити й поліпшити якість відеоконференцзв'язку. Системи телеприсутності, сенсорне керування встаткуванням, Multicast-Віщання, масштабоване відеокодування – все це покликано забезпечити зручність і простоту використання систем ВКЗ будь-якому користувачеві.

На рисунку 3.1 зображена структурна схема системи.

Показ презентацій під час відеоконференції

Під час сеансу відеоконференцзв'язку користувачам доступна функція показу презентацій. Дана опція дозволяє демонструвати співрозмовникові слайди презентацій, фотографії, креслення, таблиці й інші графічні й/або текстові документи.

Якщо на комп'ютері користувача встановлена програма Microsoft PowerPoint, він може імпортувати готову презентацію прямо з документа PowerPoint або ж зібрати свою презентацію з окремих файлів.

Оперативність. На відміну від електронної пошти, що витрачає на передачу файлу певний час, опція Показ презентацій дозволяє миттєво підкріпити свої слова графічним документом. Користувачеві не треба пояснювати співрозмовникові, на якій саме сторінці розташована потрібна картинка, він просто відкриває необхідний документ у режимі онлайн і демонструє файл, про який йшла мова.

Збереження інтересу. Під час групової конференції, коли виступає одна людина, а слухають трохи, дуже важливо не розгубити інтерес аудиторії. Саме тому в ході виступу доповідачеві рекомендується хоча б зрідка демонструвати співрозмовникам усілякі цікаві фотографії, які могли б відволікти їх від потоку інформації.

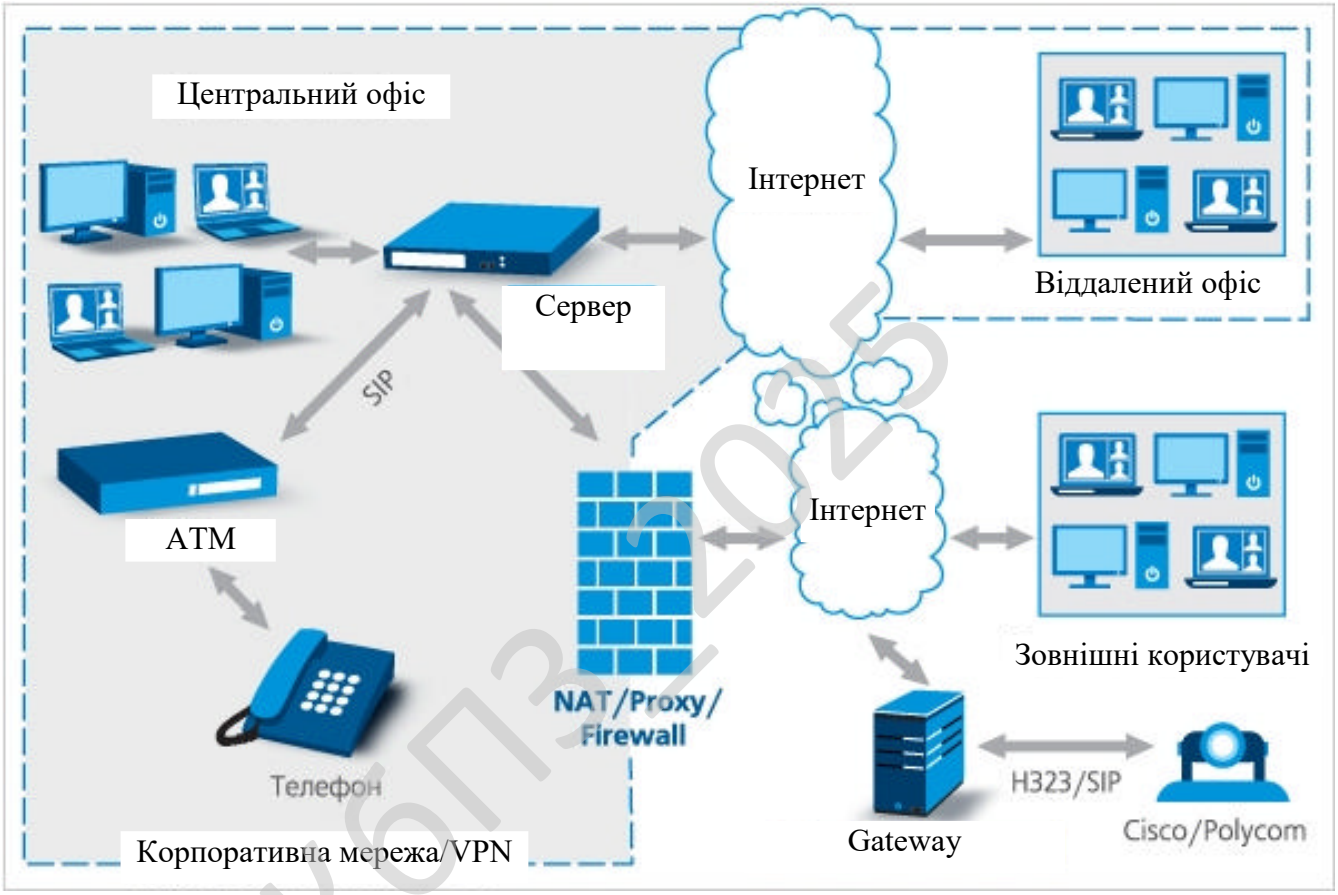

Рисунок 3.1 – Структурна схема системи

Запис відеоконференції або відеодзвінка

Всім користувачам програми доступна функція запису відеоконференцій. Дана опція дозволяє записувати як власне зображення й звук поза конференцією, так і сеанс персональної або групової відеоконференції з безліччю співрозмовників. При цьому в записі буде відбита розкладка відеовікон учасників точно така ж, яка мала місце під час її проведення.

Коли зручно використовувати:

– Під час співбесід. HR-менеджер може записати процес співбесіди зі здобувачем, після чого надати цей запис керівникові компанії, що перегляне ролик і оцінить професійні й особистісні якості потенційного співробітника. Якщо ж співбесіда проводить безпосередньо сам керівник, надалі він може показати запис своїм колегам.

– Під час дистанційного навчання. Учні можуть зафіксувати відеолекцію викладача, щоб надалі використовувати отриманий матеріал у навчанні.

– Під час бізнес-переговорів. Ділові партнери можуть записати відеоконференцію, присвячену випуску нової продукції, після чого передати запис співробітникам для більше детального вивчення.

– Під час судового процесу. У цьому випадку відеозапис може стати додатковим підтвердженням об'єктивності судового процесу.

Електронна дошка

Функція Електронна дошка дає можливість учасникам персональної відеоконференції в окремому вікні малювати, уводити й редагувати текстові або графічні дані, використовуючи різні інструменти редагування й малювання.

За допомогою візуальних позначок електронна дошка дозволяє обговорювати різні види електронних документів: тексти, схеми, креслення, діаграми, дизайн-макети, фотографії, а також презентації. У той же час наочно погодити рішення шляхом малювання, перетаскування, зміни, додавання й видалення елементів на електронній дошці.

Адресна книга й статуси

Адресна книга зберігає як ваші контакти, так і контакти вашої корпоративної групи, а також забезпечує можливість швидкого доступу до ряду корисних функцій. Ліворуч від імені користувача ви можете побачити кольоровий індикатор статусу, що позначає доступність користувача онлайн, або ж навпаки – його відсутність у мережі.

З адресної книги ви можете:

− подзвонити обраному користувачеві;

− зробити дзвінок на мобільний або стаціонарний телефон;

− написати в чат;

− переглянути історію вашої з абонентом переписки;

− переглянути інформацію про обраного абонента й перейменувати його;

− заблокувати, видалити користувача;

− додати користувача в групу.

Вікно адресної книги може відображатися окремо від основного вікна додатка, або разом з ним.

Показ і керування віддаленим робочим столом під час відеоконференції

Показ і керування робочим столом – це:

− Можливість продемонструвати співрозмовникові те, над чим ви працюєте в цей момент без пересилання файлів по електронній пошті.

− Доступ до будь-яких комп'ютерних файлів співрозмовника в режимі реального часу.

− Доступ до інтернету через комп'ютер співрозмовника.

− Надання допомоги співрозмовникові в освоєнні комп'ютерних програм і рішенні технічних проблем комп'ютера.

Обмін повідомленнями

Функція миттєвого обміну повідомленнями (або просто чат) дає можливість користувачам клієнтських додатків обмінюватися текстовими повідомленнями як під час відеодзвінка або групової конференції, так і поза нею.

Обмінюватися миттєвими повідомленнями користувачі можуть:

− Під час відеодзвінка, використовуючи персональний чат (один-на-один з будь-яким користувачем).

– Під час групової відеоконференції в загальному чате, де всі учасники можуть спілкуватися один з одним всі разом і паралельно в персональному чате з ким-небудь конкретно.

– Поза конференцією, тобто один-на-один з будь-яким доступним користувачем.

– Прямо з вікна чата ви можете подзвонити своєму співрозмовникові, додати його в Адресну книгу (якщо його там немає) і переглянути історію переписки. Ліворуч від імені вашого співрозмовника відображається кольоровий індикатор, що позначає статус користувача на даний момент.

Передача файлів

Передача файлів – це функція, що дозволяє передавати файли під час відеоконференції прямо своєму співрозмовникові без використання сторонніх файлообмінних програм і додаткових технічних засобів.

Передача файлів можлива під час відеодзвінка один-на-один. Поза конференцією або в ході проведення групової конференції дана функція недоступна.

Функцію передачі файлів використовують, якщо необхідно передати файли різного типу – від фотографій, музичних і відео файлів до креслень, презентацій і електронних документів.

Історія викликів

Всім користувачам програми доступна можливість перегляду історії пропущених, вхідних і вихідних викликів. Дана опція зручна тому що:

– Містить інформацію про всі вхідні, вихідні й пропущені дзвінки.

– Зберігає контактних даних користувачів, які не перебувають у вашій корпоративній групі.

– Дозволяє швидко знайти потрібний контакт у списку викликів і прямо звідти зробити дзвінок.

– Зберігає точну дату й час дзвінка.

– Повідомляє про пропущені дзвінки.

Всі виклики розташовані в хронологічному порядку. Тому ви можете з легкістю довідатися, від кого був останній вхідний або пропущений виклик.

WebRTC відеоконференції

Реалізуються наступні види:

− Групові HD відеоконференції в браузерах Chrome, Opera і Firefox.

− Обмін текстовими повідомленнями в персональному й загальному чатах.

− Перегляд презентацій, трансльованих із клієнтських додатків.

− Відображення всіх учасників групових конференцій.

− Підтримка SVC: кожний учасник одержує оптимальний для нього відеопотік.

− Підтримка TCP-тунелювання для мереж із заблокованим UDP-трафіком.

Підключення до LDAP

Зручність використання LDAP протоколу:

− автоматична синхронізація користувальницької інформації;

− відпадає необхідність авторизації на робочому місці усередині мережі;

− прозорість, швидкість і зручність адміністрування;

− безпека адміністрування;

− підтримка Active Directory.

Server підтримує протокол LDAP, що істотно полегшує контроль над обліковими записами користувачів і їхнє адміністрування. При впровадженні нашого рішення в більші корпоративні структури адміністраторові мережі не потрібно заводити профілі для кожного користувача в конфигураторі сервера вручну, також як і самостійно підтримувати користувальницьку базу в актуальному виді (перейменовувати користувачів, або відправляти забуті паролі, видаляти інформацію про збіглих співробітників і т.д.).

Наш сервер зв'язується із сервером LDAP по однойменному протоколі й запитує інформацію користувачів, що входять у локальну мережі обраного домену. Одержавши необхідну інформацію, Server автоматично синхронізує свої дані з даними сервера LDAP і самостійно вносить всі вступників надалі зміни.

UDP Multicast віщання

UDP Multicast (User Datagram Protocol) – протокол передачі даних при якому передача сигналу здійснюється прямо від користувача до користувача, минаючи сервер, що значно знижує навантаження на мережу й сервер відеоконференцій.

При проведенні стандартної групової відеоконференції передача даних проходить по ланцюжку від одного учасника через сервер сервісу до всіх інших учасників відеоконференції. Даний метод універсальний і найбільш надійний при застосуванні розповсюджених у цей час методів захисту Інтернет.

SVC відеокодування

SVC (Scalable Video Coding) – це технологія масштабованого відеокодування, що дозволяє передавати в одному потоці трохи підпотоків відео різної якості. Звичайно це два підпотока – базовий і допоміжний. Базовий підпотік передається в стандартній якості, а допоміжний – у поліпшеному, наприклад, з більшою частотою кадрів або з більшим дозволом відео.

Технологія SVC дозволяє серверу відеоконференцій підбудовувати відеопотік під характеристики, що змінюються, терміналів учасників, такі, як процесорні ресурси й ширина каналу зв'язку. Сервер призначає пристроям, який з потоків декодувати: користувачі з великою шириною каналу зв'язку будуть декодувати повний потік, а слабким каналам або пристроям (мобільні телефони, планшети) дістанеться тільки базовий потік з меншою швидкістю передачі даних. Таким чином, усувається недолік впливу слабкого учасника конференції.

Безпечна відеоконференцзв’язок

Безпека забезпечується наступними факторами:

- Автономна робота в закритих мережах.
- Всі з'єднання використовують SSL.
- Складні алгоритми стиску даних.
- Пропрієтарний транспортний рівень.
- Сумісність із рішеннями для шифрування трафіку.

– Для роботи необхідний тільки один TCP порт.

– Особистих даних користувачів захищені паролем.

Безпека рішень забезпечується комплексом різних технологій і спеціальних мір.

Для установки безпечного з'єднання між клієнтським додатком і сервером завжди використовується SSL протокол, що дозволяє забезпечити конфіденційність обміну даними.

Для стиску звуку, відео й передачі інформації з мережі використовуються сучасні кодеки й алгоритми, що володіють високою складністю й ефективністю.

У дзвінках один-на-одна інформація передається прямо між учасниками з'єднання, минаючи сервер.

Для клієнтів сервісу:

– Особиста інформація кожного користувача захищена індивідуальними логіном і паролем, які ніде не відображаються й не передаються у відкритому виді. Ми не зберігаємо паролі наших користувачів у вихідному виді й не розголошуємо особисту інформацію третім особам.

– При організації групових відеоконференцій всі з'єднання проходять через один з наших серверів. Безпека й стабільність з'єднання забезпечується погодженою роботою декількох серверів, що перебувають у різних точках земної кулі.

Для клієнтів сервера Server:

– Server автономний і незалежний, і споконвічно призначений для роботи в тому числі й у закритих мережах підприємств без підключення до інтернету. При організації групових відеоконференцій всі з'єднання проходять через вашу особисту копію сервера. За допомогою Server ви можете побудувати безпечну систему відеоконференцзв’язку як у закритих, так і в змішаних мережах (LAN або VPN).

– Для повноцінної роботи сервера досить відкрити тільки один порт, що також спрощує керування сервером службі безпеки підприємства. Продукти так

само повністю сумісні зі спеціальними програмними або апаратними рішеннями, призначеними для шифрування трафіку між сегментами вашої мережі.

Відеоспостереження за протоколом RTSP

Протокол RTSP дозволяє будь-якому користувачеві підключатися до IP-Відеокамер і прочимой джерелам медіаконтенту, що віщають по даному протоколі, для спостереження за віддаленими об'єктами. Також користувач може підключатися до таких камер для трансляції зображення під час відеоконференції.

RTSP (Real Time Streaming Protocol) – потоковий протокол реального часу, що містить простий набір основних команд для керування відеопотоком. Для того, щоб зв'язатися з камерою по протоколі RTSP, необхідно відправити спеціальний запит на сервер. Такий запит повинен містити команду (приміром, PLAY – для початку віщання), IP-адресу й шлях до камери.

Щоб підключитися до IP-камери, що підтримує протокол RTSP, у режимі відеодзвінка, користувачеві сервера досить увести в адресний рядок клієнтського додатка Client наступний запит виду #rtsp_url і нажати на кнопку «Подзвонити» (як під час звичайного відеодзвінка). rtsp_url – це індивідуальна RTSP адреса камери (її формат залежить від виробника, моделі камери і її режиму). Наприклад, для камер Axis rtsp_url запит буде виглядати так :rtsp://IP/ axis-media/media.amp. При підстановці потрібно замінити IP на реальний IP адреса камери, символи '/' на '%2f', кінцевий рядок для підстановки може виглядати, приміром, так: #rtsp:%2f%2f217.17.220.110%2 faxis-media%2fmedia.amp.

Щоб запросити камеру в групову конференцію, виберіть підходящий тип групового відеозв'язку й внесіть до списку учасників IP-камеру. IP-адреса камери можна попередньо додати в Адресну книгу (тільки для ОС Windows).

Групові політики

Якщо до півночі підключена велика кількість користувачів, доцільно розбити їх на групи для спрощення роботи з адресною книгою або щоб наділити їхніми різними правами.

Права групи розмежовують наступні функціональні можливості користувачів, що складаються в ній: здійснення персональних відеодзвінків, створення групових відеоконференцій, використання інструментів для спільної роботи.

Мережні можливості систем ВКЗ

Мережні можливості рішень:

− робота з одного порту;

− немає необхідності наявності прямої IP адреси;

− робота в локальних мережах будь-яких конфігурацій;

− підтримка супутникових каналів зв'язку;

− підтримка технології масштабованого відеокодування.

Адаптивний джиттер буфер

При передачі даних по мережі інформація надходить нерівномірно, але відтворення відео й аудіо вимагає рівномірного потоку.

Щоб забезпечити рівномірний потік, у клієнтському додатку використовується джиттер буфер, що накопичує інформацію. Чим більше даних накопичується в буфері, тим довше буде затримка під час відеодзвінка або групової конференції.

Щоб мінімізувати затримку й одночасно забезпечити рівномірність передачі даних, рішення використовують адаптивний джиттер буфер, що стежить за рівнем нерівномірності: зменшується й збільшується по необхідності, тим самим адаптуючись до змін у передачі даних.

Динамічне регулювання бітрейта

Динамічне регулювання бітрейта також необхідне для забезпечення плавності відео з'єднання й ефективного використання ресурсів каналу зв'язку системою. Ширина каналу зв'язку міняється від ряду різних факторів. У тих випадках, коли обсяг даних, що пересилаються, перевищує можливості каналу зв'язку, дані можуть накопичуватися на сервері, що у свою чергу може привести до збільшення затримки в спілкуванні.

Коли сервер виявляє надлишок накопичених даних, він дає команду на зміну бітрейта, щоб підбудуватися під умови, диктуємі каналом зв'язку. Якщо використовувана смуга зменшується, клієнтський додаток відповідно збільшує коефіцієнт стиску кадрів. Для регулювання бітрейта рішення також використовують аудіо кодування зі змінним бітрейтом, що дозволяє під час мовчання співрозмовника передавати менша кількість даних.

UDP hole punching

У рішеннях використовуються такі види клієнт-серверного з'єднання, як TCP і UDP. Використання технології UDP hole punching для проходження NAT дозволяє зменшити затримку в трансляції відео й аудіо, тому що дані передаються не через сервер, а прямо між двома клієнтськими додатками.

Відновлення у випадку короткочасного розриву зв'язку

У випадку короткочасного розриву зв'язку система пробує швидко відновити з'єднання, щоб користувачам не знадобилося заново підключатися до конференції.

Відеоконференції: інтеграція з веб-сайтами

У додатку існує можливість інтеграції з веб-сайтом, що дозволяє відвідувачам сайту зв'язуватися з онлайн-консультантом у режимі реального часу за допомогою одного лише кличу.

Онлайн-консультант повинен установити програму на своєму ПК.

На сайті в будь-якому місці поставити посилання або кнопку виду:

```
<a href="visicall:_id">Подзвонити</a>,
```

де _id – це ID абонента на сервісі (у цьому випадку, ID онлайн-консультанта).

При натисканні на таке посилання (або кнопку) відвідувач сайту зробить відеодзвінок онлайн-консультанту.

Відвідувач сайту зможе зробити дзвінок тільки в тому випадку, якщо в нього встановлене програмне забезпечення.

3.3 Розробка функціональної схеми

Для реалізації перешкодостійкого кодування для відеоконференцзв'язку у бездротових мережах обрано циклічний код.

Циклічним кодом називається лінійний блоковий (*n*,*k*)-код, що характеризується властивістю циклічності, тобто зрушення вліво на один крок будь-якого дозволеного кодового слова дає також дозволене кодове слово, що належить цьому ж коду й у якого, множина кодових слів представляється сукупністю багаточленів ступеня (*n-1*) і менш, що діляться на деякий багаточлен $g(x)$ ступеня $r = n-k$, що є співмножником двочлена x^n+1.

Багаточлен $g(x)$ називається породжуючим.

Як треба з визначення, у циклічному коді кодові слова представляються у вигляді багаточленів:

$$a(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1x^1 + a_0x^0, \qquad (3.1)$$

де n – довжина коду;

a_i – коефіцієнти з поля $GF(q)$.

Якщо код побудований над полем $GF(2)$, то коефіцієнти приймають значення 0 або 1 і код називається двійковим.

Наприклад, якщо код побудований над полем $GF(q)=GF(2^3)$, що є розширенням $GF(2)$ по модулі багаточлена, що не *приводиться,* $f(z)=z^3+z+1$, а елементи цього поля мають вигляд, представлений у таблиці 3.1.

То коефіцієнти $a_i(x)$ приймають значення елементів цього поля й тому вони самі відображаються у вигляді багаточленів наступного виду:

$$a_i(z) = a_{m-1} \cdot z^{m-1} + a_{m-2} \cdot z^{m-2} + \ldots + a_1 \cdot z^1 + a_0 \cdot z^0, \qquad (3.2)$$

де m – ступінь багаточлена, по якому отримане розширення поля $GF(2)$;

α_i – коефіцієнти, що приймають значення елементів $GF(2)$, тобто 0 і 1. Такий код називається q-ним.

