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
“____” _____ 2024 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“ Програмне забезпечення системи кібербезпеки комплексу захищеної взаємодії з інформаційними носіями”

Виконав здобувач вищої освіти
IV курсу, групи КБ-20
ОПП «Кібербезпека»
спеціальності 125 «Кібербезпека»
_____ Тихий Р. В.
«____» _____ 2024 р.

Керівник проекту
кандидат технічних наук
_____ Олександр Улічев
«____» _____ 2024 р.
Рецензент _____
_____

м. Кропивницький

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітньо-кваліфікаційний рівень *бакалавр*

Спеціальність *125 Кібербезпека*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ О.А. Смірнов
« 17 » січня 2024 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Тихому Ростиславу Вадимовичу
(прізвище, ім'я, по батькові)

1. Тема проекту *Програмне забезпечення системи кібербезпеки комплексу захищеної взаємодії з інформаційними носіями*

керівник проекту *Улічев Олександр Сергійович, канд. техн. наук*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по кафедрі *№135-02, від 01.04.24*

2. Строк подання студентом проекту *19.05.2024 р.*

3. Вихідні дані до проекту *Пояснювальна записка, графічні матеріали, робоча програма*

4. 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 в промислову експлуатацію.
6. Висновки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

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

6. Дата видачі завдання « 17 » січня 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів бакалаврської дипломної роботи	Примітка
1.	Аналіз існуючих систем	21.01.2024 р.	
2.	Постановка задачі, оформлення ТЗ	11.03.2024 р.	
3.	Розробка моделі компонента	16.03.2024 р.	
4.	Розробка структур даних	21.03.2024 р.	
5.	Розробка алгоритмів зв’язку та відображення	27.03.2024 р.	
6.	Програмування алгоритмів	12.04.2024 р.	
7.	Оформлення ПЗ	15.05.2024 р.	
8.	Попередній захист бакалаврської дипломної роботи	19.05.2024 р.	

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Тихий Р.В. Програмне забезпечення системи кібербезпеки комплексу захищеної взаємодії з інформаційними носіями. 125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2024.

В даній кваліфікаційній роботі досліджується питання програмного захисту даних на зовнішніх носіях.

На початковому етапі дослідження приведено коротку характеристику та порівняння носіїв даних різних типів, розглянуто переваги та недоліки пристроїв, що базуються на різних технологіях. Далі розглянуто способи обмеження доступу на прикладі USB-Flash.

Спроектовано та реалізовано програмний комплекс для забезпечення захищеного доступу до файлів на зовнішніх носіях.

Основна частина роботи складається з п’яти розділів, загальний об’єм роботи 77 сторінок.

Даний проект присвячено питанням захисту файлів на зовнішніх носіях, за рахунок шифрування та доступу до змісту файлів через спеціальний програмний додаток.

Об’єкт досліджння – технології та підходи захисту даних та обмеження несанкціонованого доступу до даних на зовнішніх носіях.

Предмет дослідження – застосування технологій для реалізації програмного комплексу обмеження доступу до файлів на флеш-накопичувачі чи зовнішньому жорсткому диску.

Метою випускної бакалаврської кваліфікаційної роботи є реалізація програмного додатку, що використовує ефективні технології захисту даних на зовнішніх накопичувачах.

Ключові слова: пароль, авторизація, зовнішній носій даних, шифрування, доступ до даних.

ABSTRACT

R.V. Tykhyy. Software of the cyber security system of the complex of protected interaction with information carriers. 125 Cybersecurity. Central Ukrainian National Technical University. Kropyvnytskyi. 2024.

This qualification work examines the issue of software protection of data on external media.

At the initial stage of the research, a brief description and comparison of different types of data carriers is given, the advantages and disadvantages of devices based on different technologies are considered. Next, methods of restricting access are considered using USB-Flash as an example.

A software complex was designed and implemented to ensure secure access to files on external media.

The main part of the work consists of five chapters, the total volume of the work is 77 pages.

This project is devoted to the protection of files on external media, due to encryption and access to the content of files through a special software application.

The object of research is data protection technologies and approaches and limiting unauthorized access to data on external media.

The subject of the research is the application of technologies for the implementation of a software package for restricting access to files on a flash drive or external hard drive.

The goal of the final bachelor's qualification work is the implementation of a software application that uses effective data protection technologies on external drives.

Keywords: password, authorization, external data carrier, encryption, data access

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, СИМВОЛІВ ТА СПЕЦІАЛЬНИХ ТЕРМІНІВ

БД	-	база даних
СУБД	-	система управління базами даних
ЗД	-	захист даних
НД	-	несанкціонований доступ
ЕФ	-	екранна форма
ТЗ	-	технічне завдання
МЗ	-	медичний заклад
ЗПД	-	захист персональних даних
ЦСС	-	цифрова стеганографічна система
КРИПТОГРАФІЯ	-	наука про перетворення інформації з метою захисту
HDD	-	жорсткий диск, зовнішній накопичувач
ІСЗД	-	інформаційні системи захисту даних

ВСТУП

Наразі інформаційні системи захисту даних (ІСЗД) набувають все більшої актуальності та значущості.

Системи захисту інформації в даний час є невід'ємною частиною будь-якого процесу, пов'язаного з обробкою, зберіганням та передачею даних.

У вирішені цього питання допомагають апаратні та програмні технології захисту інформації, наприклад, шифрування інформації, що підлягає захисту.

Для того, щоб не потрапити в ситуацію, коли можуть бути скопійовані, поширені та скомпрометовані особисті дані потрібно запроваджувати системи, що дозволяють запобігати розкраданню чи несанкціонованому доступу до даних. Задача може бути реалізована за допомогою правильного вибору та інтеграції засобів захисту даних на зовнішніх носіях.

Багато користувачів електронних ресурсів достатньо безвідповідально підходять до технологій захисту інформації, в силу власної лінощів, або небажання ускладнювати собі роботу з інформацією. Користувачі зазвичай не хочуть запам'ятовувати складні паролі або порядок дій для захисту інформації.

Для підвищення обізнаності з технологіями захисту інформації потрібно ознайомити людей з альтернативними методами захисту інформації на зовнішніх накопичувачах.

Об’єкт ВКР – технології та підходи захисту даних та обмеження несанкціонованого доступу до даних на зовнішніх носіях.

Предмет дослідження ВКР – застосування технологій для реалізації програмного комплексу обмеження доступу до файлів на флеш-накопичувачі чи зовнішньому жорсткому диску.

Метою випускної бакалаврської кваліфікаційної роботи є реалізація програмного додатку, що використовує ефективні технології захисту даних на зовнішніх накопичувачах.

В випускній кваліфікаційній бакалаврській роботі визначено наступні **задачі**:

1) провести аналіз технологій захисту даних на зовнішніх накопичувачах;

2) вибрати найефективніші та прийнятні, відносно постановки задачі, технології для реалізації;

3) розробити програму, що використовує технології захисту даних на зовнішніх накопичувачах;

4) підготувати технічну документацію.

Практична значимість дослідження полягає в можливості впровадження розробленого програмного комплексу в експлуатацію для організацій, що потребують обмеження доступу до файлів на зовнішніх носіях (державні установи, комерційні організації, дослідницькі центри, організації, що оперують персональними даними).

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Система, що досліджується в роботі, призначена для захисту даних на зовнішніх носіях. Основний фактор, захист від якого передбачено систмою – несанкціонований доступ до даних.

Несанкціонований доступ (НДД).

Якщо будь-який бажаючий може отримати доступ до будь-яких даних і внести в них будь-які зміни, то дані можуть бути зруйновані некомпетентним або зловмисним користувачем або використані з утиском прав власника або на шкоду йому.

НДД може переслідувати різні цілі:

1) Крадіжка інформації.

2) Передача даних третім особам.

3) Технічний та фінансовий шпіонаж.

4) Підміна даних.

5) Інші цілі.

Саме для мінімалізації ризиків НДД і розробляється система, що є практичною складовою дослідженя ВКР.

1.2 Область застосування

Розглянемо більш детально область застосування системи, що проектується.

Комп'ютерна пам'ять (пристрій зберігання інформації, запам'ятовуючий пристрій) - частина обчислювальної машини, фізичний пристрій або середовище для зберігання даних, що використовується в обчисленнях, протягом певного часу [22].

Пам'ять комп'ютера повинна мати високу інформаційну ємність, невеликий час звернення (висока швидкодія), високу надійність та невелику (відносно) вартість. Але зі зростанням ємності знижується швидкодія та зростає ціна. Розподіл пам'яті на оперативний запам'ятовуючий пристрій (ОЗП) і зовнішній пристрій (ЗЗП) не знімає цю суперечність цілком, оскільки різниця у швидкодії процесора, ОЗП та ЗЗП дуже велика. Тому обмін інформацією відбувається через додаткові буферні пристрої, тобто пам'ять має ієрархічну багаторівневу структуру (рисунок 1.1). Чим вища швидкодія запам'ятовуючого пристрою (ЗП), тим більша вартість зберігання 1 байту й менша ємність ЗП.

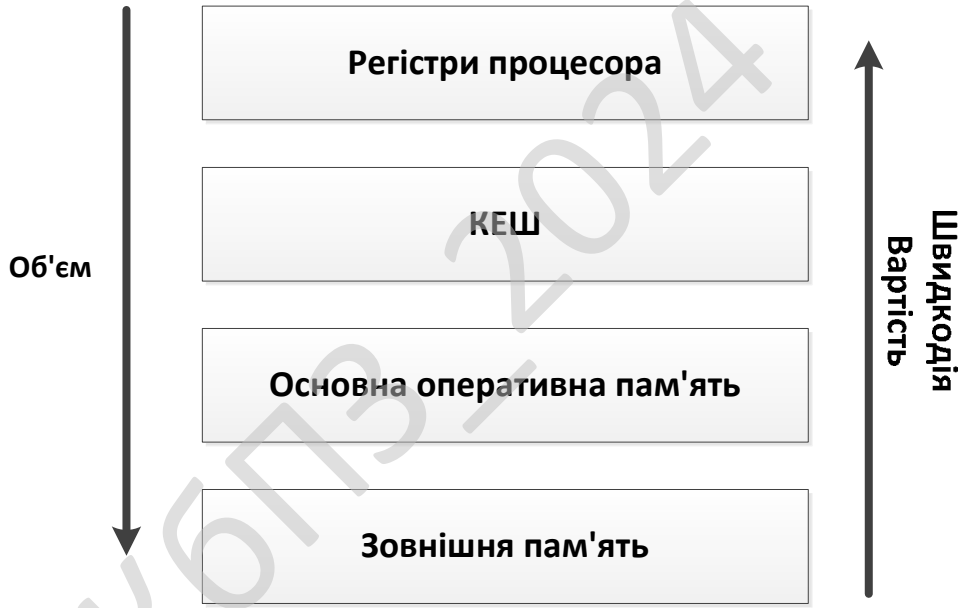


Рисунок 1.1 – Ієрархія пам’яті ЕОМ

Зовнішній запам'ятовуючий пристрій (ЗЗП) призначений для тривалого зберігання програм та даних, і цілісність його вмісту не залежить від того, увімкнено або вимкнено комп'ютер. Цей вид пам'яті має великий об’єм і невелику швидкодію. На відміну від оперативної пам'яті, зовнішня пам'ять не має прямого зв'язку з процесором. Інформація від ЗЗП до процесора та навпаки циркулює приблизно по наступному ланцюжку (рисунок 1.2).

Рисунок 1.2 – Зв'язок ЗЗП з процесором

До складу зовнішньої пам'яті комп'ютера входять:

1. Накопичувачі на жорстких магнітних дисках.
2. Накопичувачі на гнучких магнітних дисках.
3. Оптичні (лазерні) CD та DVD диски.
4. Твердотільні накопичувачі.
5. USB-флеш-накопичувачі.
6. Зовнішні жорсткі диски.

USB-флеш-накопичувачі

USB-флеш накопичувач - запам'ятовуючий пристрій, що використовує як носій флеш-пам'ять, що підключається до комп'ютера чи іншого зчитуючого пристрою через інтерфейс USB (рисунок 1.3).

Рисунок 1.3 – USB-флеш-накопичувачі

Основне призначення USB-флеш накопичувача - зберігання, перенесення та

обмін даними, резервне копіювання, завантаження операційних систем (Live USB) та ін. В табл. 1.1 представлено основні показники досить різних накопичувачів, які сьогодні представлено на ринку.