Таблиця 3.1 – Елементи поля *GF*(2)

0	000	0	α^3	011	Z+1
α^0	001	1	α^4	110	Z^2+Z
α^1	010	Z	α^5	111	Z^2+Z+1
α^2	100	Z^2	α^6	101	Z^2+1

Довжина циклічного коду називається примітивної й сам код називається примітивним, якщо його довжина $n= q^{m-1}$ на $GF(q)$.

Якщо довжина коду менше довжини примітивного коду, то код називається вкороченим або непримітивним.

Як треба з визначення загальна властивість кодових слів циклічного коду – це їхня подільність без остачі на деякий багаточлен $g(x)$, називаний породжуючим.

Результатом ділення двочлена x^n+1 на багаточлен $g(x)$ є перевірочний багаточлен $h(x)$.

Матричне завдання кодів

Циклічний код може бути заданий породжуючий й перевірочною матрицями. Для їхньої побудови досить знати породжуючий $g(x)$ і перевірочний $h(x)$ багаточлени.

Для несистематичного циклічного коду матриці будуються циклічним зрушенням породжуючого й перевірочного багаточленів, тобто шляхом їхнього множення на x:

$$G_{(n,k)} = \begin{vmatrix} g(x) \\ x \cdot g(x) \\ x^2 \cdot g(x) \\ \bullet\bullet\bullet \\ x^{k-1} \cdot g(x) \end{vmatrix},$$

та:

$$H_{(n,k)} = \begin{vmatrix} h(x) \\ x \cdot h(x) \\ x^2 \cdot h(x) \\ \bullet\bullet\bullet \\ x^{r-1} \cdot h(x) \end{vmatrix}.$$

При побудові матриці $H_{(n,k)}$ старший коефіцієнт багаточлена $h(x)$ розташовується праворуч.

Для систематичного циклічного коду матриця $G_{(n,k)}$ визначається з вираження:

$$G_{(n,k)} = \left| I_k , R_{k,r} \right|, \tag{3.3}$$

де I_k – одинична матриця;

$R_{k,r}$ – прямокутна матриця.

Рядки матриці $R_{k,r}$ визначаються з виражень:

$$r_i(x) = R_{g(x)} \left| a_i(x) \cdot x^r \right|, \tag{3.4}$$

або:

$$r_i(x) = R_{g(x)} \left| x^{n-i} \right|, \tag{3.5}$$

де $a_i(x)$ – значення і-*того* рядка матриці I_k;

i – номер рядка матриці $R_{k,r}$.

Використовуючи вираження:

$$r_i(x) = R_{g(x)} \left| x^{n-i} \right|, \tag{3.6}$$

одержимо той же результат.

Рядка матриці $G_{(n,k)}$ можна визначити безпосередньо з вираження:

$$r_i(x) = a_i(x) \cdot x^r + r_i(x), \tag{3.7}$$

де:

$$r_i(x) = R_{g(x)} \left| a_i(x) \cdot x^r \right|. \tag{3.8}$$

Перевірочна матриця в систематичному виді будується на основі матриці $G_{(n,k)}$, а саме:

$$H_{(n,k)} = \left| R_{k,r}^{T}, I_r \right|, \tag{3.9}$$

де I_r – одинична матриця;

$R_{k,r}^{T}$ – матриця з $G_{(n,k)}$ у транспонованому виді.

Одна з основних задач, що коштують перед розроблювачами пристроїв захисту від помилок при передачі дискретних повідомлень по каналах зв'язку є вибір багаточлена, породжуючий, $g(x)$ для побудови циклічного коду, що забезпечує необхідну мінімальну кодову відстань для гарантійного виявлення й виправлення t-кратних помилок.

Існують спеціальні таблиці на вибір $g(x)$ залежно від пропонованих вимог до коригувальних можливостей коду. Однак у кожного циклічного коду є свої особливості формування $g(x)$. Тому при вивченні конкретних циклічних кодів будуть розглядатися відповідні способи побудови $g(x)$.

Задача кодування полягає у формуванні по інформаційних словах $a(x)$ кодових слів (x) циклічного (n,k)-коду, що по своїй структурі може бути несистематичним і систематичним.

Формування кодових слів несистематичного коду полягає в множенні багаточлена $a(x)$, що відображає інформаційну послідовність довжини k, на породжуючий багаточлен, тобто $(x)=a(x)\times g(x)$. Формування кодових слів систематичного коду полягає в перетворенні інформаційної послідовності $a(x)$ відповідно до вираження $(x)=a(x)\oplus x^r+r(x)$.

Перевірочна послідовність $r(x)$ визначається двома способами:

– при використанні "класичного" способу кодування:

$$r(x) = R_{g(x)}\left[a(x) \cdot x^r \right]; \tag{3.10}$$

– при використанні способу кодування, рекомендованого МККТТ:

$$r(x) = \overline{R}_{g(x)}\left[a(x) \cdot x^r + x(1)^{r-1} \cdot x^k \right], \tag{3.11}$$

де $x(1)^{r-1}$ – одиничний багаточлен ступеня (r-1).

Зазначені вище математичні операції виконують кодерыи несистематичного й систематичного кодів.

Способи декодування з виявленням помилок

Процедура декодування циклічного коду з виявленням помилок, за аналогією із процесом кодування, використовує два способи:

– При кодуванні "класичним" способом декодування засноване на використанні властивості подільності без остачі кодового багаточлена (x) циклічного (n,k)-коду на породжуючий багаточлен $g(x)$. Тому алгоритм декодування містить у собі ділення прийнятого кодового слова, описуваного багаточленом $z'(x)$ на $g(x)$, обчислення й аналіз остачі $r(x)$. Якщо $r(x)$=0, то прийняте кодове слово вважається неспотвореним. Якщо $r(x)$<>0, то прийняте кодове слово стирається й формується сигнал "помилка".

– При кодуванні способом МККТТ декодування засноване на властивості одержання певної контрольної остачі $R_0(x)$ при діленні прийнятого кодового багаточлена (x) на породжуючий багаточлен. Тому, якщо отриманий при діленні остача $\overline{r(x)} = R_0(x)$, то прийняте кодове слово вважається неспотвореним. Якщо остача $\overline{r(x)} \neq R_0(x)$, то прийняте кодове слово стирається й формується сигнал "помилка". Значення контрольної остачі визначається з вираження:

$$R_0(x) = R_{g(x)}\left[x(1)^{r-1} \cdot x^k\right]. \qquad (3.12)$$

Способи декодування з виправленням помилок і схемна реалізація декодувальних пристроїв

Декодування циклічного коду в режимі виправлення помилок можна здійснювати різними способами. Нижче викладаються два способи, що є найбільш простими.

В основу першого способу покладене використання таблиці синдромів (декодування), у якій кожному багаточлену або зразку помилок $e_i(x)$, відповідає певний синдром $S_i(x)$, що представляє остачу від ділення прийнятого кодового

слова $v_i'(x)$ й відповідного йому $e_i(x)$ на $g(x)$. Процедура декодування наступна. Прийняте кодове слово $v_i'(x)$ ділиться на $g(x)$, визначається $S_i(x)$ і відповідний йому багаточлен $e_i(x)$, а потім $v_i'(x)$ підсумується з $e_i(x)$. У результаті одержуємо виправлене кодове слово, тобто:

$$v_i(x) = v_i'(x) + e_i(x). \tag{3.13}$$

До складу декодера входять: обчислювач синдрому (ВР), два регістри зрушення *RG*1 і *RG*2, постійний запам'ятовувальний пристрій (ПЗП), що містить $\sum_{i=1}^{t_u} C_n^i$ слова довжини n, що відповідають багаточленам помилок $e_i(x)$.

Прийняте кодове слово $v_i'(x)$ надходить на вхід обчислювача синдрому, де здійснюється ділення його на $g(x)$ і формування $S_i(x)$, і одночасно – на вхід *RG*2, де $v_i'(x)$ накопичується. Синдром $S_i(x)$ використовується як адреса, по якому із ПЗП в регістр *RG*1 записується $e_i(x)$, що відповідає синдрому $S_i(x)$. Перераховані операції завершуються за n тактів. Протягом наступних n тактів відбувається заелементне підсумовування вмісту *RG*2 і *RG*1, тобто операція:

$$v_i(x) = v_i'(x) + e_i(x), \tag{3.14}$$

і виправлення помилок.

В основі другого способу виправлення помилок, що дозволяє значно скоротити об'єм використовуваних табличних синдромів і істотно спростити схему декодера, лежать наступні положення:

1. Синдром $S_i(x)$, що відповідає прийнятому кодовому слову дорівнює остачі від ділення $v_i'(x)$ на $g(x)$, а також остачі від ділення відповідного багаточлена помилок $ei(x)$ на $g(x)$, тобто:

$$S_i(x) = R_{g(x)}\left[v_i'(x)\right] = R_{g(x)}\left[e_i(x)\right]. \tag{3.15}$$

2. Якщо $S_i(x)$ відповідає $v_i'(x)$ й $e_i(x)$, то $x \in S_i(x)$ є синдромом, що відповідає:

$$R_{g(x)}\left[x \cdot S_i(x)\right] = R_{g(x)}\left[x \cdot v_i'(x) \bmod (x^n - 1)\right] = \\ = R_{g(x)}\left[x \cdot e_i(x) \bmod (x^n - 1)\right], \tag{3.16}$$

і:

$$x \cdot \hat{a}_i'(x) \bmod(x^n - 1), \tag{3.17}$$

або:

$$x \cdot e_i(x) \bmod(x^n - 1). \tag{3.18}$$

3. При виправленні помилок використовуються синдроми зразків помилок тільки з ненульовим коефіцієнтом у старшому розряді.

Тому при реалізації цього способу множина всіх зразків помилок розбивається на класи еквівалентності. Кожний клас представляє циклічне зрушення одного зразка помилок, а синдром цього класу відповідає зразку помилок з ненульовим старшим розрядом. Якщо обчислений синдром належить одному із класів еквівалентності зразків помилок, що виправляються, то старший символ кодового слова виправляється. Потім прийняте слово й синдром циклічно зрушується, а процес знаходження в попередній по старшинству позиції повторюється.

Для виправлення помилок, що належать даному класу еквівалентності, потрібно зробити *n* циклічних зрушень.

Найпростішим є декодер Меггітта. До складу декодера входять: обчислювач синдрому, що здійснює ділення кодового слова $\hat{a}_i'(x)$ на $g(x)$ і формування відповідного синдрому; блок декодерів (ДК), що настроєний на синдроми всіх зразків помилок, що виправляються, з ненульовими старшими розрядами; регістр зрушення *RG*.

При надходженні на вхід схеми кодового слова $\hat{a}_i'(x)$ його символи заповнюють регістр *RG*, а в обчислювачі формується відповідний синдром $S_i(x)$. Обчислений синдром рівняється з усіма табличними синдромами, закладеними в схему блоку ДК, і у випадку збігу з одним з них на його виході формується сигнал, що виправляє помилковий символ, що перебуває в старшому розряді регістра. Після цього вміст обчислювача й *RG* циклічно зрушується на один крок. Це зрушення реалізує операції $R_{g(x)}[x \cdot S_i(x)]$ й $x \cdot \hat{a}_i'(x) \bmod(x^n - 1)$. Якщо новий

синдром збігається з одним з табличних синдромів, то це означає, що відбулася помилка в другому по старшинству символі кодового слова, що, перейшовши в старший розряд *RG*, виправляється. Потім виробляється нове циклічне зрушення на одну позицію й нову перевірку на збіг синдромів. Після повторення цього процесу *n* раз в *RG* буде сформоване виправлене кодове слово. Введення зворотного зв'язка для *RG* не обов'язково, тому що в процесі виправлення помилок символи кодового слова надходять на вихід декодера.

При декодуванні циклічних кодів використовуються багаточлен помилок $e(x)$ і синдромний багаточлен $S(x)$.

Багаточлен помилок ступеня не більше (n-1) визначається з вираження:

$$e(x) = v'(x) + v(x), \tag{3.19}$$

де $v'(x)$ и $v(x)$ – багаточлени, що відображають відповідно прийняте (з помилкою) і передане кодові слова.

Ненульові коефіцієнти в $e(x)$ займають позиції, які відповідають помилкам.

Синдромний багаточлен, використовуваний при декодуванні циклічного коду, визначається як остача від ділення прийнятого кодового слова на породжуючий багаточлен, тобто:

$$S_i(x) = R_{g(x)}[v'(x)], \tag{3.20}$$

або:

$$S_i(x) = R_{g(x)}[v'(x) + e_i(x)] = R_{g(x)}[e_i(x)]. \tag{3.21}$$

Отже, синдромний багаточлен залежить безпосередньо від багаточлена помилок $e(x)$.Це положення використовується при побудові таблиці синдромів, застосовуваної в процесі декодування. Ця таблиця містить список багаточленів помилок і список відповідних синдромів, обумовлених з вираження:

$$S_i(x) = R_{g(x)}[e_i(x)] \tag{3.22}$$

Таблиця 3.2 – Список багаточленів помилок і список відповідних синдромів

(x)	$S(x)$
1	$R_{g(x)}[1]$
X	$R_{g(x)}[x]$
X^2	$R_{g(x)}[x^2]$
X+1	$R_{g(x)}[x+1]$
X^2+1	$R_{g(x)}[x^2+1]$

У процесі декодування по прийнятому кодовому слову обчислюється синдром, потім у таблиці перебуває відповідний багаточлен $e(x)$, підсумовування якого із прийнятим кодовим словом дає виправлене кодове слово, тобто:

$$u_i(x) = u_i'(x) + e_i(x). \tag{3.23}$$

Перераховані багаточлени $v(x), v'(x), g(x), h(x), e(x)$ и $S(x)$ можна складати, множити й ділити, використовуючи відомі правила алгебри, але із приведенням результату по mod 2, а потім по mod x^n+1, якщо ступінь результату перевищує ступінь (n-1).

При побудові й декодуванні циклічних кодів у результаті ділення багаточленів звичайно необхідно мати не частка, а остача від ділення.

Тому рекомендується більш простий спосіб ділення, використовуючи не багаточлени, а тільки його коефіцієнти.

На рисунку 3.2 зображена функціональна схема системи. Нижче розглянемо її більш докладно.

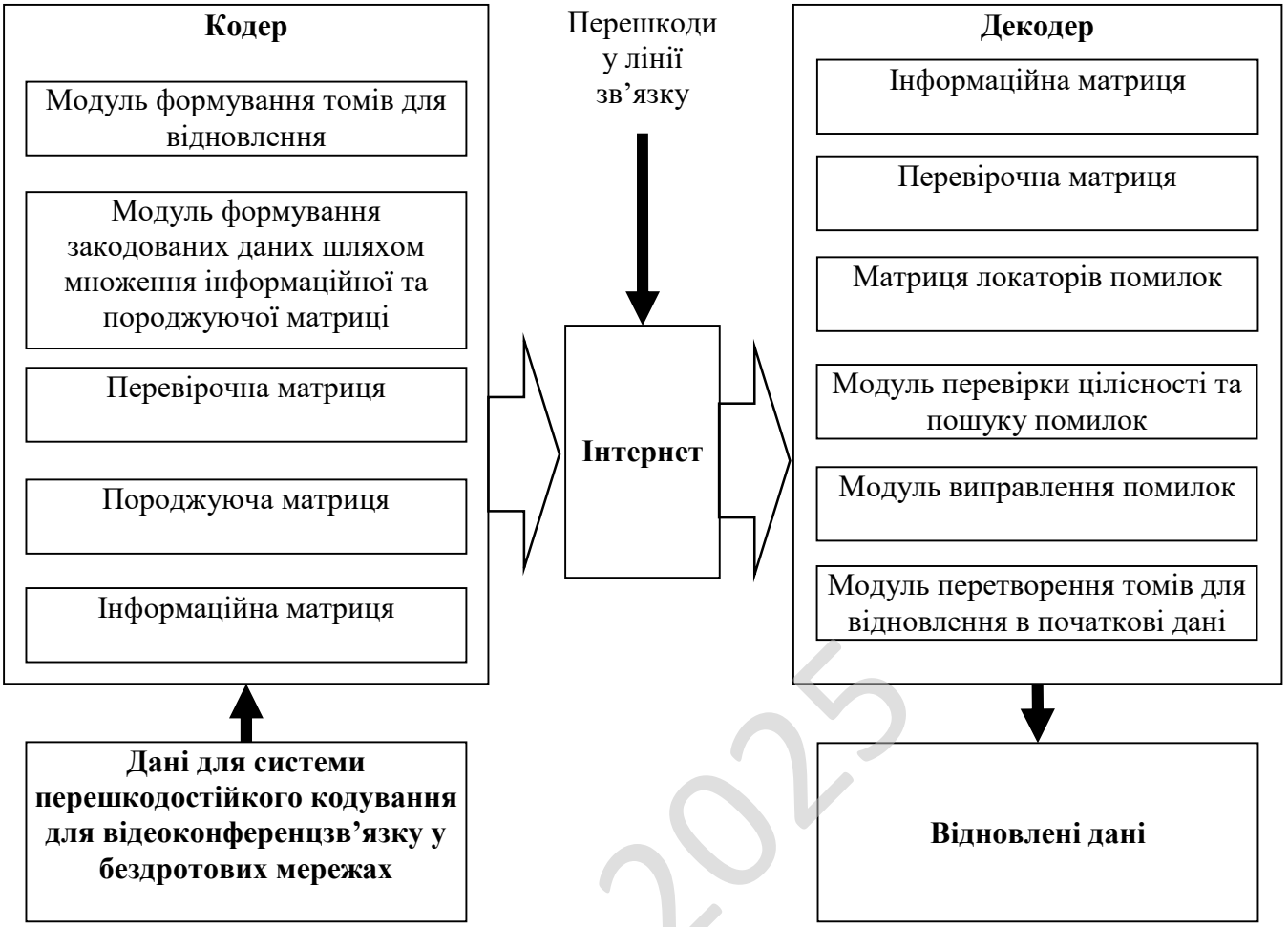

Рисунок 3. 2 – Функціональна схема системи

З неї бачимо, що розроблена система перешкодостійкого кодування для відеоконференцзв'язку у бездротових мережах складається з наступних основних функціональних блоків:

– клієнт відеоконференцзв'язку;

– кодер циклічного коду, який використовується, коли відбувається відправлення інформації у системі перешкодостійкого кодування для відеоконференцзв'язку у бездротових мережах;

– декодер циклічного коду, який використовується при читанні даних з мережі;

– дані для перешкодостійкого збереження

– відновлені дані.

Розглянемо більш детально перераховані функціональні блоки кодування та декодування за допомогою циклічного кодування.

Блок кодування складається з наступних підблоків:

– Модуль формування пакетів для відновлення.

– Модуль формування закодованих даних шляхом множення інформаційної та породжуючої матриці.

– Перевірочна матриця.

– Породжуюча матриця.

– Інформаційна матриця.

Блок декодування складається з наступних підблоків:

– Інформаційна матриця.

– Перевірочна матриця.

– Матриця локаторів помилок.

– Модуль перевірки цілісності та пошуку помилок.

– Модуль виправлення помилок.

– Модуль перетворення пакетів для відновлення в початкові дані.

Коди циклічного коду базуються на спеціальному розділі математики – полях Галуа (GF) або кінцевих полях. Арифметичні дії (+, -, х, / і т.д.) над елементами кінцевого поля дають результат, що також є елементом цього поля. Кодер та декодер циклічного коду повинні вміти виконувати ці арифметичні операції. Ці операції для своєї реалізації вимагають спеціального устаткування або спеціалізованого програмного забезпечення.

Кодове слово циклічного коду формується із залученням спеціального полінома. Всі коректні кодові слова повинні ділитися без залишку на ці утворюючі поліноми. Загальна форма утворюючого полінома має вигляд:

$$g(x) = (x\text{-}a^{i})(x\text{-}ai^{+1})\ldots(x\text{-}a^{i+2t}), \quad (3.24)$$

а кодове слово формується за допомогою операції:

$$c(x) = g(x)i(x), \quad (3.25)$$

де – $g(x)$ є утворюючим поліномом;

– $i(x)$ являє собою інформаційний блок;

– $c(x)$ – кодове слово, що називається простим елементом поля.

$2t$ символів парності в кодовому слові циклічного коду визначаються з наступного співвідношення:

$$p(x) = i(x) \cdot x^{n-k} \bmod g(x). \quad (3.26)$$

Перейдемо до розгляду іншого функціонального блоку – декодеру циклічного коду.

Декодер працює наступним чином.

Введемо позначення:

– $r(x)$ – Отримане кодове слово.

– S_i – Синдроми.

– $L(x)$ – Поліном локації помилок.

– X_i – Положення помилок.

– Y_i – Значення помилок.

– $c(x)$ – Відновлене кодове слово.

– v – Число помилок.

Отримане кодове слово r(x) являє собою вихідне (передане) кодове слово c(x) плюс помилки: $r(x) = c(x) + e(x)$.

Декодер циклічного коду намагається визначити позицію й значення помилки для числа t помилок (або $2t$ втрат) і виправити помилки й втрати.

Обчислення синдрому

Обчислення синдрому схоже на обчислення парності. Кодове слово циклічного коду має $2t$ **синдромів**, це залежить тільки від помилок (а не переданих кодових слів). Синдроми можуть бути обчислені шляхом підстановки $2t$ коріння утворюючого полінома $g(x)$ в $r(x)$.

Знаходження позицій символьних помилок

Це робиться шляхом рішення системи рівнянь із t невідомими. Існує кілька швидких алгоритмів для рішення цього завдання. Ці алгоритми використовують особливості структури матриці кодів циклічного коду й сильно

скорочують необхідну обчислювальну потужність. Робиться це у два етапи:

1. Визначення полінома локації помилок

Це може бути зроблене за допомогою алгоритму Berlekamp-Massey або алгоритму Евкліда. Алгоритм Евкліда використовується частіше на практиці, тому що його легше реалізувати, однак, алгоритм Berlekamp-Massey дозволяє одержати більш ефективну реалізацію встаткування й програм.

2. Знаходження кореня цього полінома. Це робиться із залученням алгоритму пошуку Chien.

Знаходження значень символьних помилок

Тут також потрібно вирішити систему рівнянь із t невідомими. Для рішення використовується швидкий алгоритм Forney.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.3. Після початку роботи розробленого ПЗ ми потрапляємо до головного блоку системи звідки через ланку дій відбувається наступне:

– Інтерфейс ПЗ.

– Налаштування системи.

– Налаштування бездротових мереж.

– Налаштування загальної кількості пакетів.

– Моніторинг відеоконференцзв’язку.

– Кодування мультимедіа даних.

– Журнал роботи ПЗ.

– Створення пакетів для відновлення даних.

– Декодування.

– Відновлення пошкоджених пакетів.

– Перевірка цілісності даних.

– Встановлення шифруючого фільтру.

– Встановлення паролю на закодовані дані.

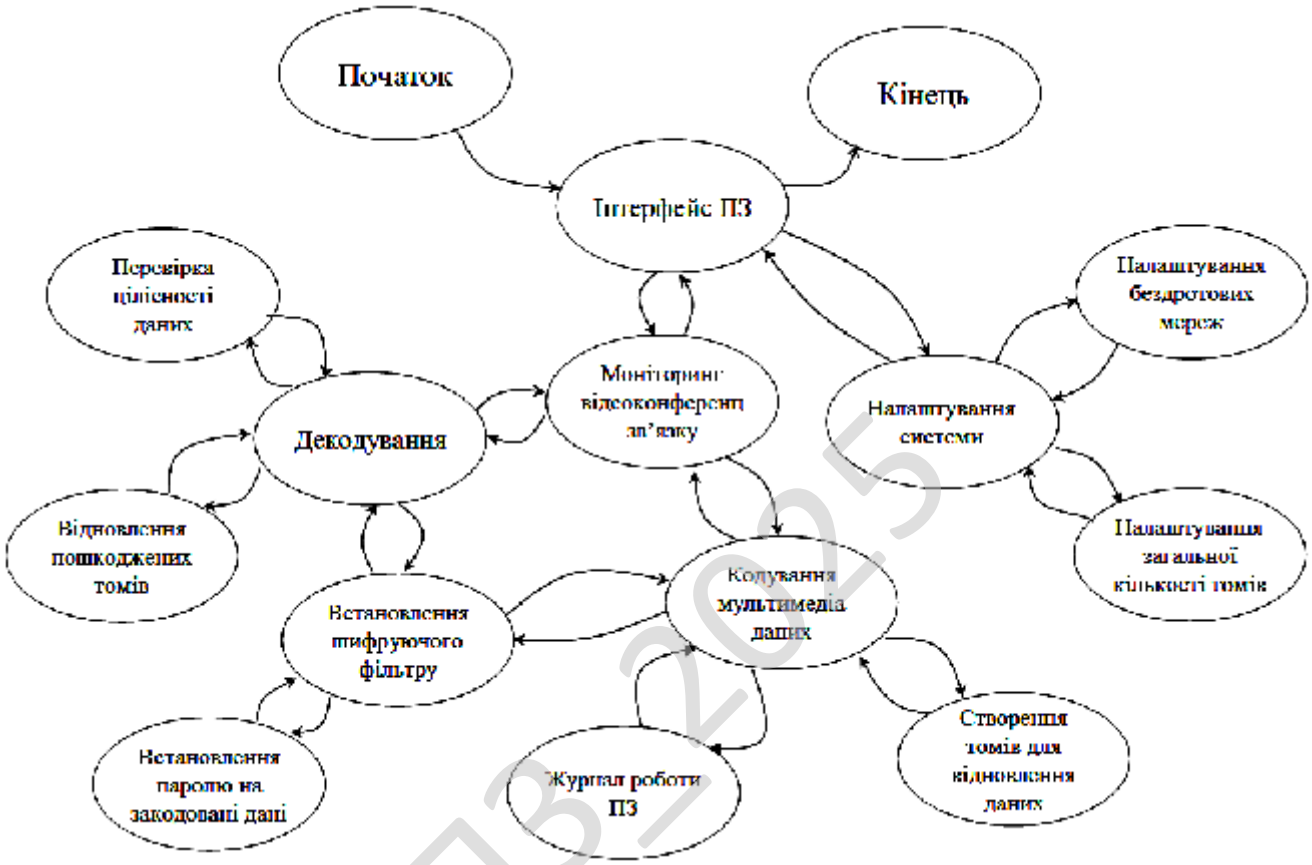

Рисунок 3.3 – Діаграма взаємодії процесів

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Первинною стадією без якої не відбувається розробка програмного забезпечення це звичайно розробка блок-схем. На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 зображено роботу підпрограми.

З якої видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підпрограми та останньої стадії – перевірка поточного стану з завершенням роботи розробленого ПЗ.

При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

Блок-схеми є першоджерелами стратегії розвитку ПЗ. Тому від точності і детальної блок-схеми залежить результат всієї програми.

При виборі початкової точки відліку при побудові схем я враховував, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає оптимізації програми високого рівня, також те, що при розробці програми слід надати особливу увагу модулю кодування для відеоконференцзв’язку у бездротових мережах.

Крім цього було використано підходи з використанням UML, це уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування.

Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, називаної UML-моделлю. UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація.

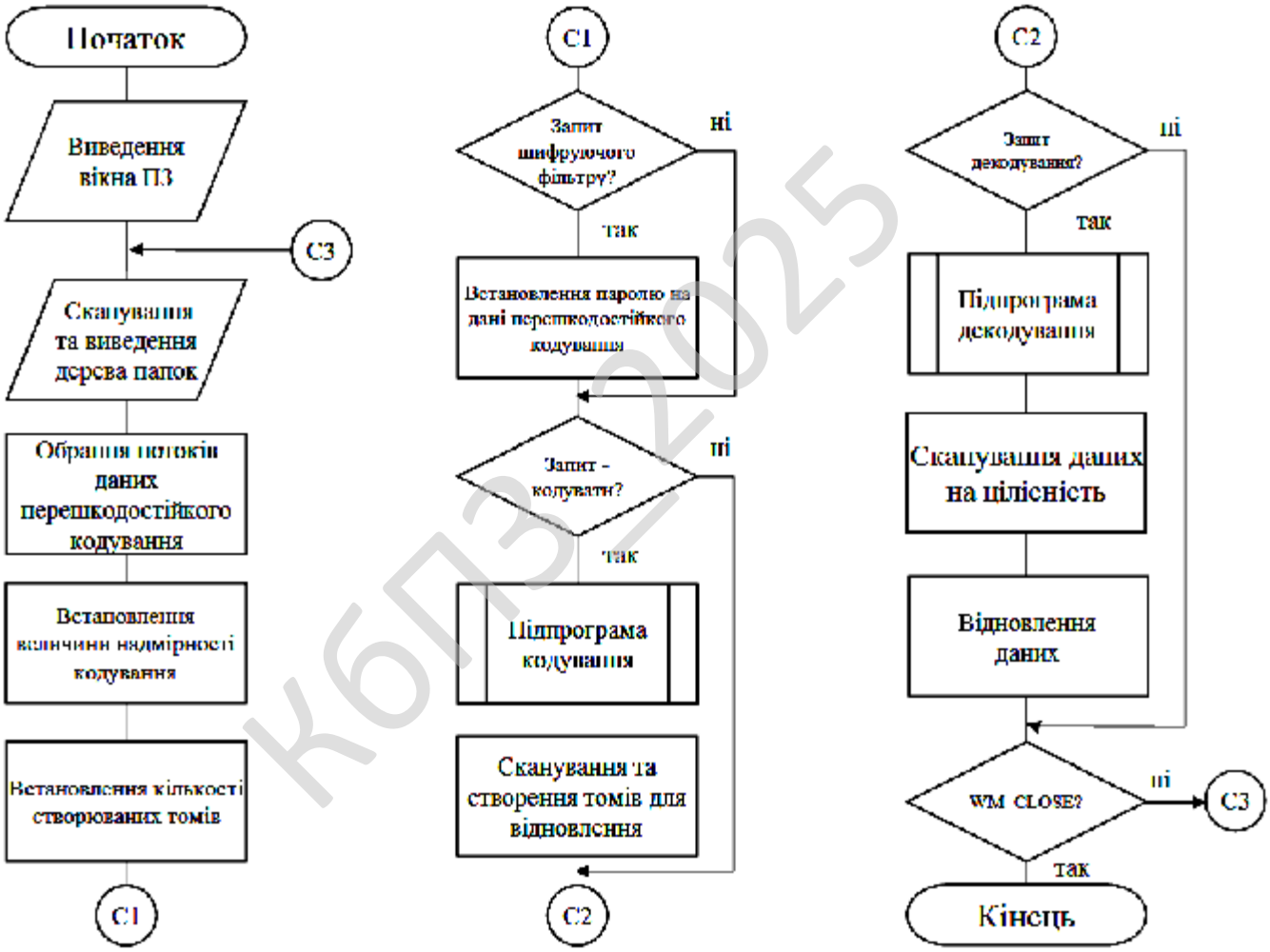

Рисунок 4.1 – Блок схема основної програми

UML може бути застосовано на всіх етапах життєвого циклу аналізу бізнес-систем і розробки прикладних програм. Різні види діаграм які підтримуються UML, і найбагатший набір можливостей представлення певних

аспектів системи робить UML універсальним засобом опису як програмних, так і ділових систем.

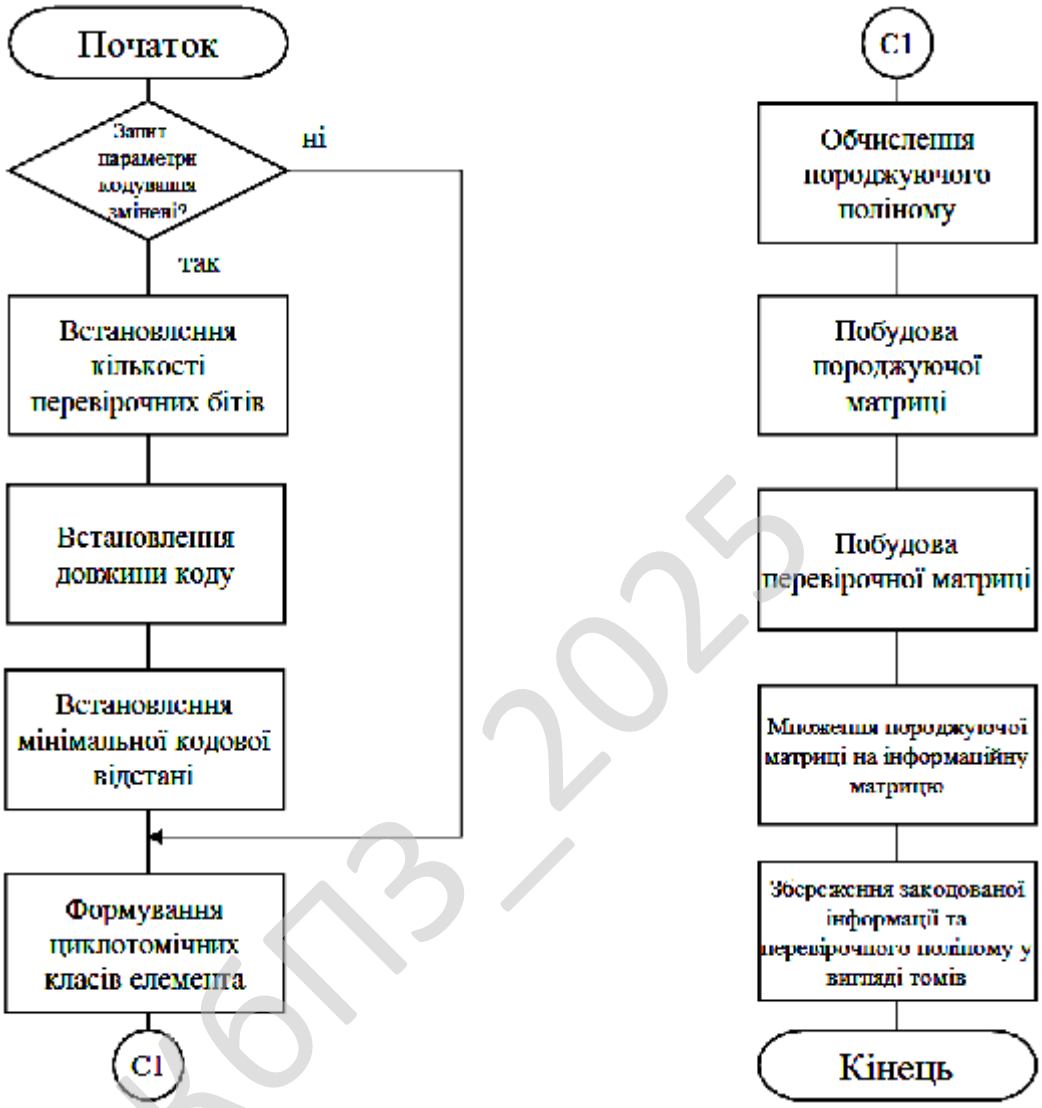

Рисунок 4.2 – Блок схема підпрограми

Діаграми дають можливість представити систему (як ділову, так і програмну) у такому вигляді, щоб її можна було легко перевести в програмний код. Основною причиною використання мови UML є спілкування розробників між собою.

Крім того, UML спеціально створювалася для оптимізації процесу розробки програмних систем, що дозволяє збільшити ефективність їх реалізації у кілька разів і помітно поліпшити якість кінцевого продукту.

UML прекрасно зарекомендувала себе в багатьох успішних програмних проектах. Засоби автоматичної генерації кодів дозволяють перетворювати моделі мовою UML у вихідний код об’єктно-орієнтованих мов програмування, що ще більш прискорює процес розробки. Практично усі CASE-засоби (програми автоматизації процесу аналізу і проектування) мають підтримку UML. Моделі розроблені в UML, дозволяють значно спростити процес кодування і направити зусилля програмістів безпосередньо на реалізацію системи.

Діаграми підвищують супроводжуваність проекту і полегшують розробку документації.

UML необхідний:

– Керівникам проектів, які керують розподілом завдань і контролем за проектом.

– Проектувальникам інформаційних систем які розробляють технічні завдання для програмістів.

– Бізнес-аналітикам, які досліджують реальну систему і здійснюють інжиніринг і реінжиніринг бізнесу компанії.

– Програмістам які реалізовують модулі інформаційної системи.

При модифікації системи об'єктний підхід дозволяє легко включати в систему нові об'єкти і виключати застарілі без істотної зміни її життєздатності. Використання побудованої моделі при модифікаціях системи дає можливість усунути небажані наслідки змін, оскільки вони не ламають структури системи, а тільки змінюють поведінку об'єктів.

При складанні блок-схем програмного забезпечення і напрацювання алгоритмів я зіткнувся з масою проблем, які вимагали напрацювання процедур і функцій над основною проблематикою. Для чого були створені додаткові класи, типи даних і константи, що забезпечило вирішення проблем.

Розглянемо опис алгоритмів функціонування системи. Розглянемо декодер циклічного кодування.

Декодування кодів циклічного кодування являє собою досить складне завдання, рішення якого виливається в громіздкий, заплутаний і надзвичайно ненаглядний програмний код, що вимагає від розроблювача великих знань у багатьох областях вищої математики.

Типова схема декодування, що одержала назву авторегрессионого спектрального методу декодування, складається з наступних кроків:

– Обчислення синдрому помилки (синдромний декодер).

– Побудови полінома помилки, здійснювана або за допомогою високоефективного, але складно реалізованого алгоритму Берлекемпа-Мессі, або за допомогою простого, але гальмового Евклідового алгоритму.

– Знаходження корінь даного полінома, що звичайно вирішується лобовим перебором (алгоритм Ченя).

– Визначення характеру помилки, що зводиться до побудови бітової маски, що обчислюється на основі обігу алгоритму Форні або будь-якого іншого алгоритму обігу матриці.

– Нарешті, виправлення помилкових символів, шляхом накладення бітової маски на інформаційне слово й послідовне інвертування всіх перекручених біт через операцію XOR.

Перейдемо до розгляду блок-схеми процедури декодування.

Спершу відбувається перетворення кадру в синдромний поліном та обчислюються коефіцієнти полінома синдрому. Якщо усі коефіцієнти дорівнюють нулю, то відбувається повне відновлення даних й видається повідомлення, що помилок не знайдено.

У іншому випадку формується ключова задача декодування й знаходження поліному локаторів помилок по алгоритму Берлекемпа-Мессі. Якщо поліном локаторів не знайдено, то помилки не підлягають виправленню, про що виводиться відповідне повідомлення.

У іншому випадку знаходяться корні рівняння локаторів, виділяються локатори та відбувається перевірка локаторів на умову. Якщо локатори не відповідають умові то помилки не підлягають виправленю, про що виводиться відповідне повідомлення. У іншому випадку відбувається обчислення поліному величини помилок, обчислюється його похідна, та обчислюються величини помилок. На основі цих даних формується поліном викривлень та відбувається корекція помилок у кадрі, у результаті чого дані відновлюються, про видається відповідне повідомлення.