Таблиця 1.1 - Основні характеристики USB-флеш-накопичувачів різних виробників

№	Виробник	Назва	Ємність	Ціна	Читання	Запис
1	PNY	Turbo	256GB	$30	80 MB/s	20 MB/s
2	Patriot	Viper Fang	128GB	$35	380 MB/s	70 MB/s
3	Samsung	T3 SSD	1TB	$230	450 MB/s	450 MB/s
4	SanDisk	Ultra Fit CZ43	32GB	$12	130 MB/s	40 MB/s
5	Verbatim	Pinstripe	128GB	$15	10 MB/s	4 MB/s
6	Kingston	IronKey D300S	64GB	$60	250 MB/s	85 MB/s
7	Corsair	Flash Survivor Stealth	256GB	$50	200 MB/s	90 MB/s
8	PNY	Pro Elite	256GB	$35	415 MB/s	249 MB/s

Пристрій USB flash досить компактні, мобільні та дають можливість підключитися до будь-якого комп'ютера, який має USB-роз'єм.

Пристрій USB-флеш накопичувача складається з наступних електронних компонентів (рис. 1.4).

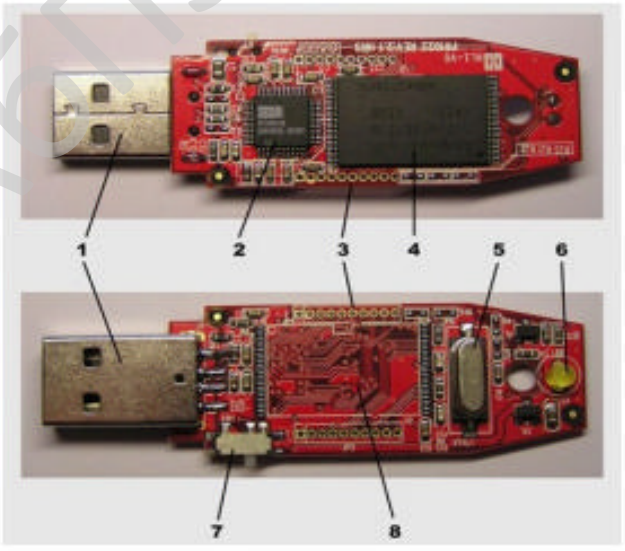


Рисунок 1.4 – Компоненти USB Flash накопичувача

1. Роз'єм USB.

2. Мікроконтролер.

3. Контрольні точки.

4. Чіп (мікросхема) флеш-пам'яті.

5. Кварцовий резонатор.

6. Світлодіод.

7. Перемикач (захист від запису).

8. Місце для мікросхем пам'яті (додаткове місце).

Переваги USB Flash накопичувача:

1. Невеликий розмір, вага, портативність.

2. Накопичувач можна підключити до будь-якого пристрою зчитування (майже скрізь є USB).

3. Практично немає впливу від зовнішнього середовища (пил, подряпини, забрудненість).

4. USB флешка може працювати у широкому діапазоні температур.

5. Малі габарити дозволяють зберігати великий обсяг інформації.

6. Низьке енергоспоживання.

7. У порівнянні з жорстким диском, вона стійкіша до зовнішніх впливів, вібрації та удари.

8. Зручність підключення до пристрою.

9. Висока швидкість доступу до даних.

Недоліки USB-флешок:

1. Обмежена кількість циклів запису та стирання перед виходом з ладу.

2. Обмежений термін автономного зберігання даних.

3. Швидкість запису та читання обмежені пропускною спроможністю шини USB та самої флеш-пам'яті.

Накопичувачі на жорстких магнітних дисках (HDD) та твердотільні накопичувачі (SSD)

Жорсткий диск складається з декількох металевих пластин з магнітним покриттям, а зчитування та запис даних відбувається з допомогою магнітної

головки, яка пересувається над поверхнею обертається на високій швидкості диска (рис. 1.5).

Рисунок 1.5 – Жорсткий диск HDD

У твердотільних накопичувачів інший принцип роботи. У SSD немає будь-яких рухомих компонентів, а всередині він виглядає як набір мікросхем флеш-пам'яті, розміщених на одній платі (рис. 1.6).

Рисунок 1.6 – Твердотільні накопичувачі (SSD)

Такі чіпи можуть встановлюватись як на материнську плату системи, так і на карту PCI Express для стаціонарних комп'ютерів або спеціальний слот ноутбука. Використовувані в SSD-чіпи, відрізняються від чіпів у USB-флешнакопичувачі, вони значно надійніші та швидші.

Плюси та мінуси SSD та HDD

Завдання накопичувачів кожного класу зводяться до одного: забезпечити користувача функціонуючою операційною системою та дозволити зберігати йому

персональні дані. Але і у SSD, і HDD є свої особливості.

Ціна: SSD дорожче за традиційні HDD. Стандартний HDD на 8 ТБ у середньому обходиться у 8-10 тис. грн.. SSD ємністю в 8 ТБ в середньому обійдеться в 35-37 тис. грн, що майже в 4 рази дорожче за HDD.

Середня та максимальна ємність SSD та HDD сьогодні майже зрівнялись. Єдине – з збільшеннням розміру коефіцієнт різниці ціни зостає. SSD великого об’єму кошьтують дуже дорого.

Швидкість SSD та HDD: саме за цей показник переплачує користувач, коли віддає перевагу SSD-накопичувачу. Його швидкість багаторазово перевершують показники, якими може похвалитися HDD. Система здатна завантажуватися за кілька секунд, на запуск великих додатків та ігор йде менше часу, а копіювання значних обсягів даних з багатогодинного процесу обертається на 5-10 хвилинний. Дані з SSD-накопичувача видаляються так само швидко, наскільки копіюються.

Фрагментація: Після заповнення HDD, багаторазового видалення/копіювання файлів, жорсткий диск починає працювати повільніше. Це пов'язано з тим, що по всій поверхні магнітного диска розкидані частини файлу і коли користувач двічі клацаєте мишкою по будь-якому файлу, зчитуавльна головка змушена шукати ці фрагменти із різних секторів. Це явище і називається фрагментацією, а як профілактика передбачена програмно-апаратний процес дефрагментації чи упорядкування таких частин файлів у єдиний ланцюжок. Дефрагментацію періодично рекомендується виконувати на всіх типах HDD-накопичувачів, тим самим підтримуючи їхню оптимальну швидкість. Принцип роботи SSD відрізняється від HDD, а будь-які дані можуть записуватися в будь-який сектор пам'яті з подальшим миттєвим зчитуванням. Тому для накопичувачів SSD дефрагментація не потрібна.

Надійність та термін служби: головна перевага SSD-накопичувачів - відсутність рухомих елементів. Тому можна використовувати ноутбук з SSD в умовах, неминуче пов'язаних із зовнішніми вібраціями. На стабільності роботи системи це не позначиться. У HDD зчитуюча головка

розташовується всього в декількох мікрометрах від намагнічених болванок, і тому будь-яка сильна вібрація може призвести до появи «битих секторів» — областей, які стають непридатними до роботи.

Незважаючи на всі переваги SSD, у них є суттєвий недолік - обмежений цикл використання. Він безпосередньо залежить від кількості циклів перезапису блоків пам'яті. Іншими словами, якщо користувач щодня копіюватимете, видалятимете і знову копіюватимете гігабайти інформації, то незабаром це призведе до вичерпування ресурсу SSD.

Сучасні SSD-накопичувачі оснащені спеціальним контролером, який дбає про рівномірний розподіл даних по всіх блоках SSD. Так вдалося значно підвищити максимальний час роботи до 3000 - 5000 циклів.

Форм-фактор: конкуренція розмірів накопичувачів завжди була викликана типом пристроїв, на яких вони встановлюються. Так, для стаціонарного комп'ютера абсолютно некритична установка як 3.5-дюймового, так і 2.5-дюймового диска, а ось для портативних пристроїв, на кшталт ноутбуків, плеєрів та планшетів потрібен компактніший варіант. Найбільш мініатюрним серійним варіантом HDD вважався 1.8- дюймовий формат. У світі SSD все набагато перспективніше. Загальноприйнятий формат 2,5-дюйма став таким не через будь-які фізичні обмеження, з якими зіштовхуються технології, а лише з сумісності. У новому поколінні ультрабуків від формату 2.5" поступово відмовляються, роблячи накопичувачі все більш компактні, а корпуси самих пристроїв більш тонкими.

Підсумовуючи порівняння HDD і SSD, хочеться сформулювати основні переваги кожного типу накопичувачів.

Позитивні якості HDD: ємні, недорогі, доступні.

Недоліки HDD: повільні, бояться механічних впливів, шумні.

Позитивні якості SSD: абсолютно безшумні, зносостійкі, дуже швидкі, немає фрагментації.

Недоліки SSD: дорогі, теоретично обмежений ресурс експлуатації.

Зовнішні жорсткі диски

Зовнішній жорсткий диск - портативний пристрій. Назва успадкована від накопичувачів на жорстких магнітних дисках і не передбачає обов'язкову наявність будь-яких дисків у пристрої. Під цією назвою може матися на увазі як зовнішній жорсткий диск, так і зовнішній твердотільний накопичувач (рисунок 1.7).

Рисунок 1.7 – Зовнішній жорсткий диск

Система, що джосліджується в роботі може бути застосована як для окремих пристроїв так і для сховищ даних. Розглянемо деякі аспекти сховищ даних.

До найважливіших вимог до СД належить продуктивність. Однак через свої фізичні обмеження, HDD диски, що використовуються в сучасних СД, не можуть забезпечити необхідний приріст продуктивності [20]. Велика кількість механічних частин, незахищеність від трясіння та перевантаження, нешироκий робочий діапазон температур, велике енергоспоживання - це основні недоліки HDD-дисків.

Поява нових видів носіїв інформації - твердотільних накопичувачів дозволяє перейти на якісно новий рівень зберігання інформації, обумовлений досить високою надійністю та швидкістю зчитування інформації. Напівпровідникові накопичувачі інформації відрізняються високою швидкістю роботи. Якщо для жорсткого диска на обробку запиту йде в середньому 6-7 мс, для твердотільних накопичувачів інформації цей показник досягає 0,1 мс. При цьому кількість транзакцій у секунду зростає на 1-2 порядки.

Донедавна твердотільні накопичувачі були досить дорогими і використовувалися в гібридних системах разом із дисковими. В даний час при зниженні вартості твердотільних накопичувачів з'являється можливість створення

повністю твердотільних СД, що дозволяє суттєво заощадити площі, знизити споживання електроенергії та виділення тепла.

Твердотільний диск складається з головного контролера та підключених до нього мікросхеми флеш-пам'яті. При цьому окрема мікросхема пам'яті не має високою швидкістю. Завдання головного контролера твердотільного диска в тому, щоб записувати дані, що надходять паралельно відразу на кілька мікросхеми пам'яті.

Прискорення операцій читання/запису в порівнянні з жорсткими дисками здійснюється за рахунок розпаралелювання операцій. Робота з великим кількістю невеликих за обсягом файлів чи блоків даних відбувається одночасно, т.к. у накопичувача немає головок читання/запису, які необхідно переміщати, тобто. Контролер може паралельно працювати з кількома мікросхемами пам'яті.

Таким чином, при використанні твердотільних дисків у системах зберігання даних, необхідно враховувати два недоліки твердотільних дисків, що залишилися, а саме високу вартість і обмеження на перезапис даних, яка призводить до зношування.

Автори [20] представили науково-популярну статтю, що описує основні засади роботи SSD. Можна також відзначити статтю [22], в якій представлено короткий популярний огляд швидких пристроїв пам'яті SSD із тривимірною структурою 3D XPoint (розробленою Intel). Показано, що для цих пристроїв на сучасних платформах затримки однаково пов'язані з пристроєм зберігання даних та іншою (хостовою) системою. Зазначається, що у майбутньому такі пристрої можуть грати роль оперативної пам'яті.

Доступ до даних на SSD здійснюється з детальністю рівня сторінки.

У цьому сенсі сторінка пам'яті може розглядатися як блок HDD. Різниця між ними полягає в тому, що сторінка пам'яті SSD не підтримує місцевого оновлення, а може бути перезаписана лише після повного очищення. Однак операція очищення проводиться з детальністю не однієї сторінки, а блоку очищення, який складається з деякого числа (зазвичай 64 або 128) послідовні сторінки пам'яті. Відповідно,

перед тим, як блок буде очищено, всі актуальні дані повинні бути скопійовані з сторінок.

Це веде до серйозної проблеми - збільшення обсягу запису на SSD. Щоб приховати таку поведінку, у SSD існує спеціальне вбудоване програмне забезпечення (flash translation layer – FTL). Для підтримки немісцевого оновлення FTL підтримує відображення номерів логічних сторінки на номери фізичних сторінок, надаючи системі ілюзію місцевого оновлення. Інша важлива функція FTL – виконання складання сміття (Garbage collection - GC). У процесі складання сміття відбувається очищення одного або кількох блоків, коли стає недостатньо вільних сторінок для обслуговування запитів на запис або при простої диска. Зазвичай для збирання сміття вибирається блок із найменшою кількістю сторінок із актуальними даними, щоб мінімізувати надлишковий запис на диск. З іншого боку, так як кожен блок SSD має обмежену кількість циклів очищення, FTL виконує вирівнювання зносу, щоб рівномірно розподілити операції запису по блокам і цим продовжити час життя SSD. FTL зазвичай реалізований як внутрішнє програмне забезпечення (прошивка), яке запускає контролер SSD. Контролер SSD перекладає вхідні запити читання/запису в операції flash-пам'яті через flash-контролер Крім контролера SSD всередині диска існує ще три важливих компонента. Інтерфейс хоста з'єднує диск із хостом через інтерфейсний конектор, такий як SATA. Буфер оперативної пам'яті також зазвичай є в SSD для підвищення можливостей, пов'язаних із тимчасовим зберіганням даних та буферизацією запитів на запис. Дані постійно зберігаються в масиві осередків flash-пам'яті, які з'єднані з flash-контролером через кілька каналів. Кожен осередок flash-пам'яті складається з декількох кристалів, кожен кристал має кілька матриць, кожна з яких має всередині кілька flash-блоків. Докладніше архітектура SSD розглянута, наприклад, у статті [23]. Чотирирівнева ієрархія flash-пам'яті відповідає чотирьом рівням паралелізму: рівень каналу, рівень комірки, рівень кристала та рівень матриці.

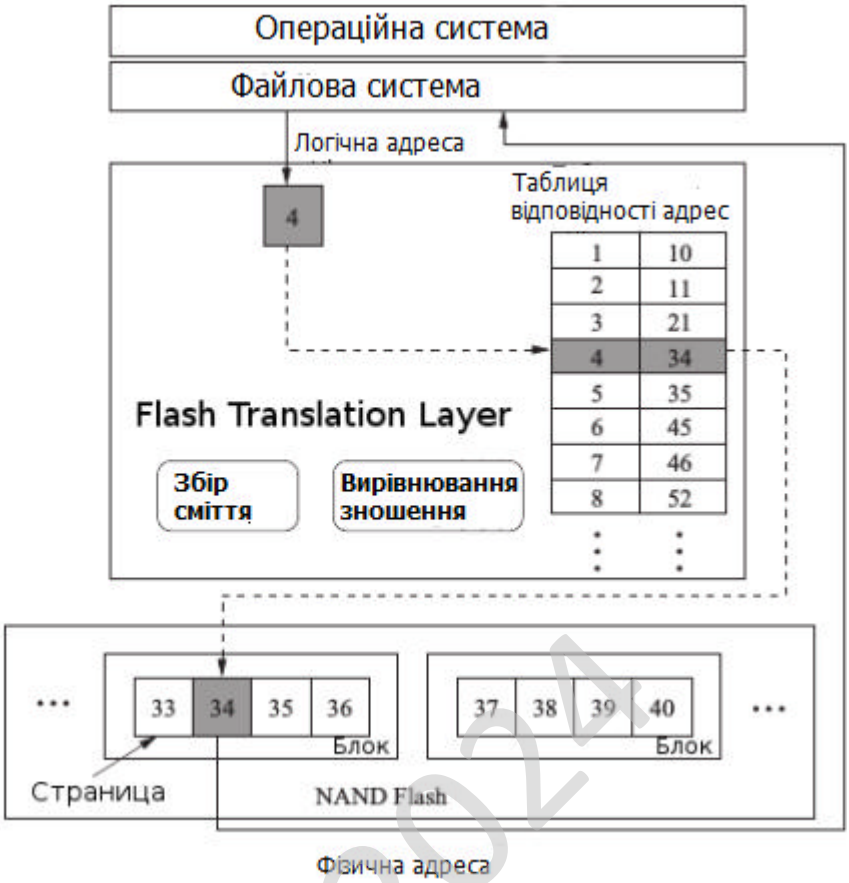


Рисунок 1.8 – Логічна схема SSD

Сторінки пам'яті, доступні по різних каналах, осередках або кристалах, можуть оброблятися незалежно, відповідно, маючи внутрішній паралелізм. Однак паралелізм рівня матриці неактивний у загальному випадку доки кілька операцій одного типу одночасно не виконуються над сторінками на різних матрицях одного кристала. В цьому випадку паралелізм рівня матриці може бути задіяний через команду n-plane, яка змушує n (зазвичай 2 або 4) матриці одного кристала працювати одночасно. Ряд досліджень (див., наприклад, [24; 25] були присвячені використання внутрішнього паралелізму SSD підвищення продуктивності операцій вводу-вывода. Так як такий паралелізм задіюється лише за послідовних записах, спеціалізовані додатки намагаються робити операції запису на SSD якомога послідовнішими. Одна з ключових особливостей SSD – це значне прискорення операцій довільного введення-виведення порівняно з HDD. Ієрархічна організація флеш-пам'яті передбачає використання кількох рівнів. Окремий осередок пам'яті є

транзистором з плаваючим затвором, програмування якого можливе тільки після стирання попереднього 21 держимого. Ряд осередків організований у рядок, при цьому рядки групуються з допомогою сторінок. Багато сторінок складає блок, ряд блоків організується в масив (див. рис. 1.4). При цьому необхідно враховувати низку наступних суттєвих відмінностей SSD дисків від HDD.

1. Операції зчитування виконуються зі сторінками фіксованого розміру (зазвичай 4 KB).

2. Операції запису передує операція стирання. При цьому стирання вміст можливий лише на рівні цілого блоку, який зазвичай складається з 64–128 сторінок. Це призводить до того, що кожного разу під час запису необхідно копіювати вміст блоку цілком, що суттєво уповільнює запис на пристрій.

3. Операція запису не може бути проведена в конкретну комірку, сторінки, що змінюються, додаються до блоків, що стираються або мають вільне місце. Потім, тільки коли всі сторінки блоку, що стирається звільнені, вони можуть бути очищені та відновлені механізмом збирання сміття.

4. У накопичувачах високої щільності, комірка пам'яті може містити до 8 значень (за рахунок застосування 8 рівнів напруги), таким чином, кодуючи 3 біта інформації. Малі розміри осередку призводять до високої ймовірності мимовільного витоку електронів, що вимагає застосування особливих схем корекції помилок та зниження ймовірності витоку для забезпечення тривалого зберігання (особливо у так званих холодних системах зберігання, а також у системах архівного зберігання) [26].

5. Число операцій видалення для кожного блоку обмежене і лежить у межах від 10 тисяч до 1 мільйона, що обмежує тривалість працездатності диска [27]. Це призводить до необхідності оптимізації операцій запису для збереження працездатності пристрою на протязі довгого часу. Як правило, надійність пристрою вимірюється кількістю інформації, яка може бути записана без збитків для ємності пристрою. Перспективність та економічна виправданність широкого використання SSD при побудові систем зберігання даних відзначається багатьма

авторами. Наприклад, у роботі [28] представлений розрахунок сукупної вартості володіння (TCO – Total Cost of Ownership) при використанні SSD у ЦОД, враховуючи такі особливості SSD, як обмежена кількість циклів перезапису, генерація більшої кількості операцій запису, ніж обсяг записуваних даних (WAF - Write amplification factor). Для цього автори намагаються порахувати WAF для різних типів навантаження і, відповідно, визначити інтенсивність запису, отже, і інтенсивність відмови SSD. Результати розрахунків показують, що використання SSD дозволяє значно знизити сукупну вартість володіння для системи зберігання даних при збереженні або навіть покращенні показників надійності, пропускної спроможності, обсягу дискового простору тощо.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд подібних систем

В роботі розглядається сситема захищеного доступу до файлів. Сьогодні більшість розробок тяжіють до хмарних технологій. За змістом ці системи близкі, але суттєво різняться за архітектурою та принципами будови.

Розглянемо кілька систем електронного документообігу, з найбільш відомих та близьких по функціоналу для детальнішого розгляду.

До них належать такі системи:

1) Seafile.

2) BitTorent Sync.

3) NextCloud.

4) Pydio.

5) FreeNAS.

6) ProjectSend.

7) OpenFiler.

Далі наведено короткі описи вибраних ситстем.

Seafile

Seafile – це хмарний сервіс для зберігання корпоративних файлів з шифрування на клієнтській стороні[56].

Seafile створений китайськими розробниками і поширюється на вихідних кодів. Він позиціонується як засіб синхронізації файлів та спільної роботи для команд. Seafile використовує центральне сховище, якому підключаються клієнти. Серверна частина існує у двох редакціях: Open Source та Professional. Оскільки Seafile позиціонується не тільки як сервіс файлової синхронізації, а й як засіб спільної роботи, у ньому передбачені такі інструменти, як вбудована вікі, ведення списків завдань, загальний доступ до файлів через веб, онлайн-перегляд файлів з

дискусіями, керування обліковими записами та групами, підтримка LDAP, обмін повідомленнями.

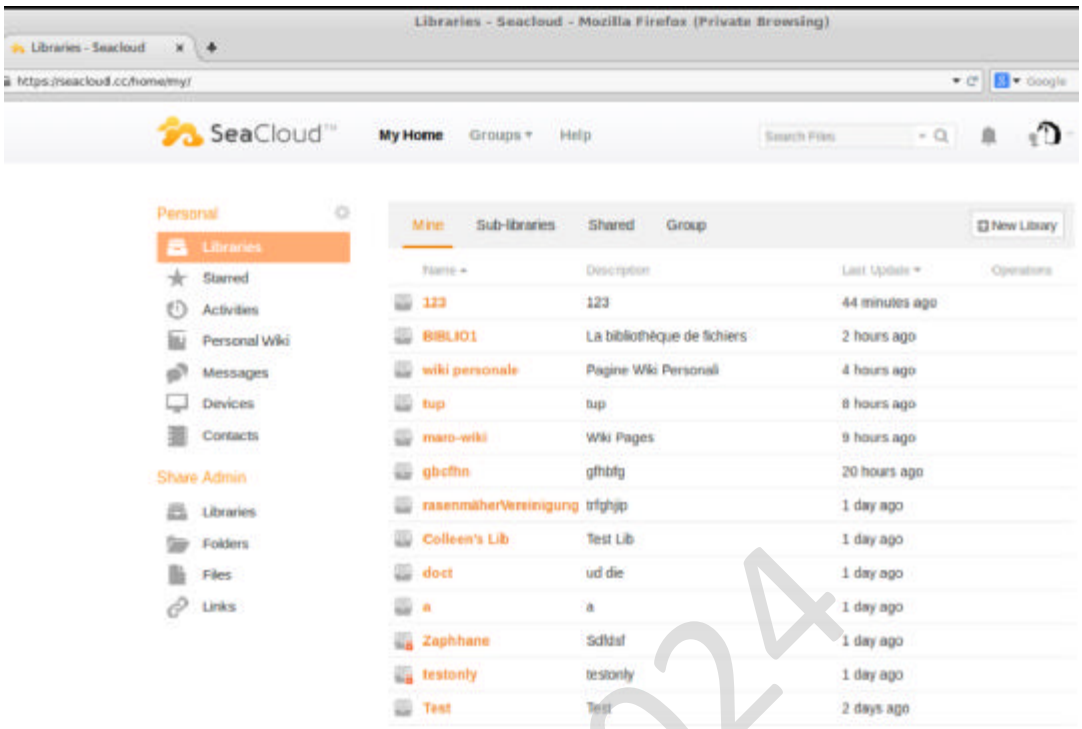


Рисунок 2.1 – Веб-інтерфейс Seafile

Seafile заснований на модифікованій під задачі файловій синхронізація моделі Git. Основним поняттям у Seafile є бібліотека, яка відповідає групі каталогів. На відміну від Git, файли поділяються на блоки для більш ефективної передачі по мережі та зберігання. Можна не тільки надавати права користувачам та групам на синхронізацію бібліотек, але і відкривати спільний доступ через веб як до окремих файлів, так і до каталогів з правами тільки на читання або на читання, і на запис. Як сервер баз даних Seafile може використовувати SQLite, MySQL, PostgreSQL, веб-сервери Apache та nginx. Скористатися Seafile можна і без встановлення свого сервера – хмарний сервіс Seacloud, побудований на основі Seafile, безкоштовно тарифному плані надає 1 Гб безкоштовного дискового простору та 5 Гб включеного трафіку. Для оцінки можливостей, що надаються Seafile, можна ознайомитись із демо-версією.