Нижче наведемо вихідний текст підпрограми декодера циклічного кодування.

```
namespace R_Disk
{
/// Клас декодера циклічного коду
    public class CicleCodeDecoder : CicleCodeBase
    {
        #region Data
// Масив булевських ознак "рядок матриці "FLog" тривіальна?"
        private bool[] FLogRowIsTrivial;
// Список порядкових номерів наявних пакетів (нумерація з нуля)
        private int[] volList;
        #endregion Data
        #region Construction & Destruction

/// Конструктор декодера за замовчуванням
        public CicleCodeDecoder()
        {
// Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }

/// Базовий конструктор декодера
/// <param name="dataCount">Кількість основних пакетів</param>
/// <param name="eccCount">Кількість пакетів для відновлення</param>
/// <param name="volList">Список порядкових номерів наявних пакетів</param>
        public CicleCodeDecoder(int dataCount, int eccCount, int[] volList)
        {
// Установка конфігурації кодера
```

```
            SetConfig(dataCount, eccCount, volList,
(int)CicleCodeType.Alternative);

// Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }

/// Розширений конструктор декодера
/// <param name="dataCount">Кількість основних пакетів</param>
/// <param name="eccCount">Кількість пакетів для відновлення</param>
/// <param name="volList">Список порядкових номерів наявних пакетів</param>
/// <param name="codecType">Тип кодера циклічного коду (по типу матриці)</param>
public CicleCodeDecoder(int dataCount, int eccCount, int[] volList, int codecType)
        {
// Установка конфігурації кодера
            SetConfig(dataCount, eccCount, volList, codecType);
// Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }
        #endregion Construction & Destruction
        #region Public Operations

/// Установка конфігурації декодера
/// <param name="dataCount">Кількість основних пакетів</param>
/// <param name="eccCount">Кількість пакетів для відновлення</param>
/// <param name="volList">Список порядкових номерів наявних пакетів</param>
/// <param name="codecType">Тип кодека кодера циклічного коду (по типу
матриці)</param>
/// <returns>Булевський прапор операції установки конфігурації</returns>
        public bool SetConfig(int dataCount, int eccCount, int[] volList, int
codecType)
        {
            int maxVolCount;
// Установлюємо константи, що відповідають обраному режиму
            if (codecType == (int)CicleCodeType.Dispersal)
            {
                maxVolCount = (int)CicleCodeConst.MaxVolCountDisp;
            } else
            {
                maxVolCount = (int)CicleCodeConst.MaxVolCountAlt;
            }
// Перевіряємо конфігурацію на коректність
```

```
            if (
                    (dataCount > 0)
                 &&
                    (eccCount > 0)
                 &&
                    ((dataCount + eccCount) <= maxVolCount)
                 &&
                    (volList.Length >= dataCount)
                )
            {
// Якщо основна конфігурація змінилася - сповіщаємо про це
                if (
                        (dataCount != this.n)
                     ||
                        (eccCount != this.m)
                     ||
                        (codecType != this.eCicleCodeType)
                    )
                {
                    this.mainConfigChanged = true;
                }
// Зберігаємо конфігурацію
                this.n = dataCount;
                this.m = eccCount;
                this.eCicleCodeType = codecType;

// Також перераховуємо кількість ітерацій всіх стадій підготовки
                double n = this.n;
                double m = this.m;
// Нормалізуємо значення для розрахунку, щоб уникнути переповнення змінних
                NormalizeNM(ref n, ref m);
// Кількість ітерацій, що відслідковуються прогресом, на першій стадії
// залежить від типу використовуваної матриці
                if (this.eCicleCodeType == (int)CicleCodeType.Alternative)
                {
                    this.iterOfFirstStage = m;
                } else
                {
          this.iterOfFirstStage = ((n * m) * n) + (n * ((n + m) + (n * (n + m))));
                }
                this.iterOfSecondStage = (n * (((n - 1) * (n - 1)) + (n * n)));
```

```
// Виділяємо пам'ять під масив булевських ознак "рядок матриці "FLog" тривіальна?"
                this.FLogRowIsTrivial = new bool[dataCount];
// Зберігаємо список наявних пакетів
                this.volList = volList;
                this.configIsOK = true;
            } else
            {
//...вказуємо на помилку конфігурації
                this.configIsOK = false;
            }

            return this.configIsOK;
        }

/// Метод множення матриці кодування на вхідний прологарифмований вектор
/// <param name="dataEccLog">Прологарифмований вхідний вектор (дані + ecc)</param>
/// <param name="data">Вихідний вектор (відновлені вихідні дані)</param>
/// <returns>Булевський прапор результату операції</returns>
        public bool Process(int[] dataEccLog, ref int[] data)
        {
// Якщо кодер зконфігуровано некоректно, обробка неможлива!
            if (!this.configIsOK)
            {
                return false;
            }
// Копіюємо покажчик на масив експонент для скорочення часу обігу
            int[] GF16Exp = this.eGF16.GFExpTable;
// Обчислення результату множення матриці на вектор
            for (int i = 0; i < this.n; i++)
            {
// Якщо поточний рядок матриці не є тривіальної, робимо обробку
                if (!this.FLogRowIsTrivial[i])
                {
                    int mulSum = 0; // Сума добутку рядка матриці на стовпець
                    int i_n = i * this.n; // Зсув у масиві до елементів i-ой рядка
                    for (int j = 0; j < this.n; j++)
                    {
                        mulSum ^= GF16Exp[this.FLog[i_n + j] + dataEccLog[j]];
                    }
                    data[i] = mulSum;
                } else
                {
```

```
                    data[i] = GF16Exp[dataEccLog[i]];
                }
            }
            return true;
        }
        #endregion Public Operations
        #region Private Operations

/// Пошук матриці, зворотної до "FLog", методом Жорданових виключень
/// (Дана модифікація методу може використовуватися тільки в тих випадках,
/// коли (-a) = (a), тому що через непотрібність пропущена стадія зміни
елементів),
/// крім того, відсутній пошук ненульового розв'язного елемента (у випадку
/// роботи з матрицею Вандермонда наявність нуля на діагоналі - збій кодека,
/// тому ситуація з виявленням нуля сприймається винятково як помилка
/// <returns>Булевський прапор результату операції</returns>
        private bool FInv()
        {
// Обчислюємо розподіл відсотків ітерацій по стадіях для
// коректної обробки відсотків
            double allStageIter   = this.iterOfFirstStage +
this.iterOfSecondStage;
            int percOfFirstStage  = (int)((100.0 * this.iterOfFirstStage)  /
allStageIter);
            int percOfSecondStage = (int)((100.0 * this.iterOfSecondStage) /
allStageIter);
// Дана стадія повинна займати хоча б один відсоток
// (для коректності розрахунків)
            if (percOfSecondStage == 0)
            {
                percOfSecondStage = 1;
            }
// Обчислюємо значення модуля, що дозволить виводити відсоток обробки
// рівно при одиничному збільшенні для циклу по "k"
            int progressMod1 = this.n / percOfSecondStage;
// Якщо модуль дорівнює нулю, то збільшуємо його до значення "1", щоб
// прогрес виводився на кожній ітерації
            if (progressMod1 == 0)
            {
                progressMod1 = 1;
            }
// Цикл вибору розв'язного елемента "pivot"
```

```
            for (int k = 0; k < this.n; ++k)
            {
// Якщо даний рядок тривіальна - просто переходимо на нову ітерацію
                if (this.FLogRowIsTrivial[k])
                {
                    continue;
                }
// Зсув у масиві до елементів k-ой рядка
                int k_n = k * this.n;
// Індекс розв'язного елемента
                int pivotIdx = k_n + k;
// Витягаємо розв'язний елемент
                int pivot = this.FLog[pivotIdx];
// Якщо розв'язний елемент дорівнює нулю - матриця не має зворотної
                if (pivot == 0)
                {
//...указуємо на помилку конфігурації
                    this.configIsOK = false;
// Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;
// Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();
                    return false;
                }
// Після добування розв'язного елемента поміщаємо на його місце "1"
                this.FLog[pivotIdx] = 1;
// Працюємо з рядками...
                for (int i = 0; i < this.n; i++)
                {
// Якщо перебуваємо на рядку розв'язного елемента - переходимо
// на нову ітерацію
                    if (i == k)
                    {
                        continue;
                    }
// Зсув у масиві до елементів i-ой рядка
                    int i_n = i * this.n;
// Працюємо зі стовпцями...
                    for (int j = 0; j < this.n; j++)
                    {
// Якщо перебуваємо на стовпці розв'язного елемента - переходимо
// на нову ітерацію...
```

```
                        if (j == k)
                        {
                            continue;
                        }
                        int idx = i_n + j;
//...а інакше робимо необхідні дії над матрицею:
// "A[i,j] = A[i,j] * pivot + A[i,k] * A[k,j]"
                        this.FLog[idx] = this.eGF16.Mul(this.FLog[idx], pivot) ^
this.eGF16.Mul(this.FLog[i_n + k], this.FLog[k_n + j]);
                    }
                }
// Розподіл матриці на розв'язний елемент заміняємо множенням на зворотний
                int pivotInv = this.eGF16.Inv(pivot);

                for (int i = 0; i < this.n; i++)
                {
// Зсув у масиві до елементів i-ой рядка
                    int i_n = (i * this.n);
                    for (int j = 0; j < this.n; j++)
                    {
                        int idx = i_n + j;
                        this.FLog[idx] = this.eGF16.Mul(this.FLog[idx], pivotInv);
                    }
                }
// Якщо є передплата на делегата відновлення прогресу -...
                if (
                        ((k % progressMod1) == 0)
                    &&
                        (OnUpdateCicleCodeMatrixFormingProgress != null)
                   )
                {
//...виводимо дані
                    OnUpdateCicleCodeMatrixFormingProgress((((double)(k + 1) /
(double)this.n) * percOfSecondStage) + percOfFirstStage);
                }
// У випадку, якщо потрібна постановка на паузу, подію "executeEvent"
// буде скинуто, і будемо на паузі аж до його появи
                ManualResetEvent.WaitAll(this.executeEvent);
// Якщо зазначено, що потрібно вийти з потоку - виходимо
                if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                {
// Указуємо, що декодер зконфігуровано некоректно
```

```
                    this.configIsOK = false;
// Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;
// Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();
                    return false;
                }
            }
            return true;
        }

/// Обчислення логарифмів значень інвертованої матриці

        private void LogFCalc()
        {
// Працюємо з рядками...
            for (int i = 0; i < this.n; i++)
            {
// Зсув у масиві до елементів i-ой рядка
                int i_n = i * this.n;
// Працюємо зі стовпцями...
                for (int j = 0; j < this.n; j++)
                {
                    int idx = i_n + j;
                    this.FLog[idx] = this.eGF16.Log(this.FLog[idx]);
                }
            }
        }

/// Заповнення матриці "FLog" (матриці декодера) даними
        protected override void FillFLog()
        {
// Якщо довжина вектора наявних пакетів менше кількості,
// необхідного для відновлення...
            if (this.volList.Length < this.n)
            {
//...указуємо на помилку конфігурації
                this.configIsOK = false;
// Активуємо індикатор актуального стану змінних-членів
                this.finished = true;
// Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();
```

```
                return;
            }
// Виділяємо пам'ять під матрицю "FLog"
            this.FLog = new int[this.n * this.n];
// Вектор лічильників всіх пакетів...
            int[] allVolCount = new int[this.n + this.m];
//...і вектор ecc-пакетів для "затикання" пробілів, створених
// загубленими основними томами
            int[] eccVolToFix = new int[this.m];
// Лічильник кількості стертих основних пакетів
            int dataVolMissCount = this.n;
// Ініціалізуємо масив лічильників всіх пакетів
            for (int i = 0; i < (this.n + this.m); i++)
            {
                allVolCount[i] = 0;
            }
// Проводимо аналіз складу представлених пакетів на предмет наявності основних
            for (int i = 0; i < this.n; i++)
            {
// Обчислюємо номер поточного тому
                int currVol = Math.Abs(this.volList[i]);
// Якщо номер тому відповідає припустимому діапазону
                if (currVol < (this.n + this.m))
                {
                    ++allVolCount[currVol];
// Якщо поточний том є основним, фіксуємо даний факт
                    if (currVol < this.n)
                    {
                        ---idataVolMissCount;
                    }
                } else
                {
// Указуємо на помилку конфігурації
                    this.configIsOK = false;
// Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;
// Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();
                    return;
                }
            }
// Перевіряємо лічильники пакетів на помилкове дублювання
```

```
            for (int i = 0; i < (this.n + this.m); i++)
            {
// Якщо деякий том був зазначений більш ніж один раз...
                if (allVolCount[i] > 1)
                {
//...указуємо на помилку конфігурації
                    this.configIsOK = false;
// Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;
// Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();
                    return;
                }
            }
// Якщо перевірка на несуперечність не виявила проблем, починаємо
// формувати матрицю "FLog"
// Якщо основна конфігурація змінилася...
            if (this.mainConfigChanged)
            {
                if (this.eCicleCodeType == (int)CicleCodeType.Dispersal)
                {
//...робимо формування дисперсної матриці "D"
                    if (!MakeDispersalMatrix())
                    {
// Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;
// Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;
// Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();
                        return;
                    }
                } else
                {
//...робимо формування альтернативного заповнення матриці "A"
                    if (!MakeAlternativeMatrix())
                    {
// Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;
// Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;
// Установлюємо подію завершення обробки
```

```
                        this.finishedEvent[0].Set();
                        return;
                    }
                }
//...і скидаємо прапор
                this.mainConfigChanged = false;
            }
// Для кожного загубленого основного тому шукаємо том для відновлення
            for (int i = 0, j = 0; i < dataVolMissCount; i++)
            {
// Рухаємося за списком пакетів доти, поки не знайдемо том для
// відновлення для затикання "дірки" (основні томи мають номера
// менше this.n (при нумерації з нуля!))
                while (this.volList[j] < this.n)
                {
                    j++;
                }
// Зберігаємо номер тому для заміни загубленого основного тому
                eccVolToFix[i] = this.volList[j];
                j++; // j++ дозволяє перейти до наступного пошуку
            }
// Працюємо по рядках матриці (в ідеалі, всі рядки повинні заповнюватися
// рядками з одиницею на головній діагоналі, що відповідає відсутності
// ушкоджень, але allVolCount укаже, якими є справи з наявністю пакетів)
            for (int i = 0, e = 0; i < this.n; i++)
            {
// Індекс рядка з дисперсної матриці, що буде поміщена в матрицю кодування
                int DRowIdx;
// Зсув у масиві до елементів i-ой рядка
                int i_n = i * this.n;
// Якщо основний том відсутній, формуємо рядок матриці Вандермонда
                if (allVolCount[i] == 0)
                {
// Обчислюємо номер рядка матриці Вандермонда, яку потрібно вставити
// на місце даного рядка формованої матриці "FLog"
                    DRowIdx = eccVolToFix[e++];
// Указуємо, що даний рядок матриці "FLog" не тривіальна
                    this.FLogRowIsTrivial[i] = false;
                } else
                {
// Формуємо в матриці "FLog" нульовий рядок з одиницею на головній діагоналі
// (відповідає наявному основному той)
```

```
                    DRowIdx = i;
// Указуємо, що даний рядок матриці "FLog" тривіальна
                    this.FLogRowIsTrivial[i] = true;
                }
// Залежно від типу декодера беремо дані з відповідного масиву
// (у ньому втримуються як рядки матриці Вандермонда, так і "тривіальні" рядки,
//  утримуючі нулі і єдиний елемент "1" на головній діагоналі)
                if (this.eCicleCodeType == (int)CicleCodeType.Dispersal)
                {
                    int bs = DRowIdx * this.n;
// Формування рядка в матриці кодування
// ("тривіальні" рядки вже втримуються в матриці "D", вони вийшли
// "автоматично" на попередньому етапі обробки MakeDispersal())
                    for (int j = 0; j < this.n; j++)
                    {
                        this.FLog[i_n + j] = this.D[bs + j];
                    }
                } else
                {
// Якщо це потрібно - формуємо "тривіальну" рядок...
                    if (this.FLogRowIsTrivial[i])
                    {
                        for (int j = 0; j < this.n; j++)
                        {
                            this.FLog[i_n + j] = 0;
                        }
                        this.FLog[i_n + i] = 1;
                    } else
                    {
                        int bs = (DRowIdx - this.n) * this.n;
//...а, інакше, беремо рядок матриці Вандермонда
                        for (int j = 0; j < this.n; j++)
                        {
                            this.FLog[i_n + j] = this.A[bs + j];
                        }
                    }
                }
// У випадку, якщо потрібна постановка на паузу, подію "executeEvent"
// буде скинуто, і будемо на паузі аж до його появи
                ManualResetEvent.WaitAll(this.executeEvent);
// Якщо зазначено, що потрібно вийти з потоку - виходимо
                if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
```

```
                {
//...указуємо на помилку конфігурації
                    this.configIsOK = false;
// Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;
// Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();
                    return;
                }
            }
// Знаходимо зворотну матрицю для "FLog"
            if (!FInv())
            {
// Указуємо, що кодер зконфігуровано некоректно
                this.configIsOK = false;
// Активуємо індикатор актуального стану змінних-членів
                this.finished = true;
// Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();
                return;
            }
// Обчислюємо логарифми елементів інвертованої матриці
            LogFCalc();
// Якщо є передплата на делегата завершення...
            if (OnCicleCodeMatrixFormingFinish != null)
            {
//...повідомляємо, що екземпляр класу готовий до роботи
                OnCicleCodeMatrixFormingFinish();
            }
// Активуємо індикатор актуального стану змінних-членів
            this.finished = true;
// Установлюємо подію завершення обробки
            this.finishedEvent[0].Set();
        }
        #endregion Private Operations
        #region Public Properties

/// Список порядкових номерів наявних пакетів

        public int[] VolList
        {
            get
```

```
            {
                if (!InProcessing)
                {
                    return this.volList;
                } else
                {
                    return null;
                }
            }
        }

        #endregion Public Properties
    }
}
```

Нижче наведемо ту частину коду, яка виконує вищеперераховані дії.

```
namespace R_Disk
{

/// Клас кодера циклічного коду
    public class CicleCodeEncoder : CicleCodeBase
    {
        #region Construction & Destruction

/// Конструктор кодера за замовчуванням
        public CicleCodeEncoder()
        {
// Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }

/// Базовий конструктор кодера
/// <param name="dataCount">Кількість основних пакетів</param>
/// <param name="eccCount">Кількість пакетів для відновлення</param>
        public CicleCodeEncoder(int dataCount, int eccCount)
        {
// Установка конфігурації кодера
            SetConfig(dataCount, eccCount, (int)CicleCodeType.Alternative);
// Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }
```

```
/// Розширений конструктор кодера
/// <param name="dataCount">Кількість основних пакетів</param>
/// <param name="eccCount">Кількість пакетів для відновлення</param>
/// <param name="codecType">Тип кодера циклічного коду (по типу матриці)</param>
        public CicleCodeEncoder(int dataCount, int eccCount, int codecType)
        {
// Установка конфігурації кодера
            SetConfig(dataCount, eccCount, codecType);
// Створюємо об'єкт класу роботи з елементами поля Галуа
            this.eGF16 = new GF16();
        }
        #endregion Construction & Destruction
        #region Public Operations

/// Установка конфігурації кодера
/// <param name="dataCount">Кількість основних пакетів</param>
/// <param name="eccCount">Кількість пакетів для відновлення</param>
/// <param name="codecType">Тип кодека кодера циклічного коду (по типу
матриці)</param>
/// <returns>Булевський прапор операції установки конфігурації</returns>
        public bool SetConfig(int dataCount, int eccCount, int codecType)
        {
            int maxVolCount;
// Установлюємо константи, що відповідають обраному режиму
            if (codecType == (int)CicleCodeType.Dispersal)
            {
                maxVolCount = (int)CicleCodeConst.MaxVolCountDisp;
            } else
            {
                maxVolCount = (int)CicleCodeConst.MaxVolCountAlt;
            }
// Перевіряємо конфігурацію на коректність
            if (
                    (dataCount > 0)
                 &&
                    (eccCount > 0)
                 &&
                    ((dataCount + eccCount) <= maxVolCount)
                )
            {
// Якщо основна конфігурація змінилася - сповіщаємо про це
                if (
```

```
                        (dataCount != this.n)
                     ||
                        (eccCount != this.m)
                     ||
                        (codecType != this.eCicleCodeType)
                    )
                {
                    this.mainConfigChanged = true;
                }
// Зберігаємо конфігурацію
                this.n = dataCount;
                this.m = eccCount;
                this.eCicleCodeType = codecType;
// Також перераховуємо кількість ітерацій всіх стадій підготовки
                double n = this.n;
                double m = this.m;
// Нормалізуємо значення для розрахунку, щоб уникнути переповнення змінних
                NormalizeNM(ref n, ref m);
// Кількість ітерацій на першій стадії залежить від типу використовуваної матриці
                if (this.eCicleCodeType == (int)CicleCodeType.Alternative)
                {
                    this.iterOfFirstStage = m;
                } else
                {
                    this.iterOfFirstStage = ((n * m) * n) + (n * ((n + m) + (n *
(n + m))));
                }
                this.iterOfSecondStage = 0; // У кодери немає інвертування матриці

                this.configIsOK = true;
            } else
            {
//...указуємо на помилку конфігурації
                this.configIsOK = false;
            }
            return this.configIsOK;
        }
/// Метод множення матриці кодування на вхідний прологарифмований вектор
/// <param name="dataLog">Прологарифмований вхідний вектор (вихідні дані)</param>
/// <param name="ecc">Вихідний вектор (надлишкові дані)</param>
/// <returns>Булевський прапор результату операції</returns>
        public bool Process(int[] dataLog, ref int[] ecc)
```

```
        {
// Якщо кодер зконфігуровано некоректно, обробка неможлива!
            if (!this.configIsOK)
            {
                return false;
            }
// Копіюємо покажчик на масив експонент для скорочення часу обігу
            int[] GF16Exp = this.eGF16.GFExpTable;
// Обчислення результату множення матриці на вектор
            for (int i = 0; i < this.m; i++)
            {
                int mulSum = 0; // Сума добутку рядка матриці на стовпець
                int i_n = i * this.n;   // Зсув у масиві до елементів i-ой рядка
                for (int j = 0; j < this.n; j++)
                {
                    mulSum ^= GF16Exp[this.FLog[i_n + j] + dataLog[j]];
                }
                ecc[i] = mulSum;
            }
            return true;
        }
        #endregion Public Operations
        #region Private Operations

/// Заповнення матриці Вандермонда даними
        protected override void FillFLog()
        {
// Якщо основна конфігурація змінилася...
            if (this.mainConfigChanged)
            {
                if (this.eCicleCodeType == (int)CicleCodeType.Dispersal)
                {
//...робимо формування дисперсної матриці "D"
                    if (!MakeDispersalMatrix())
                    {
// Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;
// Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;
// Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();
                        return;
```

```
                    }
                } else
                {
//...робимо формування альтернативного заповнення матриці "A"
                    if (!MakeAlternativeMatrix())
                    {
// Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;
// Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;
// Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();
                        return;
                    }
                }
// Виділяємо пам'ять під матрицю "FLog"
                this.FLog = new int[this.m * this.n];
// Заповнюємо матрицю кодування
                for (int i = 0; i < this.m; i++)
                {
// Зсув у масиві до елементів i-ой рядка
                    int i_n = i * this.n;
// Залежно від типу кодера беремо дані з відповідного масиву
                    if (this.eCicleCodeType == (int)CicleCodeType.Dispersal)
                    {
// Формування рядка в матриці кодування
                        for (int j = 0; j < this.n; j++)
                        {
    // У матрицю кодування поміщаємо логарифми її вихідних елементів
    // (для прискорення множення матриці на вектор)
                            this.FLog[i_n + j] = this.eGF16.Log(this.D[((this.n +
i) * this.n) + j]);
                        }
                    } else
                    {
// Формування рядка в матриці кодування
                        for (int j = 0; j < this.n; j++)
                        {
                            int idx = i_n + j;
    // У матрицю кодування поміщаємо логарифми її вихідних елементів
    // (для прискорення множення матриці на вектор)
                            this.FLog[idx] = this.eGF16.Log(this.A[idx]);
```

```
                        }
                    }
// У випадку, якщо потрібна постановка на паузу, подію "executeEvent"
// буде скинуто, і будемо на паузі аж до його появи
                    ManualResetEvent.WaitAll(this.executeEvent);
// Якщо зазначено, що потрібно вийти з потоку - виходимо
                    if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                    {
// Указуємо, що кодер зконфігуровано некоректно
                        this.configIsOK = false;
// Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;
// Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();
                        return;
                    }
                }
// Якщо є передплата на делегата завершення...
                if (OnCicleCodeMatrixFormingFinish != null)
                {
//...повідомляємо, що екземпляр класу готовий до роботи
                    OnCicleCodeMatrixFormingFinish();
                }

//...і скидаємо прапор
                this.mainConfigChanged = false;
            }

// Якщо є передплата на делегата завершення...
            if (OnCicleCodeMatrixFormingFinish != null)
            {
//...повідомляємо, що екземпляр класу готовий до роботи
                OnCicleCodeMatrixFormingFinish();
            }
// Активуємо індикатор актуального стану змінних-членів
            this.finished = true;
// Установлюємо подію завершення обробки
            this.finishedEvent[0].Set();
        }
        #endregion Private Operations
    }
}
```

4.2 Захист розробленого програмного забезпечення

Дані які використовуються у даній роботі захищаються алгоритмом ДСТУ 7624:2014 («Калина»). При розробці національного стандарту ДСТУ 7624:2014 (блоковий шифр «Калина» і режими його роботи) було ухвалене рішення забезпечити прозорість проектування й використовувати консервативний підхід із застосуванням добре досліджених конструкцій, що забезпечують запас стійкості для безпечного застосування алгоритму в умовах істотного прогресу криптоаналітичних технік і засобів обробки даних.

Національний стандарт підтримує розмір блоку й довжину ключа шифрування 128, 256 і 512 біт (довжина ключа дорівнює розміру блоку або у два рази перевищує його), забезпечуючи нормальний, високий і надвисокий рівень стійкості (зараз це єдиний у світі стандарт блокового шифрування, що підтримує 512-бітові симетричні ключі). Різні варіанти забезпечують гнучкість вибору параметрів для розроблювачів систем криптографічного захисту, що дозволяє одержати як найвищий рівень швидкодії, так і найбільший запас стійкості перетворення.