BitTorent Sync

У BitTorrent Sync використовується підхід, принципово відмінний від

інших систем. Синхронізація побудована на основі децентралізованого peer-to-peer протоколу.

Якщо файл доступний відразу на декількох пристроях, вони можуть передавати його одночасно, досягаючи при цьому максимально можливу швидкість. Не потрібно вказувати адреси сервера – пристрої з тим самим кодом знайдуть один одного автоматично. Для цього використовується кілька механізмів: пошук у локальній мережі за допомогою широкомовних пакетів, бенкети можуть обмінюватися один з одним інформацією про інші відомі їм бенкети, бенкет може бути заданий статично вказівкою адреси та порту, може бути використана DHT або BitTorrent трекер-сервер, який бенкети повідомляють про свою доступність і який може бути ними використаний для проксування трафіку за неможливості встановити пряме з'єднання.

При передачі файли шифруються і не зберігаються на будь-яких пристрої, крім тих, що були авторизовані користувачем. Для взаємної автентифікація пристроїв використовується SRP. Сама компанія BitTorrent хоч і має доступ до статистики сервісу, але заявляє, що жодні дані її Користувачів принципово неможливо знайти.

Оскільки центрального сховища у BTSync немає, всі учасники рівні, і, якщо дві групи учасників деякий час вийдуть із синхронізації, потім буде складно розібратися у тому, яка з версій основна.

Синхронізація через HTTP/HTTPS не підтримується, тому далеко не завжди він зможе пройти через мережеві екрани, і в сучасній захищеній корпоративного середовища йому доводиться туго. Немає можливості дати загальний доступ до окремого файлу/каталогу через Інтернет. Адміністрація великого кількості каталогів та пристроїв утруднено. Неможливо дати доступ для синхронізації до каталогу, що знаходиться всередині вже синхронізованого каталогу.[23]

NextCloud

Nextcloud - це набір клієнт-серверних програм для створення та

використання служб хостингу файлів. Він функціонально схожий на Dropbox, хоча Nextcloud є відкритим вихідним кодом, дозволяючи всім встановлювати та керувати ним на приватному сервері.

На відміну від пропрієтарних служб, таких як Dropbox, відкрита архітектура дозволяє додавати додаткові функції до сервера як додатків і дозволяє користувачу повністю контролювати свої дані.

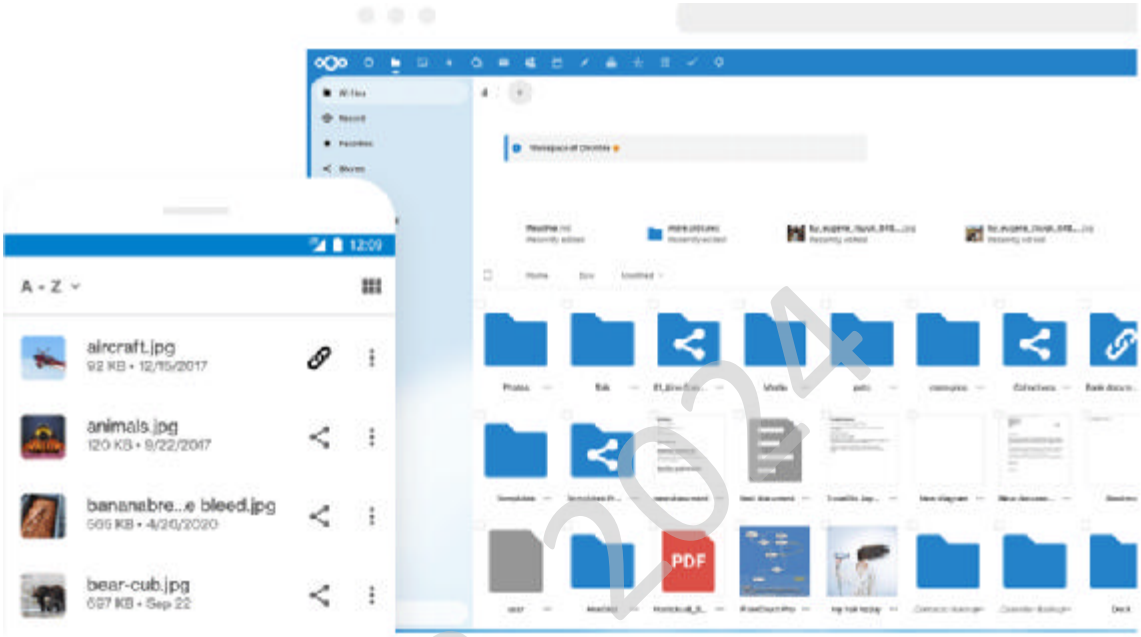


Рисунок 2.2 – Взаємодія з файлами в Nextcloud

Оригінальний розробник ownCloud Френк Карлічек розгалужив ownCloud та створив Nextcloud, який продовжує активно розвиватися Френком та іншими членами оригінальної команди ownCloud.

Файли Nextcloud зберігаються у звичайних структурах каталогів та можуть доступні через WebDAV, якщо це необхідно. Файли користувача зашифровуються під час транзиту і можуть бути зашифровані у мтані спокою.

Користувачі Nextcloud можуть керувати календарями (CalDAV), контактами (CardDAV), запланованими завданнями та потоковими медіа (Ampache) з платформи.

З погляду адміністрування Nextcloud дозволяє адміністрування користувачів та груп (через OpenID або LDAP). Контент може використовуватися

спільно, визначаючи грамотні дозволи на читання та запис між користувачами та/або групами. Крім того, користувачі Nextcloud можуть створювати загальнодоступні URL-адреси при спільному використання файлів. Також доступна реєстрація дій, пов'язаних із файлами, і навіть заборона доступу з урахуванням правил доступу до файлам.[14]

Захист диска за допомогою вбудованого Bitlocker

Перше, чим можна скористатися за необхідності захистити свої дані – це скористатися вбудованою у Windows функцією Bitlocker, яка досить надійно захистить диск. Для цього потрібно виконати декілька простих дій:

• Зайти в Панель керування та знайти кнопку «шифрування диска Bitlocker».

• У вікні, навпроти іконки диска, який хочемо захистити, натискаємо кнопку «включити Bitlocker».

• У наступному вікні уважно вибираємо способи зняття пароля. Найпростіший варіант - "Використання пароля для зняття блокування диска".

• Потрібно вибрати варіант збереження спеціального відновлюючого ключа, який потрібно використовувати, якщо пароль раптом буде втрачено. Можливо, оптимальним рішенням буде «Зберегти пароль у файл» та зберігання ключа відновлення, наприклад, у своїй хмарі.

• Якщо на ПК встановлена операційна система Windows 8 і вище, буде ще кілька пунктів: «Вказати частину диска для шифрування» і «Вибрати режим шифрування для використання».

Почнеться шифрування диска і після його закінчення доступ до диска можна буде отримати лише після введення пароля, який було встановлено. На зображенні зашифрованого диска у Провіднику з'явиться зображення замка.

При користуванні функцією Bitlocker потрібно бути дуже уважними, оскільки втративши пароль і ключ відновлення, працювати з цим диском ви зможете тільки після його повного форматування, при якому, відповідно, всі дані будуть видалені. Ще необхідно пам'ятати, що при переустановках ОС у тому випадку, якщо ви не розшифрували диск перед початком переустановки або не

заздалегідь скопіювали сертифікат на змінний жорсткий диск, отримати доступ до своїх даних ви більше не зможете.

Захист диска за допомогою сторонніх програм

Крім функції Bitlocker, яка реально шифрує дані на диску, існують програми сторонніх розробників, які можуть зашифрувати ваш диск, так і закрити диск без реального шифрування, а за допомогою функції парольного захисту.

Можна використовувати такі програми, як Disk Password Protection, Lockngo або Folder Lock.

За допомогою різних програм та утиліт повністю закривається доступ для сторонніх користувачів до диска або частини даних. Головне, при виборі та користуванні такими програмами, уважно поставитися до збереження своїх паролів і вивчити всі можливі можливості розблокування диска і даних, щоб самому не втратити доступ до них.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Microsoft Visual Studio - це програмне середовище з розробки програм для ОС Windows, як консольних, так і з графічним інтерфейсом.

У комплект входять такі основні компоненти:

1. Visual Basic.NET – для розробки додатків на VisualBasic;
2. Visual C++ - традиційною мовою C++;
3. Visual C# - мові C# (Microsoft);
4. Visual F# - F# (Microsoft Developer Division).

Функціональна структура середовища включає:

• редактор вихідного коду, який включає безліч додаткових функцій, як автодоповнення IntelliSense, рефракторинг коду тощо;

• налагоджувач коду;

• редактор форм, призначений для спрощеного конструювання графічних інтерфейсів;

• веб-редактор;

• дизайнер класів;

• дизайнер схем баз даних.

Visual Studio також дозволяє створювати та підключати сторонні доповнення (плагіни) для розширення функціональності практично на кожному рівні, включаючи додавання підтримки систем контролю версій вихідного коду (Subversion та VisualSourceSafe), додавання нових наборів інструментів (для редагування та візуального проектування коду на предметно-орієнтованих мовах) програмування або інструментів інших аспектів процесу розробки програмного забезпечення).

C# (вимовляється сі шарп) - об'єктно-орієнтована мова програмування. Розроблено в 1998-2001 роках групою інженерів компанії Microsoft під керівництвом Андерса Хейлсберга і Скотта Вільтаумота як мова розробки додатків для платформи Microsoft .NET Framework. Згодом був стандартизований як ECMA-334 та ISO/IEC 23270.

C# відноситься до сім'ї мов з C-подібним синтаксисом, їх синтаксис найбільш близький до C++ і Java. Мова має статичну типізацію, підтримує поліморфізм, навантаження операторів (у тому числі операторів явного та неявного приведення типу), делегати, атрибути, події, змінні, властивості, узагальнені типи та методи, ітератори, анонімні функції з підтримкою замикань, LINQ, винятки, коментарі у форматі XML.

Перейнявши багато від своїх попередників - мов C++, Delphi, Модула, Smalltalk і, особливо, Java - C#, спираючись на практику їх використання, виключає деякі моделі, що зарекомендували себе проблематичні при розробці програмних систем, наприклад, C# на відміну від C++ не підтримує множинне успадкування класів (між тим допускається множинна реалізація інтерфейсів).

C# розроблявся як мову програмування прикладного рівня для CLR і, як такої, залежить, перш за все, від можливостей CLR. Це стосується насамперед системи типів C#, яка відображає BCL. Присутність або відсутність тих чи інших виразних особливостей мови диктується тим, чи конкретна мовна особливість може бути трансльована у відповідні конструкції CLR. Так, з розвитком CLR від версії 1.1 до 2.0 значно збагатився сам C#; подібної взаємодії слід очікувати і надалі (проте, ця закономірність була порушена з виходом C# 3.0, що є розширення мови, що не спираються на розширення платформи .NET). CLR надає C#, як і всім іншим .NET-орієнтованим мовам, багато можливостей, яких позбавлені «класичні» мови програмування. Наприклад, складання сміття не реалізована в самому C#, а проводиться CLR для програм, написаних на C# так само, як це робиться для програм на VB.NET, J# та ін.

В реалізації широко використовуються структури даних (класи Collection). Розглянемо дану технологію (структурну одинцю) мови C#.

Інтерфейс є абстрактним типом посилання, що реалізується в класах програми. Членами інтерфейсу можуть бути методи, властивості, події та індексатори. Оскільки інтерфейси є абстрактними типами, їх члени потребують реалізації, яка може здійснюватися у класах та структурах. Список базових інтерфейсів класу міститься за ім'ям його базового класу.

Для того щоб дізнатися які інтерфейси реалізує клас System.Array можна побудувати його діаграму (рис. 2.3)

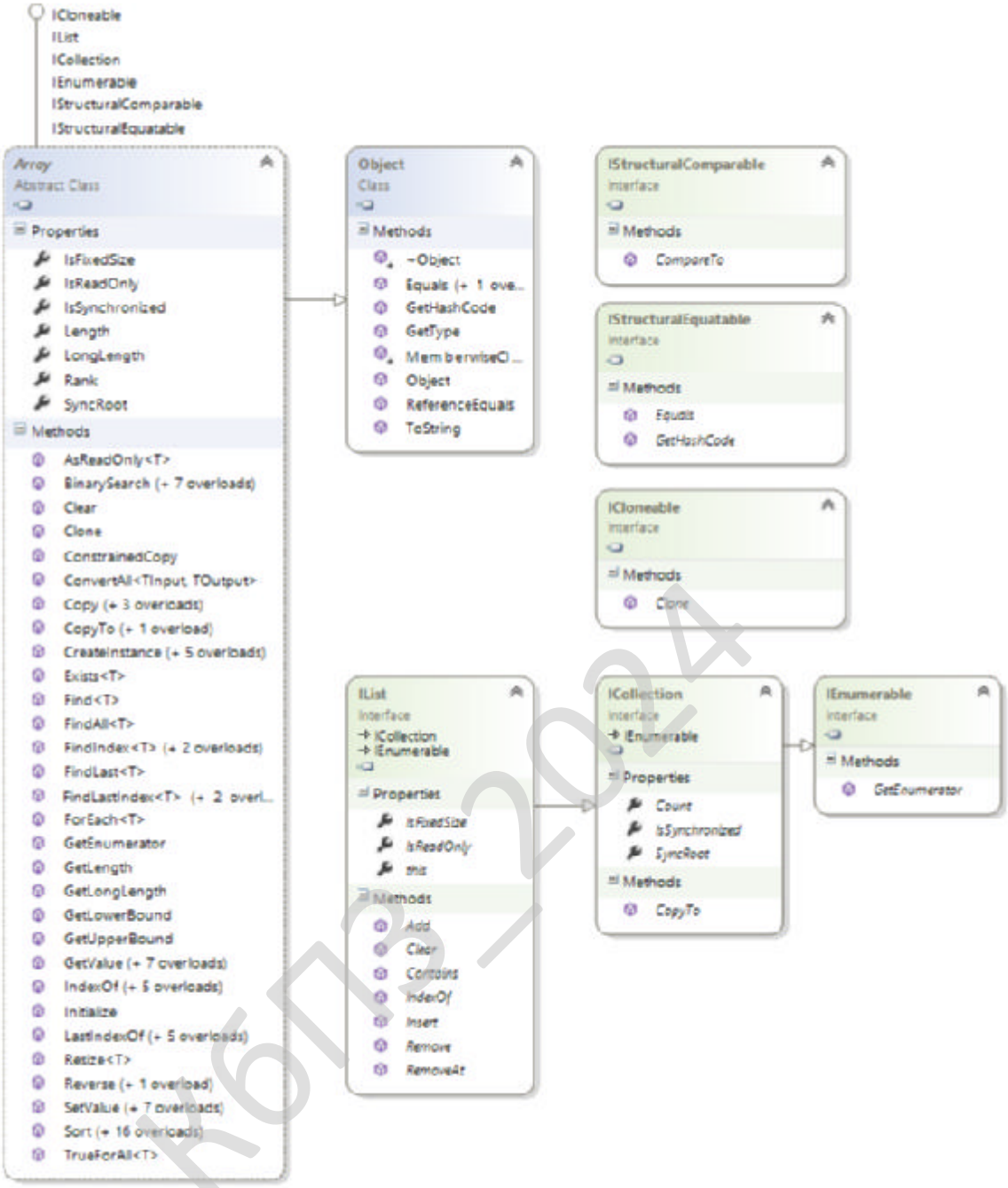


Рисунок 2.3 – Діаграма класу System.Array

Як видно з рис. 5 клас реалізує цілу низку інтерфейсів, наочно список інтерфейсів класу можна отримати, написавши наступний код:

```
var v = new int[] {1, 2, 3};
var t = v.GetType();
var i = t.GetInterfaces();
foreach(var tp in i)
Console.WriteLine(tp.Name);
```

Після запуску цей код видасть список, який повторює список, наведений на рис. 2.3:

```
ICloneable
IList
ICollection
IEnumerable
IStructuralComparable
IStructuralEquatable
IList`1
ICollection`1
IEnumerable`1
IReadOnlyList`1
ReadOnlyCollection`1
```

Застосування інтерфейсів дуже різноманітне, найчастіше до них вдаються у складніших нетривіальних ситуаціях.

Наприклад можна розглянути таку ситуацію. Клас System.Array має метод Sort що дозволяє сортувати масив упорядковуючи елементи зростання. Але цей метод працює тільки з примітивними типами даних: int, double, char, string та ін. У разі складних об'єктів потрібно використовувати інтерфейс IComparable, який дозволяє визначити єдиний метод:

```
int CompareTo(object o);
```

Цей метод і задаватиме правило порівняння складних об'єктів: поточного об'єкта та об'єкта переданого як параметр. Метод повертає значення:

більше нуля – поточний об'єкт розміщується після об'єкта, переданого як параметр (поточний більше);

рівне нулю - поточний об'єкт дорівнює об'єкту переданого як параметр, об'єкти розміщуються в порядку прямування;

менше нуля – поточний об'єкт розміщується перед об'єктом переданим як параметр (поточний менший).

2.3 Розгорнута постановка завдання

В якості програмної реалізації в роботі створюється захищений файловий менеджер з шифруанням файлів «на льоту». Процеси шифрування/дешифруваня приховані від користувача і відбуваються в тіньовому режимі. Тобто ці вимоги варто віднести до нефункціональних вимог.

Розглянемо основні функціональні вимоги до ФМ.

Файловий менеджер - це програмне забезпечення, яке дозволяє користувачам керувати файлами та папками на комп'ютері. Основні характеристики файлового менеджера можуть включати наступні:

1. Навігація по файловій системі: Файловий менеджер дозволяє користувачам переглядати файли та папки, що знаходяться на комп'ютері. Користувачі можуть переглядати різні директорії та переміщуватися між ними, відкривати та закривати їх.

2. Копіювання, переміщення та видалення файлів: дозволяє користувачам копіювати, переміщувати та видаляти файли та папки на комп'ютері.

3. Пошук файлів: Файловий менеджер дозволяє користувачам шукати файли на комп'ютері, використовуючи різні критерії, такі як ім'я файлу, розмір, дата створення, дата зміни та інші.

4. Сортування файлів: Файловий менеджер дозволяє користувачам сортувати файли та папки за різними критеріями, такими як алфавітний порядок, розмір, дата створення та дата зміни.

5. Редагування файлів: Файловий менеджер дозволяє користувачам відкривати та редагувати тексти, зображення, відео та інші файли, що підтримуються на комп'ютері.

6. Створення нових папок та файлів: Файловий менеджер дозволяє користувачам створювати нові папки та файли на комп'ютері, які вони можуть використовувати для зберігання файлів.

З огляду на вище вказаний перелік пошук файлів за певними критеріями є однією з основних задач файлового менеджера. Виберемо цей напрямок для більш детального дослідження та практичної реалізації.

Пошук даних на жорсткому диску суттєво активно впливає на швидкість, ефективність і комфортність користувача у користуванні ОС. З кожним роком жорсткі диски мають все більший обсяг пам'яті, тобто зростає кількість файлів, яку вони зберігають. У зв'язку з цим виникає потреба швидкого пошуку файлів на жорсткому диску. Для полегшення та прискорення цієї процедури використовують пошукові програми. Природно сучасні операційні системи мають обслуговування пошуку файлів, також є маса стороннього ПЗ (файлові менеджери), які так само надають подібний сервіс. У той же час іноді виникає необхідність інтеграції подібного сервісу у власні програмні продукти, і використовувати готові сторонні програми в такому випадку не дуже зручно.

Реалізація повнофункціонального менеджера досить складна і об’ємна задача. В якості практичної складової зосередимось на одній з основних функцій файлового менеджера – пошук файлів.

В практичній частині необхідно реалізувати модуль - пошук файлів з використанням різних критеріїв або їх комбінацій.

Завдання на реалізацію можна сформулювати наступним чином.

В якості практичної частини роботи реалізується програма, що дозволяє знайти на вказаному жорсткому диску (наприклад, С:) файл (файли):

- за заданим ім'ям (або шаблоном, що містить символи '?' і '*');

- за датою створення (вказується початкова та кінцева дати для пошуку);

- за розміром файлу (вказується мінімальний та максимальний розмір файлу в байтах).

Інформація про знайдені файли, що містить їх специфікацію (ім'я диска, шлях до файлу та його ім'я), дату створення та розмір (в байтах) має бути виведена на екран монітора.

Програма повинна визначити сумарний об'єм, який займає всі знайдені файли і вивести отримане значення на екран монітора.

Орієнтовний макет інтерфейсу представлено на рис. 2.4

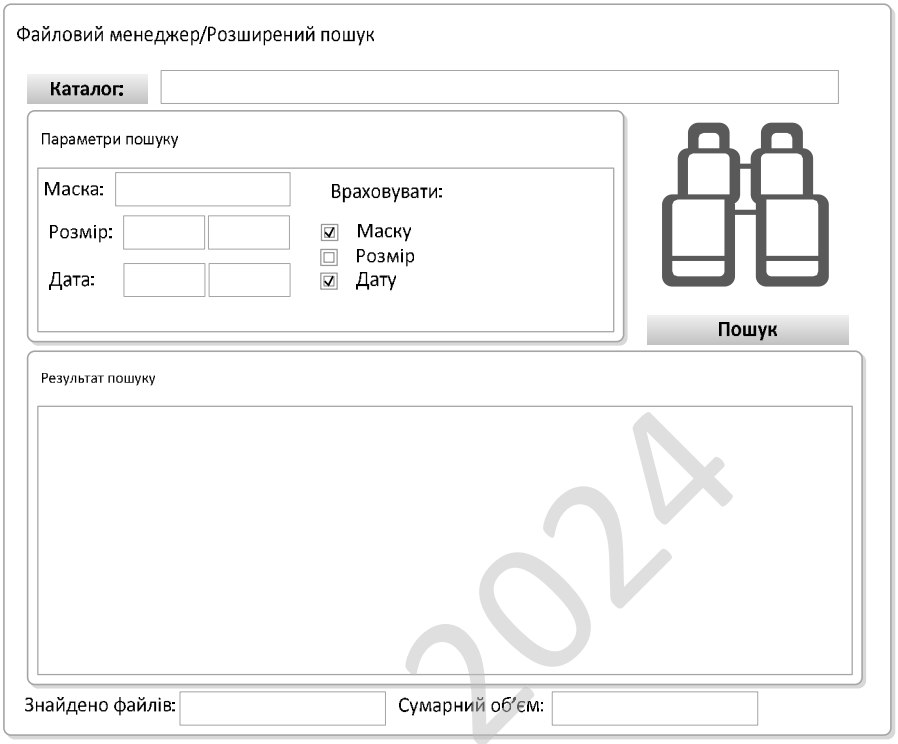


Рисунок 2.4 – Макет інтерфейсу

Як видно з макету, програма повинна мати три основні зони:

1) Вибір каталогу для пошуку

2) Налаштування параметрів та критеріїв пошуку

3) Список знайдених файлів (результат пошуку)

Підрахунок сумарного об’єму знайдених файлів. Реалізується адитивним накопиченням – отримавши новий файл, що відповідає критеріям, визначаємо його об’єм та додаємо до значення сумарного об’єму.

Для реалізації проекту обрана мова C# та середовище VisualStudio, це потужні сучасні інструменти, що дозволяють вирішувати завдання у широкому спектрі практично будь-якої спрямованості та складності.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Основний акцент в функціонуванні ситеми – захист даних.

Якщо брати повний перелік способів, якими може бути викрадена комерційна таємниця підприємства, це:

• хмарні сховища, електронна пошта, соціальні мережі та месенджери;

• носії інформації, що підключаються (у тому числі USB накопичувачі, телефони, DVD-ROM);

• друк документів на принтер;

• фізична крадіжка обладнання (стаціонарних дисків комп'ютера або його повністю).