Високорівнева конструкція використовує добре досліджену Square-подібну SPN-структуру, застосовувану в алгоритмах AES/Rijndael, Whirlpool, «Стрибог», «Коник» і багатьох інших. Циклове перетворення побудоване на базі таблиць підстановки (S-блоків) і множення на МДР-матрицю над кінцевим полем, забезпечуючи необхідні криптографічні властивості. Застосування саме такої конструкції дозволяє забезпечити доказову стійкість до диференціального, лінійному й ін. видам криптоаналізу, одночасно забезпечуючи ефективну реалізацію на широкому спектрі програмних і програмно-апаратних платформ. При виборі розміру МДР-матриці був прийнятий в увагу розмір кешу L1 сучасних і перспективних процесорів, що дозволило оптимізувати швидкодія програмної реалізації шифру.

Альтернативний варіант, який розглядався при розробці циклової функції, – ARX перетворення (Addition-rotation-xor). Цей підхід реалізований у шифрі SPECK (розроблений Агентством національної безпеки США й переданий у відкритий доступ), у блокових алгоритмах, на основі яких побудовано сімейство геш-функцій SHA-0,1,2 і ін. Перевагою походу є компактність і швидкодія перетворення. Але, у той же час, є й істотний недолік, пов'язаний з відсутністю методів, що дозволяють виконати строге аналітичне обґрунтування криптографічної стійкості таких розв'язків. Навіть із наймогутнішими у світі можливостями для аналізу, США кілька раз були змушено модифікувати свої стандарти гешування через знайдені уразливості: з 1993 по 1995 рр. діяв SHA-0, з 1995 по 2001 рр. застосовувався SHA-1, з тих пор використовується SHA-2. Через питання, що піднімаються, до стійкості й цієї версії, у США з 2008 по 2012 рр. був проведений міжнародний конкурс SHA-3. До теперішнього часу розроблений проект стандарту FIPS 202, що описує нову криптографічну геш-функцію.

Таким чином, консервативний і прозорий підхід до проектування нового національного стандарту України обумовив вибір добре перевіреної конструкції на базі S-блоків і лінійного перетворення, для якої можливо забезпечити доказову стійкість до різних видів криптоаналізу.

При порівнянні характеристик S-блоків і інших симетричних перетворень, можна відзначити, що саме національний стандарт України забезпечує найбільшу нелінійність булевих функцій, що дає додатковий запас стійкості до лінійного криптоаналізу. Ще більше значення нелінійності для взаємо- однозначної підстановки можна забезпечити, застосовуючи, наприклад, аффінно-тквівалентні статечні функції в кінцевому полі, але такі перетворення, використані в AES, Camellia і ін. алгоритмах, ставлять шифр під погрозу реалізації алгебраїчної атаки (цей метод криптоаналізу був успішно застосований проти шифру Keeloq, використовуваного в системах автомобільної безпеки).

У якості схеми розгортання ключів була запропонована нова конструкція з наступними властивостями:

– забезпечення криптографічної стійкості до відомих методів аналізу, відсутність «слабких» ключів, які можуть погіршити властивості перетворення;

– зручність програмної й програмно-апаратної реалізації (для формування циклових ключів застосовуються тільки операції, використовувані при шифруванні);

– висока обчислювальна складність відновлення ключа шифрування по одному або декільком цикловим ключам.

Остання властивість забезпечує додатковий захист до атак на реалізацію, коли зловмисник намагається атакувати інженерні розв'язки (вимірюючи споживаний пристроєм струм, навмисне викликаючи збої в роботі через навмисний перегрів шифратора та ін.). Ця особливість є істотною перевагою для ряду додатків, зокрема, при реалізації шифрування на смарт-картах, USB-токенах та ін., коли ключ прошитий у пристрої й повинен бути захищений від зовнішнього доступу (наприклад, у модулях доступу до платних цифрових ТБ-каналівм і ін.)

Кількість циклів шифрування залежить від довжини ключа: 10 циклів для 128-бітового, 14 циклів для 256-бітового й 18 циклів для 512-бітового ключа шифрування.

У порівнянні з іншими алгоритмами на основі Square-подібної SPN-структури, блоковий шифр «Калина» має наступні істотні конструктивні відмінності:

– початкове й кінцеве забілювання з використанням модульного додавання (264) для підвищення складності криптоаналітичних атак;

– застосування чотирьох різні S-блоків (замість одного) із властивостями для захисту від алгебраїчних атак, і при порівнянні характеристик з іншими блоковими й потоковими шифрами забезпечують найбільшу нелінійність булевих функцій (104), що дає додатковий запас стійкості перетворення;

– збільшений розмір МДР-перетворення, що поліпшує криптографічні властивості й дозволяє оптимізувати швидкодія на сучасних 64-бітових платформах;

– нову односпрямовану схему формування циклових ключів, що забезпечує захист від атак, ефективність програмної й програмно-апаратної реалізації, разом з додатковою стійкістю до методів аналізу спеціального виду.

Оцінка криптографічної стійкості до диференціального, лінійному, алгебраїчному, інтегральному й іншим методам аналізу (практичний критерій) показала, що шифр є стійким при 6 циклах для 128-бітового блоку, 7 циклах для 256-бітового й 9 циклах для 512-бітового (кожний додатковий цикл забезпечує експонентний ріст складності криптоаналізу). Таким чином, шифр, що містить 10, 14 і 18 циклів для блоку 128, 256 і 512 біт відповідно, забезпечує захист від розглянутих видів аналізу й має істотний запас стійкості.

Крім блокового шифру, ДСТУ 7624:2014 визначає режими роботи, що відповідають як ISO 10116:2006, так і додаткові, призначені для сучасних систем криптографічного захисту IP-трафіка, прозорого шифрування носіїв інформації й ін. У стандарті визначені обсяги повідомлень, після обробки яких потрібна обов'язкова зміна ключа. Крім того, приводяться рекомендації розроблювачам, що обертають увагу на необхідність запобігання атак з використанням особливостей реалізації засобів шифрування. Зокрема, враховані особливості, що дозволяли організацію атак BEAST і CRIME/BREACH у протоколах SSL/TLS, повторне приймання повідомлення, відновлення конфіденційних параметрів на основі залежності часу шифрування від оброблюваних даних (через промахи кешу при табличній реалізації) і ін.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ КІБЕРБЕЗПЕКИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розглянемо розроблене ПЗ яке зображено на рисунку 5.1. Розроблене програмне забезпечення призначене для перешкодостійкого кодування для відеоконференцзв'язку у бездротових мережах.

З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні розділи:

– Файл.

– Інструменти.

– Параметри.

– Довідка.

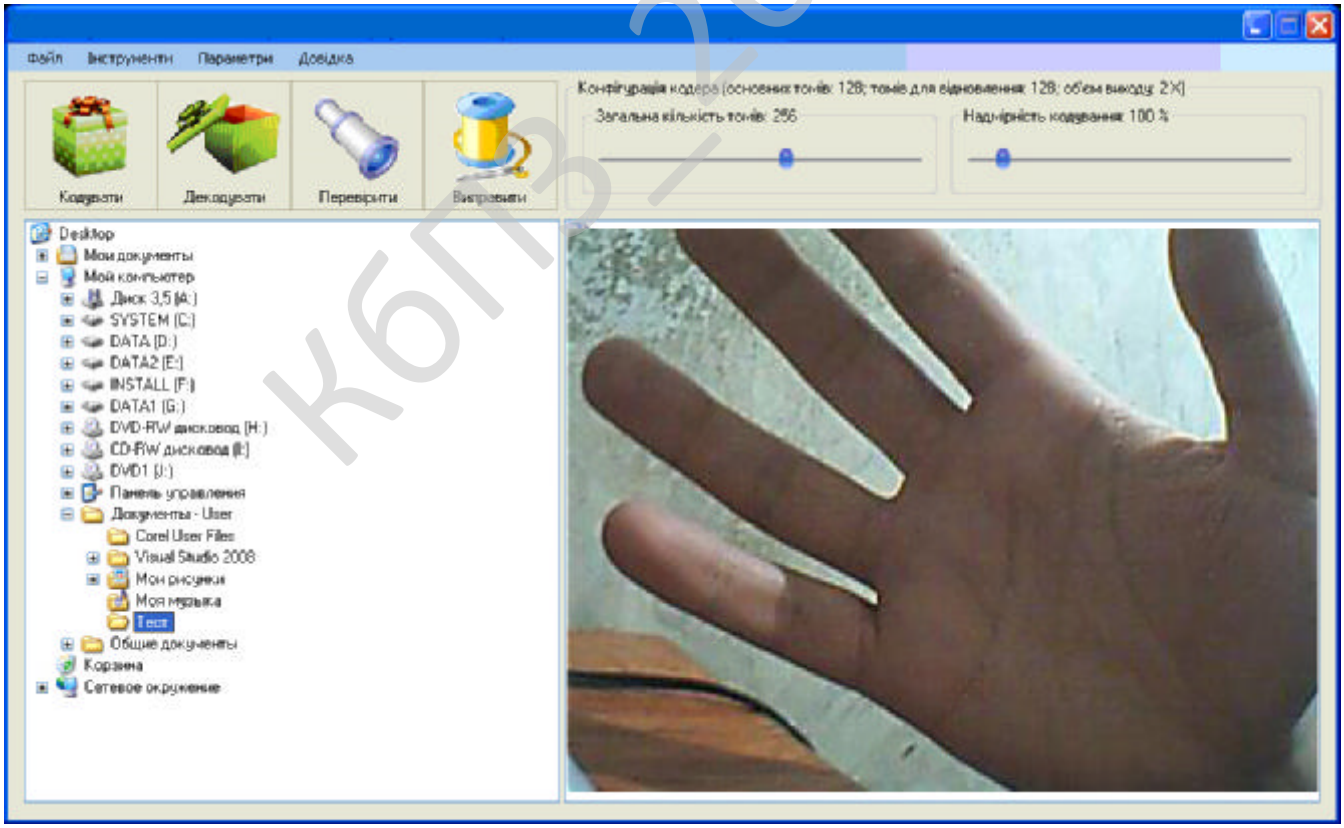

Рисунок 5.1 – Основне вікно ПЗ

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення.

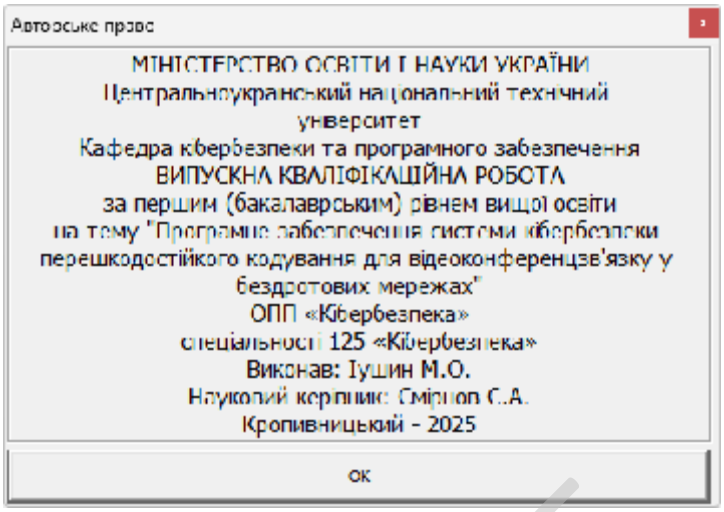

Рисунок 5.2 – Авторське право

Обрано умови розповсюдження – proprietary software. Програмне забезпечення, на яке зберігаються як немайнові, так і майнові авторські права. Отримавши або придбавши таке програмне забезпечення, користувач отримує обмежені права користування ним: може бути заборонено або закрито доступ до коду (вивчення), внесення змін, тиражування, розповсюдження та перепродаж. Програмне забезпечення вважається власницьким, якщо наявне хоча б одне з перелічених обмежень. Найчастіше основним методом захисту майнових прав на власницьке ПЗ, поза ліцензійною угодою, власник обирає закриття сирцевого коду, захищаючи свій продукт від модифікації і вбудовуючи системи обмеження користування через авторизацію. Таке програмне забезпечення називається закритим. Проте, код власницького продукту може бути і відкритим, але власник може обмежити права користувача умовами користувацької ліцензії. Власницьке програмне забезпечення та комерційне програмне забезпечення не є синонімами – власницьким може бути і безплатне (тобто, некомерційне) програмне забезпечення.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи кібербезпеки перешкодостійкого кодування для відеоконференцзв'язку у бездротових мережах.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем перешкодостійкого кодування для відеоконференцзв'язку у бездротових мережах.

– Досліджена система перешкодостійкого кодування для відеоконференцзв'язку у бездротових мережах.

– На основі отриманих результатів досліджень створена програмна реалізація системи кібербезпеки перешкодостійкого кодування для відеоконференцзв'язку у бездротових мережах.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання перешкодостійкого кодування для відеоконференцзв'язку у бездротових мережах.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об'єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Visual C#. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для

системи кібербезпеки перешкодостійкого кодування для відеоконференцзв'язку у бездротових мережах. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи кібербезпеки й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм ДСТУ 7624:2014.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms. The MIT Press. 2022 1677 p.

2. Will Grant. 101 UX Principles. Packt Publishing. 2022. 432 p.

3. Nathan Metzler. Kotlin Programming for Beginners. Independently published. 2021. 158 p.

4. Henry Lloyd. Interactive Computer Graphics. States Academic Press. 2022. 247 p.

5. Ranjan Parekh. Fundamentals of Image, Audio, and Video Processing Using MATLAB®With Applications to Pattern Recognition. CRC Press. 2021. 406 p.

6. Alasdair McAndrew. A Computational Introduction to Digital Image Processing. Chapman & Hall. 2021. 560 p.

7. Peter Shirley, Steve Marschner. Fundamentals of Computer Graphics. 2009

8. Михайло Пічугін, Іван Канкін, Володимир Воротніков Комп'ютерна графіка. Навчальний посібник / Центр навчальної літератури 346 с. 2019р.

9. Маценко В.Г. Комп’ютерна графіка: Навчальний посібник. – Чернівці: Рута, 2009 – 343 с.

10. Інженерна комп’ютерна графіка: підручник / В.В. Проців [та ін.] / М-во освіти і науки України, Нац. гірн. унт-т. – Дніпро: НГУ, 2017. – 247 с.

11. Проців В.В. Прикладна комп'ютерна графіка [Текст]: Навч. посібник / В.В. Проців, К.А. Зіборов, К.М. Бас, Г.К. Ванжа; М-во освіти і наук, Нац. гірн. ун-т. - Д.: НГУ, 2016. - 187 с.

12. Kopf, Johannes and Lischinski, Dani. Depixelizing Pixel Art (англ.) // ACM Trans. Graph. – 2011. – Vol. 30, no. 4. – P. 99:1--99:8.

13. Giachetti, Andrea and Asuni, Nicola. Real-Time Artifact-Free Image Upscaling (англ.) // Trans. Img. Proc.. – 2011. – Vol. 20, no. 10. – P. 2760—2768.

14. Kuznetsov, O., Frontoni, E., Kryvinska, N., Chevardin, V., Smirnov, O. «Wireless Network Encryption Stream Ciphers, Computational Modeling, and Security Analysis». *Computational Modeling and Simulation of Advanced Wireless Communication Systems*, 2024, pp. 379–402.

15. Kuznetsov, O., Frontoni, E., Kryvinska, N., Smirnov, O., Imoize, G.L. «Computational Modeling of Enhanced Spread Spectrum Codes for Asynchronous Wireless Communication». *Computational Modeling and Simulation of Advanced Wireless Communication Systems*, 2024, pp. 403–447

16. Al-Mudhafar Aqeel, A.M., Smirnova, T., Buravchenko, K., Smirnov, O. «The method of assessing and improving the user experience of subscribers in software-configured networks based on the use of machine learning». *Advanced Information Systems*, 2023, 7(2), pp. 49-56.

17. Smirnov, O., Sydorenko, V., Aleksander, M., Zhyharevych, O., Yenchev, S. «Simulation of the cloud IoT-based monitoring system for critical infrastructures». *CEUR Workshop Proceedings,* Volume 3530, 2023, pp. 256-265.

18. Smirnov, O., Odarchenko, R., Smirnova, T., Bondar, S., Volosheniuk, D. «Optimal Structure Construction of Private 5G Network for the Needs of Enterprises». Lecture Notes on Data Engineering and Communications Technologies, 2023, 178, pp. 208–223.

19. Smirnov, O., Neskorodieva, T., Fedorov, E., Rudakov, K., Neskorodieva, A. «Method Detection Audit Data Anomalies on Basis Restricted Cauchy Machine» CEUR Workshop Proceedings, Volume 3187, 2022,

20. Smirnov O., Smirnova T., Anas M. Al-Oraiqat, Drieiev O., Polishchuk L., Sheroz Khan, Yassin M. Y. Hasan, Aladdein M. Amro, Hazim S. AlRawashdeh «Method for Determining Treated Metal Surface Quality Using Computer Vision Technology». Sensors (Basel, Switzerland) Volume 22, Issue 16, 6223, 2022.

21. Smirnov O., Kuznetsov A., Kryvinska N., Kiian A., Kuznetsova K. «Full Non-Binary Constant-Weight Codes». SN Computer Science, Vol 2, 337, 2021. https://doi.org/10.1007/s42979-021-00739-w

22. Smirnov O., Kuznetsov A., Zhora V., Onikiychuk A., Pieshkova O. «Hiding Messages in Audio Files Using Direct Spread Spectrum». 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2021, Cracow, Poland, 22-25 September 2021. P. 414-418.

23. Smirnov O., Kuznetsov A., Lokotkova I., Kuznetsova T., Florov S., Lebid O. «Using Orthogonal Signals to Hide Information in Images». 4 IEEE International Conference on Advanced Information and Communication Technologies (AICT) - 2021, Lviv, Ukraine, September 21-25, 2021. P. 255-260.

24. Smirnov, O., Kuznetsov, A., Potii, O., Poluyanenko, N., Stelnyk, I., Mialkovsky, D. «Combining and filtering functions in the framework of nonlinear-feedback shift register». International Journal of Computing; 2020, Volume 19, Issue 2 – Research Institute for Intelligent Computer Systems – 2020. – P. 247-256.

25. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». CEUR Workshop Proceedings. Volume 2740, 2020, Pages 102-114.

26. Smirnov O., Alimseitova Zh., Adranova A., Akhmetov B., Lakhno V., Zhilkishbayeva G. «Models and algorithms for ensuring functional stability and cybersecurity of virtual cloud resources». Journal of theoretical and applied information technology Vol.98. No 21, 2020, P. 3334-3346.

27. Smirnov O., Kuznetsov A., Kovalchuk D., Kuznetsova T. «New technique for data hiding in cover images using adaptively generated pseudorandom sequences». CEUR Workshop Proceedings Volume 2654, 2020, Pages 1-14.

28. Smirnov O., Kuznetsov A., Onikiychuk A., Makushenko T., Anisimova O., Arischenko A. «Adaptive pseudo-random sequence generation for spread spectrum image steganography». 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT), Ukraine, Kyiv, May 14-18. 2020. P. 161-165.

29. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT), Ukraine, Kyiv, May 14-18. 2020. P. 172-177.

30. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In: Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. Springer, Cham. 2021. pp 557-587.

31. Smirnov, O., Drieieva, H., Drieiev, O., Polishchuk, Y., Brzhanov, R., Aleksander, M. «Method of fractal traffic generation by a model of generator on the graph». CEUR Workshop Proceedings Volume 2616, 2020, Pages 366-379.

32. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», CEUR Workshop Proceedings Volume 2608, 2020, Pages 633-645.

33. Smirnov, O., Kuznetsov, A., Gorbacheva, L., Babenko, V., «Hiding data in images using a pseudo-random sequence», CEUR Workshop Proceedings Volume 2608, 2020, Pages 646-660.

34. Zhurakovskyi, B., Tsopa, N., Batrak, Y., Odarchenko, R., Smirnova, T «Comparative analysis of modern formats of lossy audio compression». Workshop Proceedings, 2020, 2654, стр. 315-327.

35. Smirnov O. Kuznetsov A., Zaicenko Yu., Pastukhov M., Oleshko O., Kuznetsova K., «Formation of Discrete Signals with Special Correlation Properties». International Conference on Information and Telecommunication Technologies and Radio Electronics, UkrMiCo 2019; Odessa; Ukraine; 9-13 September 2019. P.22-28.

36. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019,

Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

37. Smirnov, O., Kuznetsov, A., Reshetniak, O., Ivko, N., Katkova, T., Kuznetsova, T., «Generators of Pseudorandom Sequence with Multilevel Function of Correlation». 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T), Kyiv, Ukraine, 8 – 11 October 2019 . P.517-522.

38. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». CEUR Workshop Proceedings, Vol 2588, P. 90-106, 2019.

39. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Averchev, A., Pastukhov, M., Kuznetsova, K., «Formation of Pseudorandom Sequences with Special Correlation Properties», 2019 3rd International Conference on Advanced Information and Communications Technologies, AICT -2019/ Lviv, Ukraine, 2-6 July, 2019, P. 395-399.

40. Smirnov, O., Kuznetsov, A., Kavun, S., Babenko, B., Nakisko, O., Kuznetsova, K., «Malware Correlation Monitoring in Computer Networks of Promising Smart Grids», 2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS), Kyiv, Ukraine April 17-19, 2019 P. 347-352.

41. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special Correlation Properties», CEUR Workshop Proceedings Volume 2353, CEUR Workshop Proceedings 2019, Pages 618-629.

42. Smirnov A.A., Kuznetsov A.A., Danilenko D.A., Berezovsky A., «The statistical analysis of a network traffic for the intrusion detection and prevention systems», Telecommunications and Radio Engineering. – Volume 74, Issue 1. – Begel House Inc. – 2015. – P. 61-78.

43. Smirnov O., Kuznetsov A., Kovalchuk D., Kuznetsova T. «New Technique for Hiding Data in Cover Images Using Adaptively Generated

Pseudorandom Sequences». CEUR Workshop Proceedings Volume 2732, 2020, Pages 214-227.