Для захисту інформації на носіях, що підключаються, традиційно є два підходи до вирішення питання:

• можна блокувати використання працівниками не потрібних для роботи пристроїв та носіїв інформації. Це скорочує кількість варіантів як забрати з робочого комп'ютера інформацію, а в більшості випадків, змушує працівника взагалі відмовитися від цієї ідеї.

• якщо використання зовнішніх накопичувачів необхідно для роботи співробітника, потрібно контролювати усе, що він копіює і своєчасно виявляти потенційні витоку інформації.

Варіант із блокуванням менш трудомісткий, однак, застосовувані засоби не повинні завдавати шкоди виробничому процесу, тобто не гальмувати роботу комп'ютера та мережі, не порушувати рух інформації в компанії, зберігати працездатність мишки, клавіатури, сканерів та принтерів, веб-камери.

Додаткові завдання, які при цьому з'являються:

• як дозволяти підключати певні накопичувачі, а інші блокувати (наприклад, щоб звичайні співробітники не могли використовувати флешки на комп'ютері, а співробітника IT відділу міг підключити флешку до будь-якого комп'ютера)

• або певні пристрої не блокувати, а обмежувати доступ до файлів на них.

Обмеження доступу до USB портів

Вимкнути пристрої (у тому числі USB-порти) фізично на комп'ютерах.

Плюси:

• вирішує проблему блокування портів;

Мінуси:

• станеться повне відключення портів, що не дасть використовувати потрібні для роботи пристрою USB (веб камеру, наприклад або USB принтер).

• якщо відключати порти фізично, доведеться розбирати системний блок.

Вимкнути через налаштування BIOS або системи.

В тому числі:

• Вимкнення USB портів через налаштування BIOS

• Увімкнення та вимкнення USB-накопичувачів за допомогою редактора реєстру

• Вимкнення USB портів у диспетчері пристроїв

Спрацює надійно, але це найнезручніший із програмних способів. Крім того, це відключить навіть необхідні для роботи USB пристрою. Підійде якщо комп'ютерів всього пара штук, змінювати налаштування не знадобиться і USB пристрою ніколи не знадобляться.

Технічно робиться наступне: зайти в BIOS, відключити всі пункти, пов'язані з контролером USB (наприклад, USB Controller або Legacy USB Support), зберегти зміни.

Вимкнення USB накопичувачів через редактор реєстру.

Зручніше попереднього способу тим, що можна відключити USB накопичувачі, але при цьому залишити увімкненими принтери, сканери, мишки і т.д.

Мінуси:

• спрацьовує не завжди (при підключенні нової флешки для неї буде перевстановлено драйвер і блокування не спрацює);

• користувач з правами адміністратора легко увімкне пристрій назад;

• не можна адресно дозволити підключення певних носіїв.

Технічна реалізація способу:

• зайти до редактора реєстру (regedit.exe);

• в реєстрі перейти на гілку:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\USBSTOR;

• для ключа реєстру «Start» задайте значення «4», щоб заблокувати доступ до USB накопичувачів. Щоб дозволити назад, змініть значення знову на "3".

Вимкнення USB портів у диспетчері пристроїв.

Також спрацьовує не завжди. Може не застосовуватись для нових підключених пристроїв. Користувач з правами адміністратора легко ввімкне все назад і відновить доступ.

Мінуси:

• блокування порту не спрацює на новий підключений пристрій, тобто доведеться постійно стежити за цим, заходити з правами адміністратора в систему і відключати пристрій в Диспетчері;

• користувач з правами адміністратора може увімкнути пристрої назад;

• неможливо встановити права на пристрій, тільки вимкнення;

• підійде лише адміністрування пари комп'ютерів, т.к. немає автоматизації.

Заборона доступу до знімних носіїв через групову політику (Active Directory). Це рішення допоможе обмежити користувачеві можливість використання флешок, не зачепивши корисних USB-пристроїв.

Плюси:

• можна заборонити разом підключення всіх флешок;

• гнучке налаштування підключених пристроїв до портів: залишити лише читання флешок, можливість редагувати тощо;

• забезпечує можливість централізованого адміністрування та управління ресурсами системи;

• централізоване управління правами доступу до інформації на основі функціональних груп та рівнів доступу.

Мінуси:

• потрібна допомога фахівця або адміністратора для налаштування, який не завжди є в штаті компанії;

• не у всіх компаніях застосовується Active Directory;

• не вдасться блокувати-дозволяти конкретні usb накопичувачі за серійними номерами.

Обмеження доступу до USB накопичувачів засобами антивірусів

Цей метод підходить не для всіх антивірусних програм (не всі мають такі функції).

Плюси:

• у корпоративних версіях антивірусів така можливість може бути вже в комплекті;

• як правило, є дистанційне централізоване управління.

Мінуси:

• у маленьких компаніях рідко використовують дорогі корпоративні версії антивірусів;

• не у всіх антивірусах є можливість обмеження пристроїв (дивіться в налаштуваннях рішення, що використовується у вас);

• потрібне залучення IT-фахівця;

• ви будете прив'язані до певної антивірусної програми і не так просто її змінити, якщо в цьому виникне потреба. А ліцензію на антивірус треба продовжувати щороку.

Крім того, не у всіх антивірусах можна реалізувати метод, коли треба не просто вимкнути пристрій, а заблокувати певні пристрої за серійними номерами.

Або коли не треба відключати, наприклад, флешку, а залишити її в режимі “тільки для читання”.

Також, жоден із перерахованих способів не вирішує завдання, коли не можна відключати пристрої (потрібні для роботи), але необхідно знати, що на них копіюється, щоб не допустити витоку важливої корпоративної інформації.

Використання спеціальних програм для блокування флеш-накопичувачів.

Застосування цих програм різноманітне: від контролю пристроїв до контролю над усіма маніпуляціями з комп'ютером.

Плюси:

• Гнучка настройка блокування пристроїв (як усіх накопичувачів, так і адресне блокування),

• Розмежування доступу до файлів на пристрої (режим лише читання).

• Контролює файли, що копіюються на пристрої.

Мінуси:

• Потрібно придбати ліцензію для роботи програми

Розглянуті способи досить не зручні та потреують постійного втручання адміністратора сисстеми для переналаштувань з метою забезпечення захисту даних.

Програмний комплекс, що розробляється в рамках бакалаврської роботи, передбачає взаємодію з файлами, що досить схожа з звичайною взаємодією в ОС через провідник. Але программа передбачає тіньову функцію шифрування файлів «на льоту». Тобто користувач працює з файлами через спеціальний інтерфейс, що досить подібний до звичайного файлового менеджера, з іншого боку – доступ до файлів через стандартний провідник не буде можливим, бо файли зашифровані.

В рамках програми, що розробляється з використанням технологій захисту даних на зовнішніх пристроях було розроблено функціонал, вказаний у технічному завданні.

3.2 Розробка структурної схеми

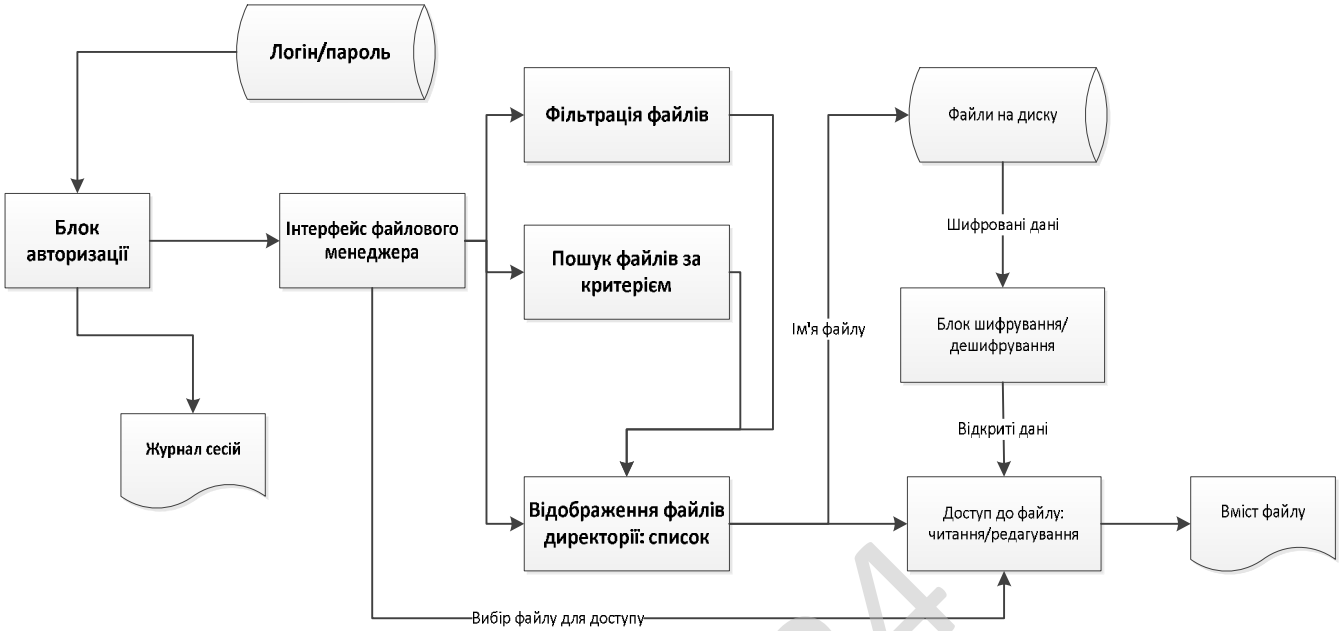


Рисунок 3.1 – Структурна схема программного комплексу

Як видно з структурної схеми программна система включає три основних блоки:

1) Блок авторизації. Передбачає парольну аутентифікацію користувача. Що дозволяє обменжити доступ та фіксувати в журналі сесії роботи користувачів

2) Блок функціональної взаємодії з файлами. Включає інтерфейс для взаємодії та функції пошуку й фільтрації. Що дозволяє пришвидшити роботу та полегшити процеси відшукання потрібного файлу серед інших файлів активної директорії

3) Блок шифрування/дешифруання. Блок відповідає за реалізацію алгоритму шифрування даних. Шифруванні відбувається приховано від користувача в момент звернення до файлу. На диску файл зберігається в шифрованому вигляді, в момент звернення файл дешифрується в рперативну пам’ять та відобажається користувачеві для перегляду. В момент завершення роботи з файлом данні знову шифруються та зберігаються на диск.

3.3 Розробка функціональної схеми

Функціональна схема програмного забезпечення (Software Architecture Diagram) є графічним зображенням компонентів програмної системи та їх взаємодії. Це важливий інструмент для розробників, архітекторів та інших зацікавлених сторін для розуміння структури, поведінки та взаємозв'язків компонентів програмного забезпечення.

Основні елементи функціональної схеми

1. **Компоненти (Components)**: Основні частини системи, які виконують конкретні функції. Це можуть бути модулі, сервіси, підсистеми або бібліотеки.
2. **Взаємодія (Interactions)**: Способи, якими компоненти взаємодіють один з одним. Це можуть бути виклики методів, повідомлення, запити через API тощо.
3. **Інтерфейси (Interfaces)**: Точки доступу до компонентів, через які відбувається взаємодія з іншими компонентами.
4. **Користувачі (Users)**: Кінцеві користувачі або інші системи, які взаємодіють з програмним забезпеченням.
5. **Дані (Data)**: Потоки даних між компонентами, включаючи збереження, передачу та обробку даних.

Для нашого конкретного випадку системи можна запропонувати наступну деталізацію.

Перед розробкою системи для спрощення подальшої роботи були створено різні діаграми та інформаційна модель даного проекту.

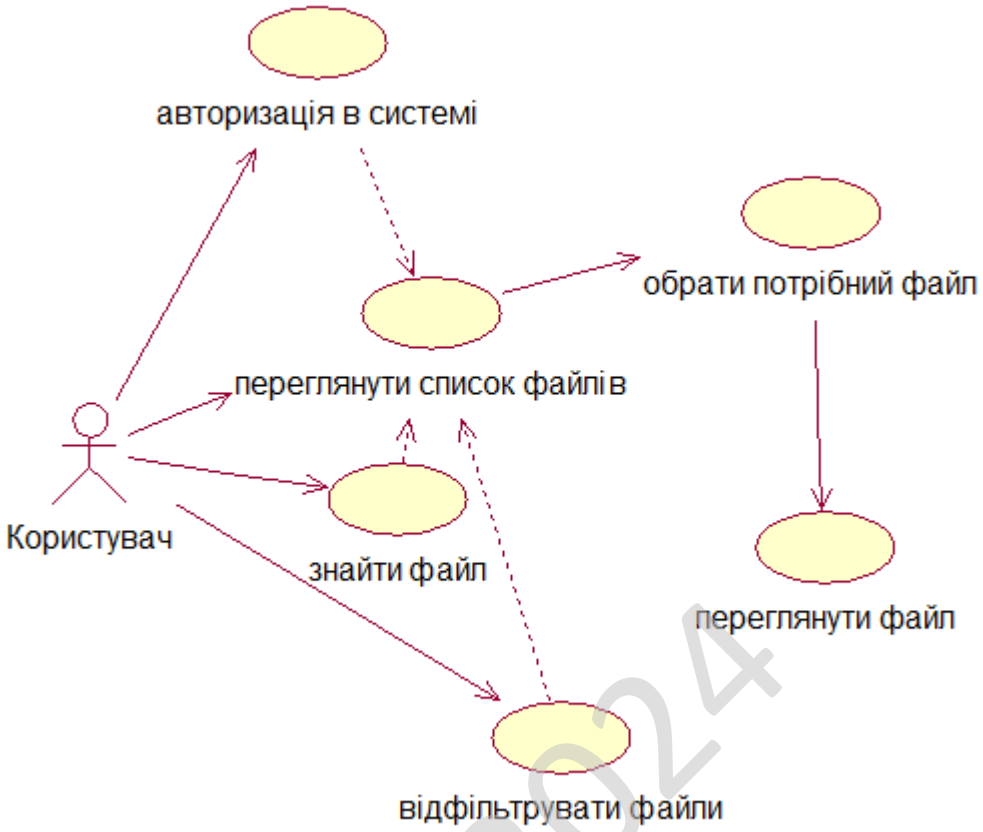


Рисунок 3.2 – UML діаграма використання (UseCase)

На цій діаграмі зображені можливості користувача програми.

Користувач може взаємодіяти з файлами. Попередньо користувач має авторизуватись. Програма надає стандартний набір функцій пошукук та фільтрації. Для пошуку і фільтрації застосовуються критерії: ім’я (чи частина) файлу, розширення (тип файлу), дата створення, тощо. Для перегляду вмісту файлу користувач має обрати файл в списку, вікиристовуючи інтерфейс програми.

Механізми взаємодії користувача з програмою (сценарій) представлено на рис. 3.3

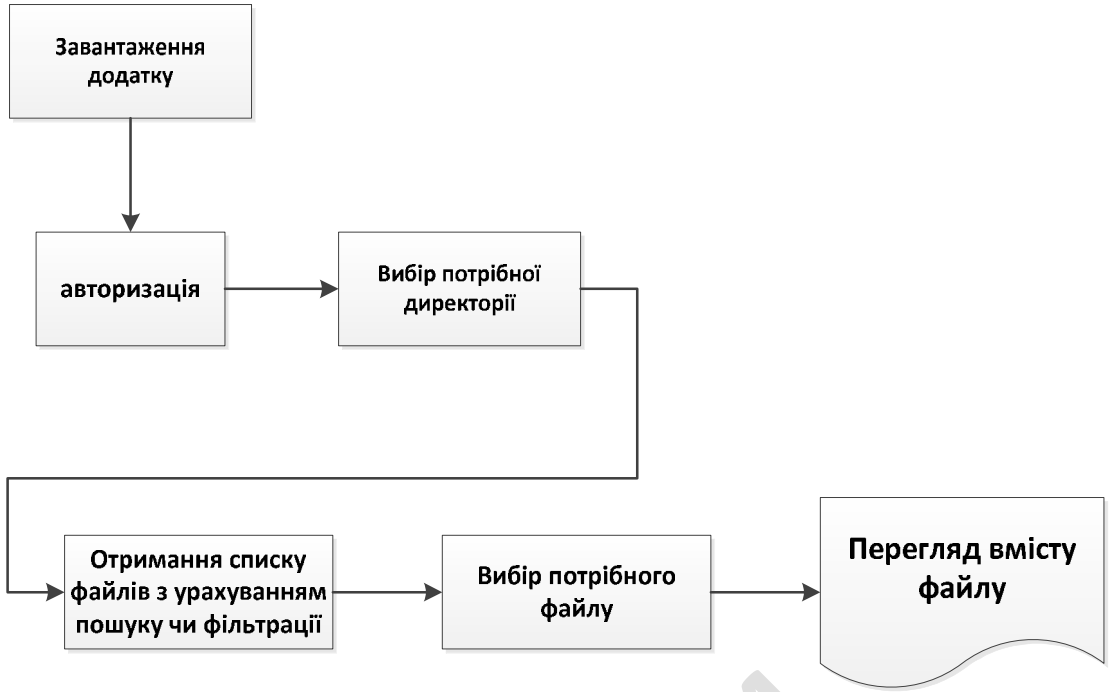


Рисунок 3.3 – Послідовність дій користувача

Наступним кроком було розроблено функціонально-сруктурну модель програми.

Функціональна модель дозволяє відобразити функціональну структуру системи. Функціональна модель системи першого рівня зображена на рис. 3.4

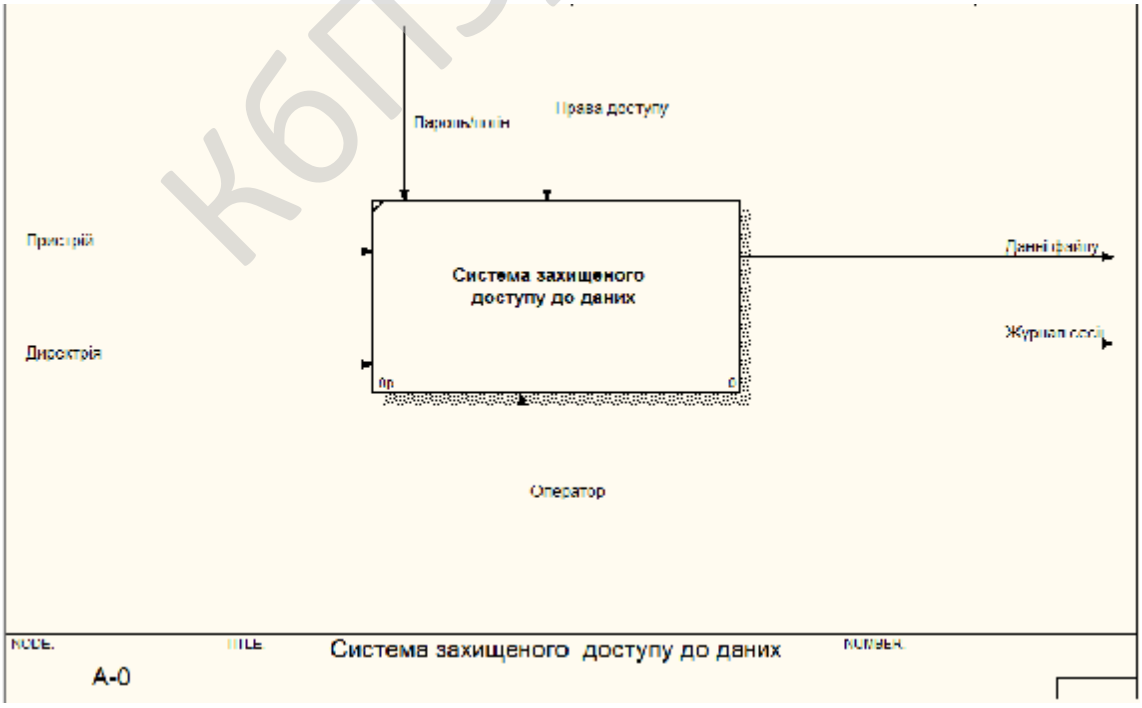


Рисунок 3.4 – Функціональна модель нульового рівня (контекстна діаграма)

Як вхід в основний блок виступають данні про пристрій, що дозволяє вибрати один з активних підключених носіїв даних та конкретна директорія на пристрої. Керуючими управлінськими засобами виступають права доступу, що визначаються через парольну аутентифікацію. Механізмом нашої моделі є користувач. На виході маємо вміст файлу для перегляду та журнал сесії користувача.

Далі розглянемо функціональну модель системи першого рівня декомпозиції, наведену на рис. 3.5

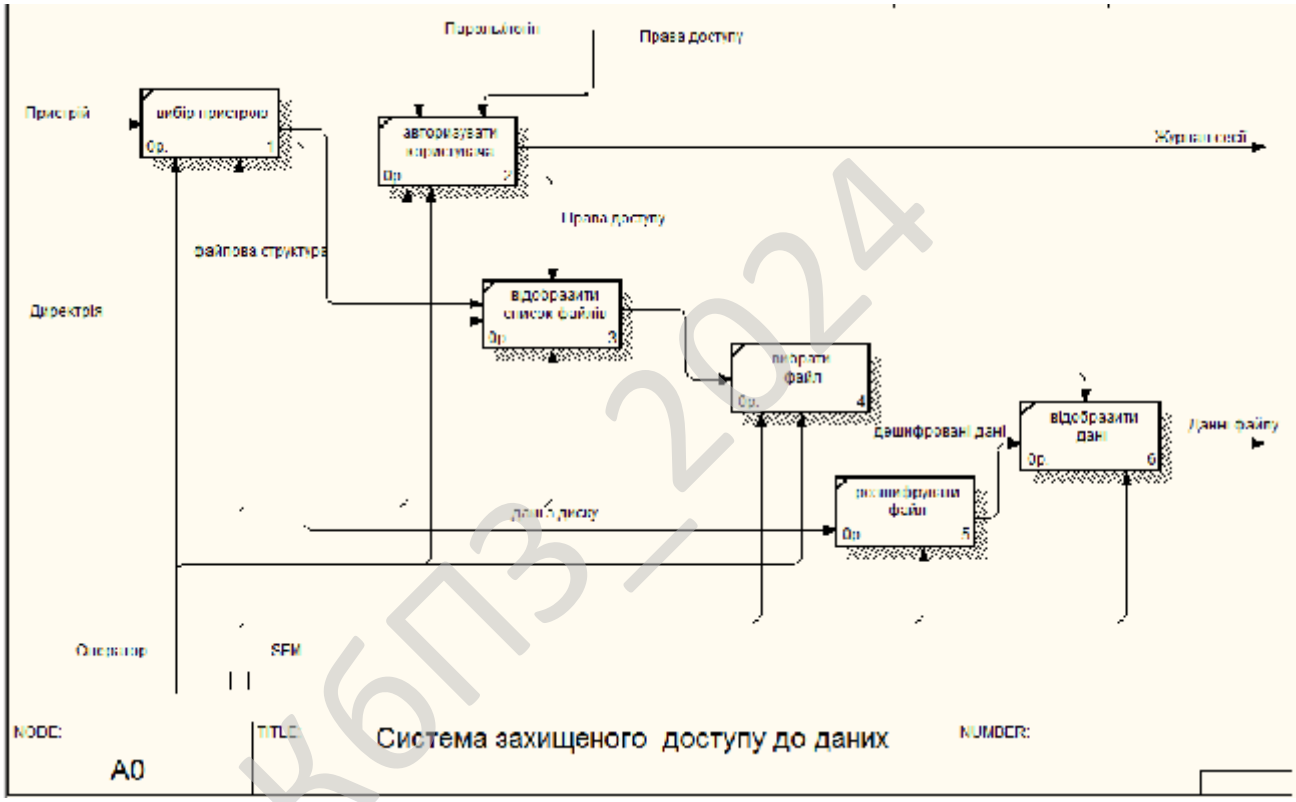


Рисунок 3.5 – Функціональна модель, декомпозиція першого рівня

На даній моделі представлені блоки, що відповідають за окремі функції. База даних та додаток для захисту даних на зовнішніх накопичувачах, що виступає в ролі інтерфейсу користувача. Користувач надсилає запит до програми для здоступу до файлу на зовнішньому накопичувачі. Додаток звертається до бази даних далі отримує з бази даних необхідні дані для автентифікації користувача або ж забороняє користувачеві доступ до носію. Також якщо носій не захищений, то

додаток пропонує користувачу внести ці дані ці дані до бази даних за допомогою свого інтерфейсу.