44. Смірнова Т.В., Конопліцька-Слободенюк О.К., Буравченко К.О., Смірнов С.А., Кравчук О.В., Козірова Н.Л., Смірнов О.А. «Дослідження технологій забезпечення кібербезпеки хмарних сервісів IaaS, PaaS та SaaS». *Кібербезпека: освіта, наука, техніка.* 2024. №4(24), С. 6-27.

45. Батрак О., Смірнова Т., Гнатюк В., Одарченко Р., Смірнов О. «Дослідження показників ефективності функціонування та перспектив розвитку систем IP-телефонії». *Підводні технології,* 2024, № 13, с. 28-35.

46. Аль-Мудхафар Акіл Абдулхуссейн М., Смірнова Т.В., Буравченко К.О., Смірнов О.А. «Метод оцінки та підвищення користувальницького досвіду абонентів в програмно-конфігурованих мережах на основі використання машинного навчання». *Сучасні інформаційні системи,* 2023, том 7, № 2, С. 49-56.

47. Т.В. Смірнова, О.М. Дрєєв, О.А. Смірнов «Хмарна інформаційна система оцінювання шорсткості з використанням дискретного частотного аналізу макрофотографій». IV міжнародна науково-практична конференція “Інформаційна безпека та комп’ютерні технології”, м. Кропивницький. 15-16 квітня 2021р. – Кропивницький: ЦНТУ. – 2021. – С. 30.

48. О.А. Смірнов, П.С. Усік, «Дослідження перспектив використання технологічних рішень в мережах 5G» у Кібербезпека та інформаційні технології: монографія. – Х. : ТОВ «ДІСА ПЛЮС», 2020.С. 122-135.

49. О.А.Смірнов, Т.В.Смірнова, Л.І. Поліщук, К.О. Буравченко, А.О.Макевнін, «Дослідження хмарних технологій як сервісів», Кібербезпека: освіта, наука, техніка. № 3(7). С. 43-62. 2020.

50. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., Смірнова Т.В. «Фрактальний аналіз генератора самоподібного трафіку на основі ланцюга Маркова». Центральноукраїнський науковий вісник. Технічні науки. № 2(33). с. 161-172, 2019.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи кібербезпеки перешкодостійкого кодування для відеоконференцзв’язку у бездротових мережах.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 57-02 від 17.01.2025 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи кібербезпеки перешкодостійкого кодування для відеоконференцзв’язку у бездротових мережах.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи кібербезпеки з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи кібербезпеки перешкодостійкого кодування для відеоконференцзв’язку у бездротових мережах;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпечення.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище Visual C#.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Перелік документів, що розробляються

- Структурна схема системи – 1 аркуш.
- Функціональна схема системи – 1 аркуш.
- Діаграма процесів – 1 аркуш.
- Блок-схема алгоритму роботи програми – 2 аркуша.
- Пояснювальна записка – 88 аркушів.

8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2025 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 5.06.2025 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

_____ Смірнов С.А.

Програмне забезпечення системи кібербезпеки перешкодостійкого кодування для відеоконференцзв’язку у бездротових мережах

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 56

Літера: РП

Кропивницький – 2025 року

Файл MainForm.cs - головне вікно програми

```
using System;
using System.Collections.Generic;
using System.Security.Cryptography;
using System.Text;
using System.Threading;

namespace SecureVideoConference
{
    class Program
    {
        static void Main(string[] args)
        {
            VideoConference vc = new VideoConference();
            vc.Initialize();
            vc.StartStreaming();
        }
    }

    class VideoConference
    {
        private EncryptionModule encryptionModule;
        private ErrorCorrectionModule errorCorrectionModule;
        private List<Participant> participants;
        private Firewall firewall;
        private BandwidthManager bandwidthManager;
        private AuthenticationModule authenticationModule;

        public void Initialize()
        {
            encryptionModule = new EncryptionModule();
            errorCorrectionModule = new ErrorCorrectionModule();
            firewall = new Firewall();
            bandwidthManager = new BandwidthManager();
            authenticationModule = new AuthenticationModule();
            participants = new List<Participant>();
            for (int i = 0; i < 10; i++)
            {
                participants.Add(new Participant(i));
            }
        }

        public void StartStreaming()
        {
            while (true)
            {
                foreach (var participant in participants)
                {
                    string data = participant.SendVideoData();
                    if (!authenticationModule.VerifyIdentity(participant))
continue;
                    string encryptedData = encryptionModule.EncryptData(data);
                    string correctedData =
errorCorrectionModule.ApplyErrorCorrection(encryptedData);
                    if (firewall.CheckTraffic(correctedData) &&
bandwidthManager.ValidateBandwidth(participant))
                    {
                        participant.ReceiveVideoData(correctedData);
                    }
                }
                Thread.Sleep(2000);
            }
        }
    }

    class EncryptionModule
```

```
{
    public string EncryptData(string data)
    {
        # Шифрує передані дані за допомогою SHA-256
        byte[] dataBytes = Encoding.UTF8.GetBytes(data);
        SHA256 sha256 = SHA256.Create();
        byte[] hash = sha256.ComputeHash(dataBytes);
        return Convert.ToBase64String(hash);
    }
}

class ErrorCorrectionModule
{
    public string ApplyErrorCorrection(string data)
    {
        # Додає перевірочний код для виправлення помилок
        return data + "CRC";
    }
}

class Firewall
{
    public bool CheckTraffic(string data)
    {
        # Перевіряє, чи є передача даних безпечною
        return !data.Contains("malware");
    }
}

class BandwidthManager
{
    public bool ValidateBandwidth(Participant participant)
    {
        # Контролює використання пропускної здатності для кожного учасника
        return true;
    }
}

class AuthenticationModule
{
    public bool VerifyIdentity(Participant participant)
    {
        # Перевіряє особу користувача перед передачею відеоданих
        return true;
    }
}

class Participant
{
    private int id;
    private Random random;
    private string sessionToken;

    public Participant(int id)
    {
        this.id = id;
        this.random = new Random();
        this.sessionToken = Guid.NewGuid().ToString();
    }

    public string SendVideoData()
    {
        # Генерує вихідні відеодані від учасника
        return "VideoDataFrom" + id;
    }

    public void ReceiveVideoData(string data)
    {
        # Отримує та обробляє відеодані після виправлення помилок
```

```
            if (data.EndsWith("CRC"))
            {
                ProcessVideoData(data.Substring(0, data.Length - 3));
            }
        }

        private void ProcessVideoData(string data)
        {
            # Імітація обробки отриманих відеоданих
        }
    }
}
```

Файл ProcessForm.cs - вікно відображення процесів запису/читання та перевірки цілісності даних

```
namespace RecoveryDisk
{
    partial class ProcessForm
    {
        /// <summary>
                /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true якщо керуючі ресурси повинні бути
розташовані, у іншому випадку  false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Необхідний метод для підтримки розробника - не модифікується
        /// зміст цього метода використовується редактором коду.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(ProcessForm));
            this.processPriorityGroupBox = new System.Windows.Forms.GroupBox();
            this.processPriorityComboBox = new System.Windows.Forms.ComboBox();
            this.processGroupBox = new System.Windows.Forms.GroupBox();
            this.processProgressBar = new System.Windows.Forms.ProgressBar();
            this.fileAnalyzeStatGroupBox = new System.Windows.Forms.GroupBox();
            this.percOfAltEccLabel = new System.Windows.Forms.Label();
            this.percOfDamageLabel = new System.Windows.Forms.Label();
            this.percOfAltEccLabel_ = new System.Windows.Forms.Label();
            this.percOfDamageLabel_ = new System.Windows.Forms.Label();
            this.logGroupBox = new System.Windows.Forms.GroupBox();
            this.logListBox = new System.Windows.Forms.ListBox();
            this.countGroupBox = new System.Windows.Forms.GroupBox();
            this.errorCountLabel = new System.Windows.Forms.Label();
            this.okCountLabel = new System.Windows.Forms.Label();
            this.errorPictureBox = new System.Windows.Forms.PictureBox();
            this.okPictureBox = new System.Windows.Forms.PictureBox();
            this.errorCountLabel_ = new System.Windows.Forms.Label();
            this.okCountLabel_ = new System.Windows.Forms.Label();
            this.toolTip = new System.Windows.Forms.ToolTip(this.components);
            this.stopButtonXP = new PinkieControls.ButtonXP();
            this.pauseButtonXP = new PinkieControls.ButtonXP();
            this.closingTimer = new System.Windows.Forms.Timer(this.components);
            this.processTimer = new System.Windows.Forms.Timer(this.components);
            this.processPriorityGroupBox.SuspendLayout();
            this.processGroupBox.SuspendLayout();
            this.fileAnalyzeStatGroupBox.SuspendLayout();
            this.logGroupBox.SuspendLayout();
            this.countGroupBox.SuspendLayout();

((System.ComponentModel.ISupportInitialize)(this.errorPictureBox)).BeginInit();
```

```
((System.ComponentModel.ISupportInitialize)(this.okPictureBox)).BeginInit();
            this.SuspendLayout();
            //
            // processPriorityGroupBox
            //

this.processPriorityGroupBox.Controls.Add(this.processPriorityComboBox);
            this.processPriorityGroupBox.FlatStyle =
System.Windows.Forms.FlatStyle.Flat;
            this.processPriorityGroupBox.Location = new
System.Drawing.Point(617, 216);
            this.processPriorityGroupBox.Name = "processPriorityGroupBox";
            this.processPriorityGroupBox.Size = new System.Drawing.Size(135,
64);
            this.processPriorityGroupBox.TabIndex = 0;
            this.processPriorityGroupBox.TabStop = false;
            this.processPriorityGroupBox.Text = "Пріоритет процесу";
            //
            // processPriorityComboBox
            //
            this.processPriorityComboBox.BackColor =
System.Drawing.SystemColors.Control;
            this.processPriorityComboBox.DropDownStyle =
System.Windows.Forms.ComboBoxStyle.DropDownList;
            this.processPriorityComboBox.FlatStyle =
System.Windows.Forms.FlatStyle.Flat;
            this.processPriorityComboBox.FormattingEnabled = true;
            this.processPriorityComboBox.Items.AddRange(new object[] {
            "За замовчуванням",
            "Знижений",
            "Нормальний",
            "Підвищений",
            "Найвищий"});
            this.processPriorityComboBox.Location = new System.Drawing.Point(9,
33);
            this.processPriorityComboBox.Name = "processPriorityComboBox";
            this.processPriorityComboBox.Size = new System.Drawing.Size(117,
21);
            this.processPriorityComboBox.TabIndex = 0;
            this.processPriorityComboBox.TabStop = false;
            this.toolTip.SetToolTip(this.processPriorityComboBox, "Список
можливих значень пріоритету процесу обробки");
            this.processPriorityComboBox.SelectedIndexChanged += new
System.EventHandler(this.processPriorityComboBox_SelectedIndexChanged);
            //
            // processGroupBox
            //
            this.processGroupBox.Controls.Add(this.processProgressBar);
            this.processGroupBox.FlatStyle =
System.Windows.Forms.FlatStyle.Flat;
            this.processGroupBox.Location = new System.Drawing.Point(12, 9);
            this.processGroupBox.Name = "processGroupBox";
            this.processGroupBox.Size = new System.Drawing.Size(871, 65);
            this.processGroupBox.TabIndex = 0;
            this.processGroupBox.TabStop = false;
            this.processGroupBox.Text = "Обробка";
            //
            // processProgressBar
            //
            this.processProgressBar.Location = new System.Drawing.Point(14, 30);
            this.processProgressBar.Name = "processProgressBar";
            this.processProgressBar.Size = new System.Drawing.Size(844, 20);
            this.processProgressBar.Style =
System.Windows.Forms.ProgressBarStyle.Continuous;
            this.processProgressBar.TabIndex = 0;
            //
            // fileAnalyzeStatGroupBox
            //
```

```
            this.fileAnalyzeStatGroupBox.Controls.Add(this.percOfAltEccLabel);
            this.fileAnalyzeStatGroupBox.Controls.Add(this.percOfDamageLabel);
            this.fileAnalyzeStatGroupBox.Controls.Add(this.percOfAltEccLabel_);
            this.fileAnalyzeStatGroupBox.Controls.Add(this.percOfDamageLabel_);
            this.fileAnalyzeStatGroupBox.FlatStyle =
System.Windows.Forms.FlatStyle.Flat;
            this.fileAnalyzeStatGroupBox.Location = new System.Drawing.Point(12,
216);
            this.fileAnalyzeStatGroupBox.Name = "fileAnalyzeStatGroupBox";
            this.fileAnalyzeStatGroupBox.Size = new System.Drawing.Size(459,
64);
            this.fileAnalyzeStatGroupBox.TabIndex = 0;
            this.fileAnalyzeStatGroupBox.TabStop = false;
            this.fileAnalyzeStatGroupBox.Text = "Результат аналізу цілісності
даних";
            //
            // percOfAltEccLabel
            //
            this.percOfAltEccLabel.AutoSize = true;
            this.percOfAltEccLabel.Location = new System.Drawing.Point(195, 41);
            this.percOfAltEccLabel.Name = "percOfAltEccLabel";
            this.percOfAltEccLabel.Size = new System.Drawing.Size(10, 13);
            this.percOfAltEccLabel.TabIndex = 0;
            this.percOfAltEccLabel.Text = "-";
            //
            // percOfDamageLabel
            //
            this.percOfDamageLabel.AutoSize = true;
            this.percOfDamageLabel.Location = new System.Drawing.Point(195, 20);
            this.percOfDamageLabel.Name = "percOfDamageLabel";
            this.percOfDamageLabel.Size = new System.Drawing.Size(10, 13);
            this.percOfDamageLabel.TabIndex = 0;
            this.percOfDamageLabel.Text = "-";
            //
            // percOfAltEccLabel_
            //
            this.percOfAltEccLabel_.AutoSize = true;
            this.percOfAltEccLabel_.Location = new System.Drawing.Point(7, 41);
            this.percOfAltEccLabel_.Name = "percOfAltEccLabel_";
            this.percOfAltEccLabel_.Size = new System.Drawing.Size(163, 13);
            this.percOfAltEccLabel_.TabIndex = 0;
            this.percOfAltEccLabel_.Text = "Резерв перевірочних даних для
відновлення:";
            //
            // percOfDamageLabel_
            //
            this.percOfDamageLabel_.AutoSize = true;
            this.percOfDamageLabel_.Location = new System.Drawing.Point(7, 20);
            this.percOfDamageLabel_.Name = "percOfDamageLabel_";
            this.percOfDamageLabel_.Size = new System.Drawing.Size(148, 13);
            this.percOfDamageLabel_.TabIndex = 0;
            this.percOfDamageLabel_.Text = "Всього пошкоджених секторів:";
            //
            // logGroupBox
            //
            this.logGroupBox.Controls.Add(this.logListBox);
            this.logGroupBox.Location = new System.Drawing.Point(12, 80);
            this.logGroupBox.Name = "logGroupBox";
            this.logGroupBox.Size = new System.Drawing.Size(871, 130);
            this.logGroupBox.TabIndex = 0;
            this.logGroupBox.TabStop = false;
            this.logGroupBox.Text = "Лог процесу";
            //
            // logListBox
            //
            this.logListBox.BackColor = System.Drawing.SystemColors.Control;
            this.logListBox.BorderStyle = System.Windows.Forms.BorderStyle.None;
            this.logListBox.FormattingEnabled = true;
            this.logListBox.HorizontalScrollbar = true;
```

```
            this.logListBox.Location = new System.Drawing.Point(7, 23);
            this.logListBox.Name = "logListBox";
            this.logListBox.SelectionMode =
System.Windows.Forms.SelectionMode.None;
            this.logListBox.Size = new System.Drawing.Size(851, 91);
            this.logListBox.TabIndex = 0;
            this.logListBox.TabStop = false;
            this.logListBox.UseTabStops = false;
            this.logListBox.SelectedIndexChanged += new
System.EventHandler(this.logListBox_SelectedIndexChanged);
            //
            // countGroupBox
            //
            this.countGroupBox.Controls.Add(this.errorCountLabel);
            this.countGroupBox.Controls.Add(this.okCountLabel);
            this.countGroupBox.Controls.Add(this.errorPictureBox);
            this.countGroupBox.Controls.Add(this.okPictureBox);
            this.countGroupBox.Controls.Add(this.errorCountLabel_);
            this.countGroupBox.Controls.Add(this.okCountLabel_);
            this.countGroupBox.Location = new System.Drawing.Point(482, 216);
            this.countGroupBox.Name = "countGroupBox";
            this.countGroupBox.Size = new System.Drawing.Size(124, 64);
            this.countGroupBox.TabIndex = 0;
            this.countGroupBox.TabStop = false;
            this.countGroupBox.Text = "Лічільник процесу";
            //
            // errorCountLabel
            //
            this.errorCountLabel.AutoSize = true;
            this.errorCountLabel.Location = new System.Drawing.Point(63, 41);
            this.errorCountLabel.Name = "errorCountLabel";
            this.errorCountLabel.Size = new System.Drawing.Size(13, 13);
            this.errorCountLabel.TabIndex = 0;
            this.errorCountLabel.Text = "0";
            this.toolTip.SetToolTip(this.errorCountLabel, "Лічильник некоректно
оброблених файлів");
            //
            // okCountLabel
            //
            this.okCountLabel.AutoSize = true;
            this.okCountLabel.Location = new System.Drawing.Point(63, 20);
            this.okCountLabel.Name = "okCountLabel";
            this.okCountLabel.Size = new System.Drawing.Size(13, 13);
            this.okCountLabel.TabIndex = 0;
            this.okCountLabel.Text = "0";
            this.toolTip.SetToolTip(this.okCountLabel, "Лічильник коректно
оброблених файлів");
            //
            // errorPictureBox
            //
            this.errorPictureBox.Image =
((System.Drawing.Image)(resources.GetObject("errorPictureBox.Image")));
            this.errorPictureBox.Location = new System.Drawing.Point(10, 40);
            this.errorPictureBox.Name = "errorPictureBox";
            this.errorPictureBox.Size = new System.Drawing.Size(21, 15);
            this.errorPictureBox.TabIndex = 2;
            this.errorPictureBox.TabStop = false;
            //
            // okPictureBox
            //
            this.okPictureBox.Image =
((System.Drawing.Image)(resources.GetObject("okPictureBox.Image")));
            this.okPictureBox.Location = new System.Drawing.Point(10, 19);
            this.okPictureBox.Name = "okPictureBox";
            this.okPictureBox.Size = new System.Drawing.Size(21, 15);
            this.okPictureBox.TabIndex = 1;
            this.okPictureBox.TabStop = false;
            //
            // errorCountLabel_
```

```
            //
            this.errorCountLabel_.AutoSize = true;
            this.errorCountLabel_.Location = new System.Drawing.Point(28, 41);
            this.errorCountLabel_.Name = "errorCountLabel_";
            this.errorCountLabel_.Size = new System.Drawing.Size(35, 13);
            this.errorCountLabel_.TabIndex = 0;
            this.errorCountLabel_.Text = "Error :";
            this.toolTip.SetToolTip(this.errorCountLabel_, "Лічильник некоректно
оброблених файлів");
            //
            // okCountLabel_
            //
            this.okCountLabel_.AutoSize = true;
            this.okCountLabel_.Location = new System.Drawing.Point(28, 20);
            this.okCountLabel_.Name = "okCountLabel_";
            this.okCountLabel_.Size = new System.Drawing.Size(28, 13);
            this.okCountLabel_.TabIndex = 0;
            this.okCountLabel_.Text = "OK :";
            this.toolTip.SetToolTip(this.okCountLabel_, "Лічильник коректно
оброблених файлів");
            //
            // toolTip
            //
            this.toolTip.AutomaticDelay = 2000;
            this.toolTip.AutoPopDelay = 20000;
            this.toolTip.InitialDelay = 2000;
            this.toolTip.ReshowDelay = 1000;
            //
            // stopButtonXP
            //
            this.stopButtonXP.BackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(0)))), ((int)(((byte)(236)))),
((int)(((byte)(233)))), ((int)(((byte)(216)))));
            this.stopButtonXP.DefaultScheme = true;
            this.stopButtonXP.DialogResult =
System.Windows.Forms.DialogResult.None;
            this.stopButtonXP.Hint = "";
            this.stopButtonXP.Location = new System.Drawing.Point(762, 257);
            this.stopButtonXP.Name = "stopButtonXP";
            this.stopButtonXP.Scheme = PinkieControls.ButtonXP.Schemes.Blue;
            this.stopButtonXP.Size = new System.Drawing.Size(121, 23);
            this.stopButtonXP.TabIndex = 2;
            this.stopButtonXP.Text = "Перервати обробку";
            this.toolTip.SetToolTip(this.stopButtonXP, "Припинення обробки
файлів із закриттям даного вікна");
            this.stopButtonXP.Click += new
System.EventHandler(this.stopButtonXP_Click);
            //
            // pauseButtonXP
            //
            this.pauseButtonXP.BackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(0)))), ((int)(((byte)(236)))),
((int)(((byte)(233)))), ((int)(((byte)(216)))));
            this.pauseButtonXP.DefaultScheme = true;
            this.pauseButtonXP.DialogResult =
System.Windows.Forms.DialogResult.None;
            this.pauseButtonXP.Hint = "";
            this.pauseButtonXP.Location = new System.Drawing.Point(762, 220);
            this.pauseButtonXP.Name = "pauseButtonXP";
            this.pauseButtonXP.Scheme = PinkieControls.ButtonXP.Schemes.Blue;
            this.pauseButtonXP.Size = new System.Drawing.Size(121, 23);
            this.pauseButtonXP.TabIndex = 1;
            this.pauseButtonXP.Text = "Пауза";
            this.toolTip.SetToolTip(this.pauseButtonXP, "Постановка/зняття
процесу обробки з паузи");
            this.pauseButtonXP.Click += new
System.EventHandler(this.pauseButtonXP_Click);
            //
            // closingTimer
```

```
            //
            this.closingTimer.Tick += new
System.EventHandler(this.closingTimer_Tick);
            //
            // processTimer
            //
            this.processTimer.Interval = 500;
            this.processTimer.Tick += new
System.EventHandler(this.processTimer_Tick);
            //
            // ProcessForm
            //
            this.AutoScaleDimensions = new System.Drawing.Size(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(894, 292);
            this.ControlBox = false;
            this.Controls.Add(this.stopButtonXP);
            this.Controls.Add(this.pauseButtonXP);
            this.Controls.Add(this.countGroupBox);
            this.Controls.Add(this.processPriorityGroupBox);
            this.Controls.Add(this.logGroupBox);
            this.Controls.Add(this.fileAnalyzeStatGroupBox);
            this.Controls.Add(this.processGroupBox);
            this.DoubleBuffered = true;
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedDialog;
            this.Icon =
((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
            this.MaximizeBox = false;
            this.MinimizeBox = false;
            this.Name = "ProcessForm";
            this.ShowInTaskbar = false;
            this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterScreen;
            this.Text = "Обробка файлу";
            this.Load += new System.EventHandler(this.ProcessForm_Load);
            this.processPriorityGroupBox.ResumeLayout(false);
            this.processGroupBox.ResumeLayout(false);
            this.fileAnalyzeStatGroupBox.ResumeLayout(false);
            this.fileAnalyzeStatGroupBox.PerformLayout();
            this.logGroupBox.ResumeLayout(false);
            this.countGroupBox.ResumeLayout(false);
            this.countGroupBox.PerformLayout();

((System.ComponentModel.ISupportInitialize)(this.errorPictureBox)).EndInit();

((System.ComponentModel.ISupportInitialize)(this.okPictureBox)).EndInit();
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.GroupBox processPriorityGroupBox;
        private System.Windows.Forms.GroupBox processGroupBox;
        private System.Windows.Forms.ProgressBar processProgressBar;
        private System.Windows.Forms.GroupBox fileAnalyzeStatGroupBox;
        private System.Windows.Forms.Label percOfDamageLabel_;
        private System.Windows.Forms.Label percOfAltEccLabel_;
        private System.Windows.Forms.GroupBox logGroupBox;
        private System.Windows.Forms.GroupBox countGroupBox;
        private System.Windows.Forms.Label errorCountLabel_;
        private System.Windows.Forms.Label okCountLabel_;
        private System.Windows.Forms.ListBox logListBox;
        private System.Windows.Forms.ComboBox processPriorityComboBox;
        private System.Windows.Forms.PictureBox errorPictureBox;
        private System.Windows.Forms.PictureBox okPictureBox;
        private System.Windows.Forms.Label errorCountLabel;
        private System.Windows.Forms.Label okCountLabel;
```

```
        private System.Windows.Forms.ToolTip toolTip;
        private System.Windows.Forms.Timer closingTimer;
        private System.Windows.Forms.Label percOfAltEccLabel;
        private System.Windows.Forms.Label percOfDamageLabel;
        private PinkieControls.ButtonXP pauseButtonXP;
        private PinkieControls.ButtonXP stopButtonXP;
        private System.Windows.Forms.Timer processTimer;
    }
}
```

Файл FileAnalyzer.cs - контроль цілісності даних

```
using System;
using System.Threading;
using System.IO;

namespace RecoveryDisk
{
    /// <summary>
    /// Клас контролю цілісності набору файлів-томів
    /// </summary>
    public class FileAnalyzer
    {
        #region Delegates

        /// <summary>
        /// Делегат відновлення прогресу контролю цілісності файлів
        /// </summary>
        public OnUpdateDoubleValueHandler OnUpdateFileAnalyzeProgress;

        /// <summary>
        /// Делегат завершення процесу контролю цілісності файлів
        /// </summary>
        public OnEventHandler OnFileAnalyzeFinish;

        /// <summary>
        /// Делегат одержання статистики ушкоджень багатотомного архіву
        /// </summary>
        public OnUpdateTwoDoubleValueHandler OnGetDamageStat;

        #endregion Delegates

        #region Public Properties & Data

        /// <summary>
        /// Булевська властивість "Файл обробляється?"
        /// </summary>
        public bool InProcessing
        {
            get
            {
                if (
                        (this.thrFileAnalyzer != null)
                     &&
                        (
                              (this.thrFileAnalyzer.ThreadState ==
ThreadState.Running)
                           ||
                              (this.thrFileAnalyzer.ThreadState ==
ThreadState.WaitSleepJoin)
                        )
                    )
                {
                    return true;

                } else
                {
                    return false;
                }
            }
        }

        /// <summary>
        /// Булевська властивість "Екземпляр класу закінчив обробку
        /// (має актуальний стан змінних-членів)?"
        /// </summary>
```

```
public bool Finished
{
    get
    {
        // Якщо клас не зайнятий обробкою - повертаємо значення
        if (!InProcessing)
        {
            return this.finished;

        } else
        {
            return false;
        }
    }
}

/// <summary>
/// Екземпляр класу повністю закінчив обробку?
/// </summary>
private bool finished;

/// <summary>
/// Булевська властивість "Безліч файлів оброблена коректно?"
/// </summary>
public bool ProcessedOK
{
    get
    {
        // Якщо клас не зайнятий обробкою - повертаємо значення
        if (!InProcessing)
        {
            return this.processedOK;

        } else
        {
            return false;
        }
    }
}

/// <summary>
/// Обробка набору файлів зроблена коректно?
/// </summary>
private bool processedOK;

/// <summary>
/// Список порядкових номерів наявних томів
/// </summary>
public int[] VolList
{
    get
    {
        if (!InProcessing)
        {
            return this.volList;

        } else
        {
            return null;
        }
    }
}

/// <summary>
/// Вектор, що вказує на состав томів
/// </summary>
private int[] volList;

/// <summary>
```

```
        /// Всі томи для відновлення коректні?
        /// </summary>
        public bool AllEccVolsOK
        {
            get
            {
                if (!InProcessing)
                {
                    return this.allEccVolsOK;

                } else
                {
                    return false;
                }
            }
        }

        /// <summary>
        /// Всі томи для відновлення коректні?
        /// </summary>
        private bool allEccVolsOK;

        /// <summary>
        /// Пріоритет процесу
        /// </summary>
        public int ThreadPriority
        {
            get
            {
                return (int)this.threadPriority;
            }

            set
            {
                if (
                        (this.thrFileAnalyzer != null)
                     &&
                        (this.thrFileAnalyzer.IsAlive)
                    )
                {
                    switch (value)
                    {
                        default:
                        case 0:
                        {
                            this.threadPriority =
System.Threading.ThreadPriority.Lowest;

                            break;
                        }

                        case 1:
                        {
                            this.threadPriority =
System.Threading.ThreadPriority.BelowNormal;

                            break;
                        }

                        case 2:
                        {
                            this.threadPriority =
System.Threading.ThreadPriority.Normal;

                            break;
                        }

                        case 3:
                        {
```

```
                            this.threadPriority =
System.Threading.ThreadPriority.AboveNormal;

                            break;
                        }

                        case 4:
                        {
                            this.threadPriority =
System.Threading.ThreadPriority.Highest;

                            break;
                        }
                    }

                    // Установлюємо обраний пріоритет процесу
                    this.thrFileAnalyzer.Priority = this.threadPriority;

                    // Дублюємо установку параметра для підконтрольного об'єкта
                    if (this.eFileIntegrityCheck != null)
                    {
                        this.eFileIntegrityCheck.ThreadPriority = value;
                    }
                }
            }
        }

        /// <summary>
        /// Пріоритет процесу контролю цілісності файлів
        /// </summary>
        private ThreadPriority threadPriority;

        /// <summary>
        /// Подія, установлювана по завершенню обробки
        /// </summary>
        public ManualResetEvent[] FinishedEvent
        {
            get
            {
                return this.finishedEvent;
            }
        }

        /// <summary>
        /// Подія, установлювана по завершенню обробки
        /// </summary>
        private ManualResetEvent[] finishedEvent;

        #endregion Public Properties & Data

        #region Data

        /// <summary>
        /// Модуль для впакування (розпакування) ім'я файлу в префіксний формат
        /// </summary>
        private FileNamer eFileNamer;

        /// <summary>
        /// Екземпляр класу контролю цілісності набору файлів
        /// </summary>
        private FileIntegrityCheck eFileIntegrityCheck;

        /// <summary>
        /// Шлях до файлів для обробки
        /// </summary>
        private String path;

        /// <summary>
        /// Ім'я файлу, якому належить безліч томів
```

```
        /// </summary>
        private String fileName;

        /// <summary>
        /// Кількість основних томів
        /// </summary>
        private int dataCount;

        /// <summary>
        /// Кількість томів для відновлення
        /// </summary>
        private int eccCount;

        /// <summary>
        /// Тип кодера циклічного коду (по типу використовуваної матриці
кодування)
        /// </summary>
        private int codecType;

        /// <summary>
        /// Використовується швидке добування з томів (без перевірки CRC-64)?
        /// </summary>
        private bool fastExtraction;

        /// <summary>
        /// Потік контролю цілісності файлу
        /// </summary>
        private Thread thrFileAnalyzer;

        /// <summary>
        /// Подія припинення обробки файлів
        /// </summary>
        private ManualResetEvent[] exitEvent;

        /// <summary>
        /// Подія продовження обробки файлів
        /// </summary>
        private ManualResetEvent[] executeEvent;

        /// <summary>
        /// Подія "пробудження" циклу очікування
        /// </summary>
        private ManualResetEvent[] wakeUpEvent;

        #endregion Data

        #region Construction & Destruction

        /// <summary>
        /// Конструктор класу перевірки цілісності набору файлів
        /// </summary>
        public FileAnalyzer()
        {
            // Модуль для впакування (розпакування) ім'я файлу в префіксний
формат
            this.eFileNamer = new FileNamer();

            // Створюємо екземпляр класу контролю цілісності набору файлів
            this.eFileIntegrityCheck = new FileIntegrityCheck();

            // Шлях до файлів для обробки за замовчуванням порожній
            this.path = "";

            // Ініціалізуємо ім'я файлу за замовчуванням
            this.fileName = "NONAME";

            // Спочатку всі томи для відновлення вважаємо ушкодженими
            this.allEccVolsOK = false;
```

```
            // Екземпляр класу повністю закінчив обробку?
            this.finished = true;

            // Обробка зроблена коректно?
            this.processedOK = false;

            // За замовчуванням встановлюється фоновий пріоритет
            this.threadPriority = 0;

            // Ініціалізуємо подію припинення обробки файлів
            this.exitEvent = new ManualResetEvent[] { new
ManualResetEvent(false) };

            // Ініціалізуємо подію продовження обробки файлів
            this.executeEvent = new ManualResetEvent[] { new
ManualResetEvent(false) };

            // Ініціалізуємо подію "пробудження" циклу очікування
            this.wakeUpEvent = new ManualResetEvent[] { new
ManualResetEvent(false) };

            // Подія, установлювана по завершенню обробки
            this.finishedEvent = new ManualResetEvent[] { new
ManualResetEvent(true) };
        }

        #endregion Construction & Destruction

        #region Public Operations

        /// <summary>
        /// Метод запуску потоку обробки обчислення й записи CRC64 у кінець
файлів
        /// </summary>
        /// <param name="path">Шлях до файлів для обробки</param>
        /// <param name="fileName">Ім'я файлу для обробки</param>
        /// <param name="dataCount">Конфігурація кількості основних
томів</param>
        /// <param name="eccCount">Конфігурація кількості томів для
відновлення</param>
        /// <param name="codecType">Тип кодера циклічного коду (по типу
матриці)</param>
        /// <param name="runAsSeparateThread">Запускати в окремому
потоці?</param>
        /// <returns>Булевський прапор операції</returns>
        public bool StartToWriteCRC64(String path, String fileName, int
dataCount, int eccCount, int codecType, bool runAsSeparateThread)
        {
            // Якщо потік обчислення CRC-64 працює - не дозволяємо повторний
запуск
            if (InProcessing)
            {
                return false;
            }

            // Скидаємо прапор коректності результату перед запуском потоку
            this.processedOK = false;

            // Скидаємо індикатор актуального стану змінних-членів
            this.finished = false;

            // Зберігаємо шлях до файлів для обробки
            if (path == null)
            {
                this.path = "";

            } else
            {
                // Робимо виділення шляху з "path" у випадку,
```

```
                // якщо туди було записано повне ім'я
                this.path = this.eFileNamer.GetPath(path);
            }

            if (fileName == null)
            {
                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

                // Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();

                return false;
            }

            // Робимо виділення короткого ім'я файлу з "fileName" у випадку,
            // якщо туди було записано повне ім'я
            this.fileName = this.eFileNamer.GetShortFileName(fileName);

            // Перевіряємо на некоректну конфігурацію
            if (
                    (dataCount <= 0)
                 ||
                    (eccCount <= 0)
                 ||
                    ((dataCount + eccCount) >
(int)CicleCodeConst.MaxVolCountAlt)
                )
            {
                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

                // Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();

                return false;
            }

            // Зберігаємо кількість основних томів
            this.dataCount = dataCount;

            // Зберігаємо кількість томів для відновлення
            this.eccCount = eccCount;

            // Зберігаємо тип кодера циклічного коду (по типу використовуваної
матриці кодування)
            this.codecType = codecType;

            // Указуємо, що потік повинен виконуватися
            this.exitEvent[0].Reset();
            this.executeEvent[0].Set();
            this.wakeUpEvent[0].Reset();
            this.finishedEvent[0].Reset();

            // Якщо зазначено, що не потрібен запуск в окремому потоці,
            // запускаємо в даному
            if (!runAsSeparateThread)
            {
                // Обчислюємо CRC-64 для кожного з файлів набору
                WriteCRC64();

                // Повертаємо результат обробки
                return this.processedOK;
            }

            // Створюємо потік обчислення й запису CRC-64...
            this.thrFileAnalyzer = new Thread(new ThreadStart(WriteCRC64));

            //...потім даємо йому ім'я...
```

```
            this.thrFileAnalyzer.Name = "FileAnalyzer.WriteCRC64()";

            //...установлюємо обраний пріоритет завдання...
            this.thrFileAnalyzer.Priority = this.threadPriority;

            //...і запускаємо його
            this.thrFileAnalyzer.Start();

            // Повідомляємо, що все нормально
            return true;
        }

        /// <summary>
        /// Метод запуску потоку обробки перевірки CRC64, записаного в кінець
        /// кожного з файлів набору, з генеруванням списку наявних томів
"volList",
        /// який буде використаний декодером для відновлення даних
        /// </summary>
        /// <param name="path">Шлях до файлів для обробки</param>
        /// <param name="fileName">Ім'я файлу для обробки</param>
        /// <param name="dataCount">Конфігурація кількості основних
томів</param>
        /// <param name="eccCount">Конфігурація кількості томів для
відновлення</param>
        /// <param name="codecType">Тип кодера циклічного коду (по типу
матриці)</param>
        /// <param name="fastExtraction">Використовується швидке добування з
томів (без перевірки CRC-64)?</param>
        /// <param name="runAsSeparateThread">Запускати в окремому
потоці?</param>
        /// <returns>Булевський прапор операції</returns>
        public bool StartToAnalyzeCRC64(String path, String fileName, int
dataCount, int eccCount, int codecType, bool fastExtraction, bool
runAsSeparateThread)
        {
            // Якщо потік обчислення CRC-64 працює - не дозволяємо повторний
запуск
            if (InProcessing)
            {
                return false;
            }

            // Спочатку всі томи для відновлення вважаємо ушкодженими
            this.allEccVolsOK = false;

            // Скидаємо прапор коректності результату перед запуском потоку
            this.processedOK = false;

            // Скидаємо індикатор актуального стану змінних-членів
            this.finished = false;

            // Зберігаємо шлях до файлів для обробки
            if (path == null)
            {
                this.path = "";

            } else
            {
                // Робимо виділення шляху з "path" у випадку,
                // якщо туди було записано повне ім'я
                this.path = this.eFileNamer.GetPath(path);
            }

            if (fileName == null)
            {
                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

                // Установлюємо подію завершення обробки
```

```
                this.finishedEvent[0].Set();

                return false;
            }

            // Робимо виділення короткого ім'я файлу з "fileName" у випадку,
            // якщо туди було записано повне ім'я
            this.fileName = this.eFileNamer.GetShortFileName(fileName);

            // Перевіряємо на некоректну конфігурацію
            if (
                    (dataCount <= 0)
                 ||
                    (eccCount <= 0)
                 ||
                    ((dataCount + eccCount) >
(int)CicleCodeConst.MaxVolCountAlt)
                )
            {
                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

                // Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();

                return false;
            }

            // Зберігаємо кількість основних томів
            this.dataCount = dataCount;

            // Зберігаємо кількість томів для відновлення
            this.eccCount = eccCount;

            // Зберігаємо тип кодека кодера циклічного коду (по типу
використовуваної матриці кодування)
            this.codecType = codecType;

            // Використовується швидке добування з томів (без перевірки CRC-64)?
            this.fastExtraction = fastExtraction;

            // Указуємо, що потік повинен виконуватися
            this.exitEvent[0].Reset();
            this.executeEvent[0].Set();
            this.wakeUpEvent[0].Reset();
            this.finishedEvent[0].Reset();

            // Якщо зазначено, що не потрібен запуск в окремому потоці,
            // запускаємо в даному
            if (!runAsSeparateThread)
            {
                // Обчислюємо й перевіряємо CRC-64 для кожного з файлів набору
із заповненням
                // властивості VolList
                AnalyzeCRC64();

                // Повертаємо результат обробки
                return this.processedOK;
            }

            // Створюємо потік обчислення й перевірки CRC-64...
            this.thrFileAnalyzer = new Thread(new ThreadStart(AnalyzeCRC64));

            //...потім даємо йому ім'я...
            this.thrFileAnalyzer.Name = "FileAnalyzer.AnalyzeCRC64()";

            //...установлюємо обраний пріоритет завдання...
            this.thrFileAnalyzer.Priority = this.threadPriority;
```

```
            //...і запускаємо його
            this.thrFileAnalyzer.Start();

            // Повідомляємо, що все нормально
            return true;
        }

        /// <summary>
        /// Метод зупинки потоку
        /// </summary>
        public void Stop()
        {
            // Указуємо, що потік обробки більше не повинен виконуватися
            this.exitEvent[0].Set();

            // Примусово знімаємо з паузи
            this.executeEvent[0].Set();

            // Знімаємо з очікування в циклі
            this.wakeUpEvent[0].Set();
        }

        /// <summary>
        /// Постановка потоку обробки на паузу
        /// </summary>
        public void Pause()
        {
            // Ставимо на паузу
            this.executeEvent[0].Reset();

            // Знімаємо з очікування в циклі
            this.wakeUpEvent[0].Set();
        }

        /// <summary>
        /// Зняття потоку обробки з паузи
        /// </summary>
        public void Continue()
        {
            // Знімаємо обробку с паузи
            this.executeEvent[0].Set();
        }

        #endregion Public Operations

        #region Private Operations

        /// <summary>
        /// Обчислення й запис у кінець файлів значення CRC-64
        /// </summary>
        private void WriteCRC64()
        {
            // Обчислюємо значення модуля, що дозволить виводити відсоток
обробки
            // рівно при одиничному збільшенні для циклу по "i"
            int progressMod1 = (this.dataCount + this.eccCount) / 100;

            // Якщо модуль дорівнює нулю, то збільшуємо його до значення "1",
щоб
            // прогрес виводився на кожній ітерації (файл дуже маленький)
            if (progressMod1 == 0)
            {
                progressMod1 = 1;
            }

            // Піддаємо обробці всі томи
            for (int volNum = 0; volNum < (this.dataCount + this.eccCount);
volNum++)
            {
```

```
                // Зчитуємо первісне ім'я файлу
                String fileName = this.fileName;

                // Одержуємо ім'я вихідного файлу в префіксній формі
                this.eFileNamer.Pack(ref fileName, volNum, this.dataCount,
this.eccCount, this.codecType);

                // Формуємо повне ім'я файлу
                fileName = this.path + fileName;

                // Робимо обчислення CRC-64 для кожного файлу
                if (this.eFileIntegrityCheck.StartToWriteCRC64(fileName, true))
                {
                    // Цикл очікування завершення обробки файлу
                    while (true)
                    {
                        // Якщо не виявили встановленої події "executeEvent",
                        // те користувач хоче, щоб ми поставили обробку на паузу
-
                        if (!ManualResetEvent.WaitAll(this.executeEvent, 0,
false))
                        {
                            //...припиняємо роботу контрольованого алгоритму...
                            this.eFileIntegrityCheck.Pause();

                            //...програма переходить у режим сна
                            ManualResetEvent.WaitAll(this.executeEvent);

                            // А коли прокинулися, указуємо, що обробка повинна
тривати
                            this.eFileIntegrityCheck.Continue();
                        }

                        // Чекаємо кожне з перерахованих подій...
                        int eventIdx = ManualResetEvent.WaitAny(new
ManualResetEvent[] { this.wakeUpEvent[0], this.exitEvent[0],
this.eFileIntegrityCheck.FinishedEvent[0] });

                        //...якщо одержали сигнал до того, щоб прокинутися -
                        // переходимо на нову ітерацію, тому що прокидаємося
                        // перед постановкою на паузу...
                        if (eventIdx == 0)
                        {
                            //...попередньо скинувши подію, що змусила нас
прокинутися
                            this.wakeUpEvent[0].Reset();

                            continue;
                        }

                        //...якщо одержали сигнал до виходу з обробки...
                        if (eventIdx == 1)
                        {
                            //...зупиняємо контрольований алгоритм
                            this.eFileIntegrityCheck.Stop();

                            // Указуємо на те, що обробка була перервана
                            this.processedOK = false;

                            // Активуємо індикатор актуального стану змінних-
членів
                            this.finished = true;

                            // Установлюємо подію завершення обробки
                            this.finishedEvent[0].Set();

                            return;
                        }
```

```
                        //...якщо одержали сигнал про завершення обробки
вкладеним алгоритмом...
                        if (eventIdx == 2)
                        {
                            //...виходимо із циклу очікування завершення (цього
й чекали в while(true)!)
                            break;
                        }

                    } // while(true)

                } else
                {
                    // Скидаємо прапор коректності результату
                    this.processedOK = false;

                    // Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;

                    // Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();

                    return;
                }

                // У зв'язку із закриттям великої кількості файлових потоків
                // необхідно дочекатися запису змін, внесених потоком
                // кодування в тіло класу. Потік уже не працює, але
                // установлена їм Булевська властивість, можливо, ще
                // "не виявилося"
                for (int i = 0; i < (int)WaitCount.MaxWaitCount; i++)
                {
                    if (!this.eFileIntegrityCheck.Finished)
                    {
                        Thread.Sleep((int)WaitTime.MinWaitTime);

                    } else
                    {
                        break;
                    }
                }

                // Якщо цикли очікування закриття файлових потоків не привели до
бажаного
                // результату - це помилка
                if (!this.eFileIntegrityCheck.ProcessedOK)
                {
                    // Указуємо на те, що обробка не була завершена коректно
                    this.processedOK = false;

                    // Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;

                    // Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();

                    return;
                }

                // Виводимо прогрес обробки
                if (
                        ((volNum % progressMod1) == 0)
                     &&
                        (OnUpdateFileAnalyzeProgress != null)
                    )
                {
                    OnUpdateFileAnalyzeProgress(((double)(volNum + 1) /
(double)(this.dataCount + this.eccCount)) * 100.0);
                }
```

```
                // У випадку, якщо потрібна постановка на паузу, подію
"executeEvent"
                // буде скинуто, і будемо на паузі аж до його появи
                ManualResetEvent.WaitAll(this.executeEvent);

                // Якщо зазначено, що потрібно вийти з потоку - виходимо
                if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                {
                    // Указуємо на те, що обробка була перервана
                    this.processedOK = false;

                    // Активуємо індикатор актуального стану змінних-членів
                    this.finished = true;

                    // Установлюємо подію завершення обробки
                    this.finishedEvent[0].Set();

                    return;
                }
            }

            // Повідомляємо про закінчення процесу обробки
            if (OnFileAnalyzeFinish != null)
            {
                OnFileAnalyzeFinish();
            }

            // Повідомляємо, що обробка пройшла коректно
            this.processedOK = true;

            // Активуємо індикатор актуального стану змінних-членів
            this.finished = true;

            // Установлюємо подію завершення обробки
            this.finishedEvent[0].Set();
        }

          /// <summary>
        /// Обчислення й перевірка значення CRC-64, записаного наприкінці файлу
        /// </summary>
        private void AnalyzeCRC64()
        {
            // Обчислюємо значення модуля, що дозволить виводити відсоток
обробки
            // рівно при одиничному збільшенні для циклу по "i"
            int progressMod1 = (this.dataCount + this.eccCount) / 100;

            // Якщо модуль дорівнює нулю, то збільшуємо його до значення "1",
щоб
            // прогрес виводився на кожній ітерації (файл дуже маленький)
            if (progressMod1 == 0)
            {
                progressMod1 = 1;
            }

            // Виділяємо пам'ять під "volList"
            this.volList = new int[this.dataCount];

            // Виділяємо пам'ять під "altEccList"
            int[] altEccList = new int[this.eccCount];

            // Індекс у масиві томів
            int volListIdx = 0;

            // Індекс у масиві томів для відновлення
            int altEccListIdx = 0;

            // Лічильник кількості ушкоджених основних томів
```

```
            int dataVolMissCount = 0;

            // Лічильник кількості знайдених томів для відновлення
            int eccVolPresentCount = 0;

            // Ім'я файлу для обробки
            String fileName;

            // Піддаємо перевірці всі основні томи
            for (int dataNum = 0; dataNum < this.dataCount; dataNum++)
            {
                // Спочатку припускаємо, що поточний том ушкоджено
                bool dataVolIsOK = false;

                // Зчитуємо первісне ім'я файлу
                fileName = this.fileName;

                // Одержуємо ім'я вихідного файлу в префіксній формі
                this.eFileNamer.Pack(ref fileName, dataNum, this.dataCount,
this.eccCount, this.codecType);

                // Формуємо повне ім'я файлу
                fileName = this.path + fileName;

                // Якщо вихідний файл існує...
                if (File.Exists(fileName))
                {
                    // Якщо не використовується швидке добування - перевіряємо
на цілісність
                    // CRC-64, інакше думаємо, що все корректно (цілісність тому
беремо
                    // по факті його наявності)
                    if (!this.fastExtraction)
                    {
                        //...- робимо його перевірку
                        if (this.eFileIntegrityCheck.StartToCheckCRC64(fileName,
true))
                        {
                            // Цикл очікування завершення обробки файлу
                            while (true)
                            {
                                // Якщо не виявили встановленої події
"executeEvent",
                                // те користувач хоче, щоб ми поставили обробку
на паузу -
                                if (!ManualResetEvent.WaitAll(this.executeEvent,
0, false))
                                {
                                    //...припиняємо роботу контрольованого
алгоритму...
                                    this.eFileIntegrityCheck.Pause();

                                    //...програма переходить у режим сна
                                    ManualResetEvent.WaitAll(this.executeEvent);

                                    // А коли прокинулися, указуємо, що обробка
повинна тривати
                                    this.eFileIntegrityCheck.Continue();
                                }

                                // Чекаємо кожне з перерахованих подій...
                                int eventIdx = ManualResetEvent.WaitAny(new
ManualResetEvent[] { this.wakeUpEvent[0], this.exitEvent[0],
this.eFileIntegrityCheck.FinishedEvent[0] });

                                //...якщо одержали сигнал до того, щоб
прокинутися -
                                // переходимо на нову ітерацію, тому що
прокидаємося
```

```
                                // перед постановкою на паузу...
                                if (eventIdx == 0)
                                {
                                    //...попередньо скинувши подію, що змусила
нас прокинутися
                                    this.wakeUpEvent[0].Reset();

                                    continue;
                                }

                                //...якщо одержали сигнал до виходу з обробки...
                                if (eventIdx == 1)
                                {
                                    //...зупиняємо контрольований алгоритм
                                    this.eFileIntegrityCheck.Stop();

                                    // Указуємо на те, що обробка була перервана
                                    this.processedOK = false;

                                    // Активуємо індикатор актуального стану
змінних-членів
                                    this.finished = true;

                                    // Установлюємо подію завершення обробки
                                    this.finishedEvent[0].Set();

                                    return;
                                }

                                //...якщо одержали сигнал про завершення обробки
вкладеним алгоритмом...
                                if (eventIdx == 2)
                                {
                                    //...виходимо із циклу очікування завершення
(цього й чекали в while(true)!)
                                    break;
                                }

                            } // while(true)

                        } else
                        {
                            // Скидаємо прапор коректності результату
                            this.processedOK = false;

                            // Активуємо індикатор актуального стану змінних-
членів
                            this.finished = true;

                            // Установлюємо подію завершення обробки
                            this.finishedEvent[0].Set();

                            return;
                        }

                        // У зв'язку із закриттям великої кількості файлових
потоків
                        // необхідно дочекатися запису змін, внесених потоком
                        // кодування в тіло класу. Потік уже не працює, але
                        // установлене їм Булевська властивість, можливо, ще
                        // "не виявилося"
                        for (int i = 0; i < (int)WaitCount.MaxWaitCount; i++)
                        {
                            if (!this.eFileIntegrityCheck.Finished)
                            {
                                Thread.Sleep((int)WaitTime.MinWaitTime);

                            } else
                            {
```

```
                                break;
                            }
                        }

                        // Указуємо, що основний том коректний
                        if (this.eFileIntegrityCheck.ProcessedOK)
                        {
                            dataVolIsOK = true;
                        }

                    } else
                    {
                        // Указуємо, що основний том коректний
                        dataVolIsOK = true;
                    }

                    // Виводимо прогрес обробки
                    if (
                            ((dataNum % progressMod1) == 0)
                     &&
                            (OnUpdateFileAnalyzeProgress != null)
                        )
                    {
                        OnUpdateFileAnalyzeProgress(((double)(dataNum + 1) /
(double)(this.dataCount + this.eccCount)) * 100.0);
                    }

                    // У випадку, якщо потрібна постановка на паузу, подію
"executeEvent"
                    // буде скинуто, і будемо на паузі аж до його появи
                    ManualResetEvent.WaitAll(this.executeEvent);

                    // Якщо зазначено, що потрібно вийти з потоку - виходимо
                    if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                    {
                        // Указуємо на те, що обробка була перервана
                        this.processedOK = false;

                        // Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;

                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();

                        return;
                    }
                }

                // Якщо даний основний том не ушкоджений, записуємо його в
"volList",
                // а інакше збільшуємо лічильник ушкоджених томів і ставимо на
місце
                // номера тому значення "-1", що вкаже на необхідність
підстановки
                // тому для відновлення
                if (dataVolIsOK)
                {
                    this.volList[volListIdx++] = dataNum;

                } else
                {
                    this.volList[volListIdx++] = -1;

                    // Збільшуємо лічильник кількості ушкоджених основних томів
                    dataVolMissCount++;
                }
            }

            // Тепер, коли знаємо кількість ушкоджених основних томів,
```

```
            // потрібно просканувати всі файли для відновлення, і визначити
            // необхідну їхню частину в список томів, а "надлишок" помістити в
            // список альтренативних томів для відновлення
            for (int eccNum = this.dataCount; eccNum < (this.dataCount +
this.eccCount); eccNum++)
            {
                // Спочатку припускаємо, що поточний том ушкоджено
                bool eccVolIsOK = false;

                // Зчитуємо первісне ім'я файлу
                fileName = this.fileName;

                // Одержуємо ім'я вихідного файлу в префіксній формі
                this.eFileNamer.Pack(ref fileName, eccNum, this.dataCount,
this.eccCount, this.codecType);

                // Формуємо повне ім'я файлу
                fileName = this.path + fileName;

                // Якщо вихідний файл існує...
                if (File.Exists(fileName))
                {
                    // Якщо не використовується швидке добування - перевіряємо
на цілісність
                    // CRC-64, інакше думаємо, що все корректно (цілісність тому
беремо
                    // по факту його наявності)
                    if (!this.fastExtraction)
                    {
                        //...- робимо його перевірку
                        if (this.eFileIntegrityCheck.StartToCheckCRC64(fileName,
true))
                        {
                            // Цикл очікування завершення обробки файлу
                            while (true)
                            {
                                // Якщо не виявили встановленої події
"executeEvent",
                                // то користувач хоче, щоб ми поставили обробку
на паузу -
                                if (!ManualResetEvent.WaitAll(this.executeEvent,
0, false))
                                {
                                    //...припиняємо роботу контрольованого
алгоритму...
                                    this.eFileIntegrityCheck.Pause();

                                    //...програма переходить у режим сна
                                    ManualResetEvent.WaitAll(this.executeEvent);

                                    // А коли прокинулися, указуємо, що обробка
повинна тривати
                                    this.eFileIntegrityCheck.Continue();
                                }

                                // Чекаємо кожне з перерахованих подій...
                                int eventIdx = ManualResetEvent.WaitAny(new
ManualResetEvent[] { this.wakeUpEvent[0], this.exitEvent[0],
this.eFileIntegrityCheck.FinishedEvent[0] });

                                //...якщо одержали сигнал до того, щоб
прокинутися -
                                // переходимо на нову ітерацію, тому що
прокидаємося
                                // перед постановкою на паузу...
                                if (eventIdx == 0)
                                {
                                    //...попередньо скинувши подію, що змусила
нас прокинутися
```

```
                                this.wakeUpEvent[0].Reset();

                                continue;
                            }

                            //...якщо одержали сигнал до виходу з обробки...
                            if (eventIdx == 1)
                            {
                                //...зупиняємо контрольований алгоритм
                                this.eFileIntegrityCheck.Stop();

                                // Указуємо на те, що обробка була перервана
                                this.processedOK = false;

                                // Активуємо індикатор актуального стану
змінних-членів
                                this.finished = true;

                                // Установлюємо подію завершення обробки
                                this.finishedEvent[0].Set();

                                return;
                            }

                            //...якщо одержали сигнал про завершення обробки
вкладеним алгоритмом...
                            if (eventIdx == 2)
                            {
                                //...виходимо із циклу очікування завершення
(цього й чекали в while(true)!)
                                break;
                            }

                        } // while(true)

                    } else
                    {
                        // Скидаємо прапор коректності результату
                        this.processedOK = false;

                        // Активуємо індикатор актуального стану змінних-
членів
                        this.finished = true;

                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();

                        return;
                    }

                    // У зв'язку із закриттям великої кількості файлових
потоків
                    // необхідно дочекатися запису змін, внесених потоком
                    // кодування в тіло класу. Потік уже не працює, але
                    // установлена їм Булевська властивість, можливо, ще
                    // "не виявилася"
                    for (int i = 0; i < (int)WaitCount.MaxWaitCount; i++)
                    {
                        if (!this.eFileIntegrityCheck.Finished)
                        {
                            Thread.Sleep((int)WaitTime.MinWaitTime);

                        } else
                        {
                            break;
                        }
                    }

                    // Указуємо, що том для відновлення коректний
```

```
                        if (this.eFileIntegrityCheck.ProcessedOK)
                        {
                            eccVolIsOK = true;
                        }

                    } else
                    {
                        // Указуємо, що том для відновлення коректний
                        eccVolIsOK = true;
                    }

                    // Виводимо прогрес обробки
                    if (
                            ((eccNum % progressMod1) == 0)
                         &&
                            (OnUpdateFileAnalyzeProgress != null)
                        )
                    {
                        OnUpdateFileAnalyzeProgress(((double)(eccNum + 1) /
(double)(this.dataCount + this.eccCount)) * 100.0);
                    }

                    // У випадку, якщо потрібна постановка на паузу, подію
"executeEvent"
                    // буде скинуто, і будемо на паузі аж до його появи
                    ManualResetEvent.WaitAll(this.executeEvent);

                    // Якщо зазначено, що потрібно вийти з потоку - виходимо
                    if (ManualResetEvent.WaitAll(this.exitEvent, 0, false))
                    {
                        // Указуємо на те, що обробка була перервана
                        this.processedOK = false;

                        // Активуємо індикатор актуального стану змінних-членів
                        this.finished = true;

                        // Установлюємо подію завершення обробки
                        this.finishedEvent[0].Set();

                        return;
                    }
                }

                // Якщо том для відновлення гарний...
                if (eccVolIsOK)
                {
                    //...- додаємо його в список
                    altEccList[altEccListIdx++] = eccNum;

                    // Збільшуємо лічильник кількості томів для відновлення
                    eccVolPresentCount++;

                } else
                {
                    //...а інакше вказуємо, що том ушкоджено
                    altEccList[altEccListIdx++] = -1;
                }
            }

            // Якщо значення лічильника кількості коректних томів для
відновлення збігається
            // зі значенням лічильника томів для відновлення конфігурації - всі
томи для
            // відновлення є неушкодженими
            if (eccVolPresentCount == this.eccCount)
            {
                this.allEccVolsOK = true;
            }
```

```
            // Виводимо статистику ушкоджень
            if (OnGetDamageStat != null)
            {
                // Обчислюємо загальний відсоток ушкоджень (суму ушкоджень
                // основних томів і томів для відновлення ділимо на загальну
кількість томів)
                double percOfDamage = ((double)(dataVolMissCount +
(this.eccCount - eccVolPresentCount)) / (double)(this.dataCount +
this.eccCount)) * 100;

                // Обчислюємо відсоток "" альтернативних томів, щовижили, для
відновлення
                // Альтернативні томи - це спочатку ті томи, які не планується
використовувати для відновлення
                double percOfAltEcc = ((double)(eccVolPresentCount -
dataVolMissCount) / (double)this.eccCount) * 100;

                // Виводимо статистику ушкоджень
                OnGetDamageStat(percOfDamage, percOfAltEcc);
            }

            // Якщо немає ушкоджених основних томів, просто виходимо
            if (dataVolMissCount == 0)
            {
                // Повідомляємо про закінчення процесу обробки
                if (OnFileAnalyzeFinish != null)
                {
                    OnFileAnalyzeFinish();
                }

                // Указуємо на те, що дані не ушкоджені
                this.processedOK = true;

                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

                // Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();

                return;
            }

            // Якщо ми не зможемо відновити ушкодження...
            if (eccVolPresentCount < dataVolMissCount)
            {
                //...указуємо на те, що дані не можуть бути відновлені
                this.processedOK = false;

                // Активуємо індикатор актуального стану змінних-членів
                this.finished = true;

                // Установлюємо подію завершення обробки
                this.finishedEvent[0].Set();

                return;
            }

            // Переміщаємося на початок списку альтернативних томів для
відновлення
            altEccListIdx = 0;

            // Тепер пробігаємося по вектору "volList", і замість кожного зі
значень "-1"
            // підставляємо чергове значення зі знайденого діапазону
            for (int i = 0; i < this.dataCount; i++)
            {
                if (this.volList[i] == -1)
                {
                    // Пробігаємося по векторі томів для відновлення,
```

```
                    // зупиняючись на коректному томі для відновлення
                    while (altEccList[altEccListIdx] == -1)
                    {
                        altEccListIdx++;
                    }

                    // Підставляємо на місце ушкодженого основного тому
                    // том для відновлення,...
                    this.volList[i] = altEccList[altEccListIdx];

                    //...забираючи використаний том зі списку альтернативних
                    altEccList[altEccListIdx] = -1;
                }
            }

            // Повідомляємо про закінчення процесу обробки
            if (OnFileAnalyzeFinish != null)
            {
                OnFileAnalyzeFinish();
            }

            // Повідомляємо, що обробка пройшла коректно
            this.processedOK = true;

            // Активуємо індикатор актуального стану змінних-членів
            this.finished = true;

            // Установлюємо подію завершення обробки
            this.finishedEvent[0].Set();
        }

        #endregion Private Operations
    }
}
```

Файл About.cs - вікно довідки про програму

```
namespace RecoveryData
{
    partial class AboutForm
    {
        /// <summary>
        /// Необхідні змінні розробника.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true якщо керуючі ресурси повинні бути
розташовані, у іншому випадку  false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Необхідний метод для підтримки розробника - не модифікується
        /// зміст цього метода використовується редактором коду.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.developersListBoxdevelopersListBox = new
System.Windows.Forms.ListBox();
            this.CicleCodeIconTimer = new
System.Windows.Forms.Timer(this.components);
            this.okButtonXP = new PinkieControls.ButtonXP();
            this.SuspendLayout();
            //
            // developersListBoxdevelopersListBox
            //
            this.developersListBoxdevelopersListBox.BackColor =
System.Drawing.SystemColors.Control;
            this.developersListBoxdevelopersListBox.BorderStyle =
System.Windows.Forms.BorderStyle.None;
            this.developersListBoxdevelopersListBox.FormattingEnabled = true;
            this.developersListBoxdevelopersListBox.Items.AddRange(new object[]
{
            "БАКАЛАВРСЬКА ДИПЛОМНА РОБОТА",
            "",
            "На тему:",
            "",
            " Програмне забезпечення системи кібербезпеки перешкодостійкого
кодування для відеоконференцзв’язку у бездротових мережах ",
            "",
            "",
            "Керівник: Смірнов С.А.",
            "",
            "Розробив: студент Іушин Максим Олександрович",
            "                  гр. КБ-21",
            "",
            "м. Кропивницький 2025"});
            this.developersListBoxdevelopersListBox.Location = new
System.Drawing.Point(11, 6);
            this.developersListBoxdevelopersListBox.Name =
"developersListBoxdevelopersListBox";
```

```
            this.developersListBoxdevelopersListBox.SelectionMode =
System.Windows.Forms.SelectionMode.None;
            this.developersListBoxdevelopersListBox.Size = new
System.Drawing.Size(330, 182);
            this.developersListBoxdevelopersListBox.TabIndex = 0;
            this.developersListBoxdevelopersListBox.TabStop = false;
            this.developersListBoxdevelopersListBox.SelectedIndexChanged += new
System.EventHandler(this.developersListBoxdevelopersListBox_SelectedIndexChanged
);
            //
            // CicleCodeIconTimer
            //
            this.CicleCodeIconTimer.Interval = 40;
            this.CicleCodeIconTimer.Tick += new
System.EventHandler(this.CicleCodeIconTimer_Tick);
            //
            // okButtonXP
            //
            this.okButtonXP.BackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(0)))), ((int)(((byte)(236)))),
((int)(((byte)(233)))), ((int)(((byte)(216)))));
            this.okButtonXP.DefaultScheme = true;
            this.okButtonXP.DialogResult =
System.Windows.Forms.DialogResult.None;
            this.okButtonXP.Hint = "";
            this.okButtonXP.Location = new System.Drawing.Point(266, 177);
            this.okButtonXP.Name = "okButtonXP";
            this.okButtonXP.Scheme = PinkieControls.ButtonXP.Schemes.Blue;
            this.okButtonXP.Size = new System.Drawing.Size(75, 23);
            this.okButtonXP.TabIndex = 0;
            this.okButtonXP.Text = "OK";
            this.okButtonXP.Click += new
System.EventHandler(this.okButtonXP_Click);
            //
            // AboutForm
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(350, 212);
            this.Controls.Add(this.okButtonXP);
            this.Controls.Add(this.developersListBoxdevelopersListBox);
            this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedDialog;
            this.MaximizeBox = false;
            this.MinimizeBox = false;
            this.Name = "AboutForm";
            this.ShowInTaskbar = false;
            this.StartPosition =
System.Windows.Forms.FormStartPosition.CenterParent;
            this.Text = "Про програму...";
            this.Load += new System.EventHandler(this.AboutForm_Load);
            this.FormClosing += new
System.Windows.Forms.FormClosingEventHandler(this.AboutForm_FormClosing);
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.ListBox developersListBoxdevelopersListBox;
        private System.Windows.Forms.Timer CicleCodeIconTimer;
        private PinkieControls.ButtonXP okButtonXP;

    }
}
```