3.4 Розробка діаграми процесів

Доцільним буде розглянути послідовність виконання та взаємозв'язок окремих процесів в ході захищеного документообігу. Розглянемо ці процеси для повного сценарію використання програми.

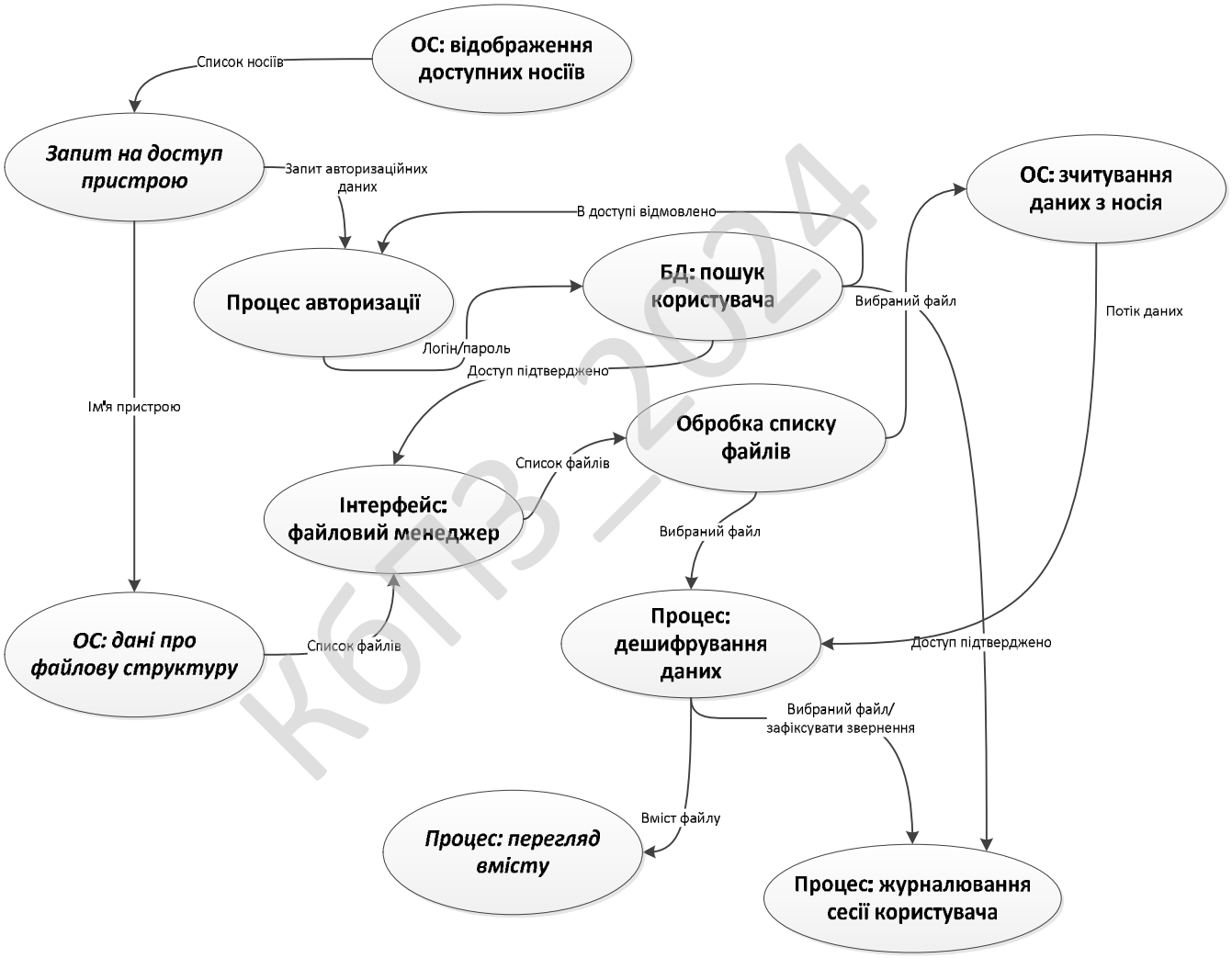


Рисунок 3.6 – Діаграма процесів

На цій діаграмі представлені проміжні процеси, що виникають в ході роботи програми захищенного доступу до даних на зовнішніх носіях. Діаграма досить детально описує та ілюструє хід виконання процесів.

Процеси розподілено між різними виконавцями (з точки зору механізмів та технолгій).

Операційна система: забезпечує відображення доступних носіїв, їх файлову структуру, безпосередні дані з диску.

База даних: забезпечує авторизацію користувача та зберігає журнал сесій.

Інші процеси забезпечуються програмною ситемою – захищений файловий менеджер (SFM).

Вся програма складається з ряду модулів і підмодулів, які здійснюють виконання задач, поставлених перед системою користувачем. Це істотно спрощує обіг із програмою, робить її менш складною, і більш зрозумілою для оператора.

Такий підхід також дозволяє масштабувати програму та розширювати її функціонал за потреби.

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

Сьогодні вже досить рідко програми реалізуються на низькому рівні, в більшості випадків використовується високорівневе конструювання з використанням готового набору базових рішень.

Допоміжні інструменти можуть мати різні назви (бібліотеки, фреймворки, конструктори), залежно від напряму, типу завдання, основного інструментарію. Але сутність від цього не змінюється. Програмування такого роду іноді називають **programming block building**.

Суть і переваги даного методу полягають у тому, що розробник не витрачає час на реалізацію стандартних завдань, а займається логікою застосування, розробляє унікальний функціонал, творчо розвиває проект. З іншого боку, такі підходи диктує час і ринок. Основною вимогою ринку є швидкість та низька собівартість розробки. Згаданий вище підхід задовольняє ці вимоги, тому розвивається і культивується у сфері розробки ПЗ.

У вибраному інструментарії існує низка методів, які можна використовувати для реалізації поставленого завдання. У цілому нині метод та її структуру добре ілюструє структурна схема, представлена на рис. 4.1.

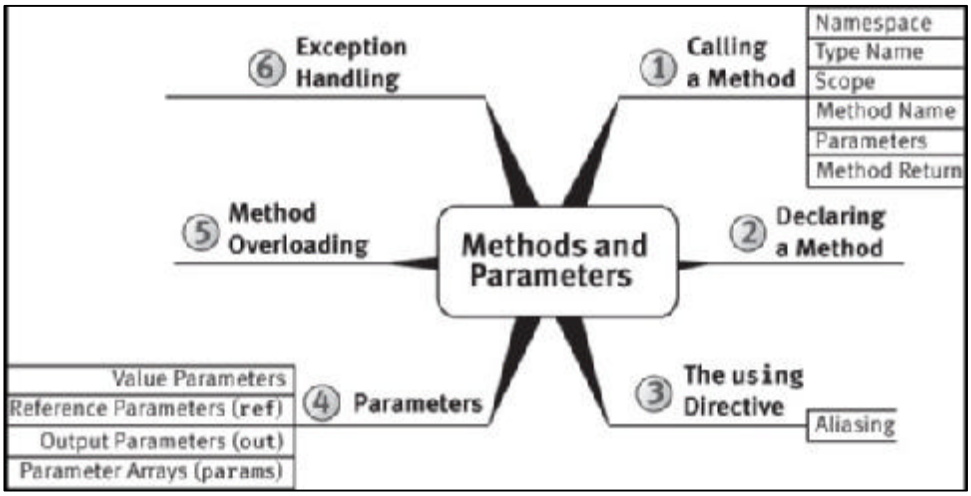


Рисунок 4.1 – Структура методу [18]

Після проектування можна перейти безпосередньо до програмування.

Структурно програма має два достатньо окремих блоки:

1) Блок забезпечення функцій роботи з списком файлів

2) Блок шифрування/дешифруавння

Блок шифрування/дешифруавння

Блок шифрування/дешифруавння містить 3 основні функції:

1) Генерування таблиці підстановки – відповідає за створення підстановочної таблиці для подальшого використання у процесах шифрування/дешифрування. Основна особливість таблиці – відсутність повторів, тобто кожен елемент таблиці має бути унікальним. Ця умова забезпечує однозначність представлення зашифрованого тексту та оборотність процесу. Отримувати таблицю підстановки можна різними способами. У нашому випадку вона генерується випадковим вибором всіх видимих символів таблиці ASCII. У процесі генерації перевіряється відсутність повторів, а генератор залежить від ключової фрази. Це реалізовано так:

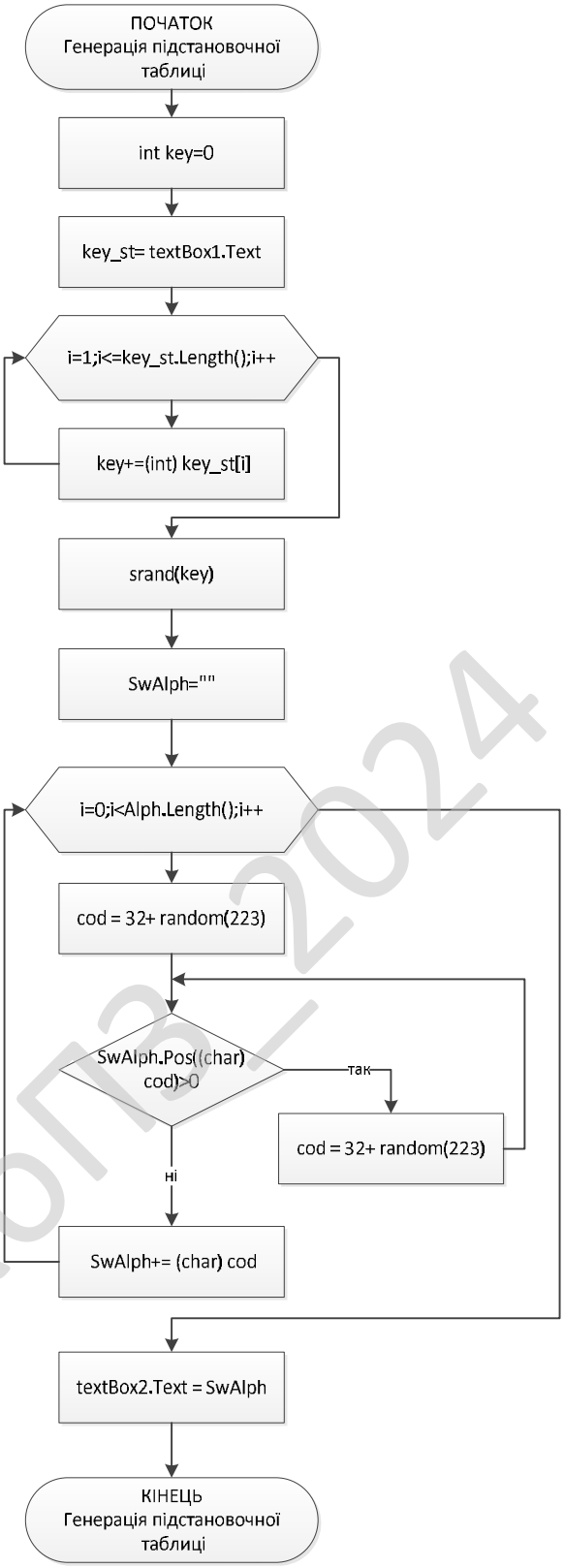


Рисунок 4.2 – Блок-схема «Генерація таблиці підстановки»

Код реалізації функції з коментарями:

```
/генерація підстановочної таблиці
        private void button1_Click(object sender, EventArgs e)
```

```
{
    int cod;
    int key = 0;
    string key_st = textBox1.Text;//отримати ключ

    for (int i = 0; i < key_st.Length; i++)
        key += (int)key_st[i];
    Random rnd = new Random(key);
    SwAlph = "";
    // Генерація таблиці (набір символів для підстановки)
    for (int i = 0; i < Alph.Length; i++)
    {

        cod = 32 + rnd.Next(223);
        while (SwAlph.IndexOf((char)cod) >= 0)
        {
            cod = 32 + rnd.Next(223);
        }
        SwAlph += (char)cod;
    }
    // вивести отриману таблицю
    textBox2.Text = SwAlph;
  }
```

2) Функція шифрування – фактично реалізує заміну символу відкритого тексту, замінюючи його відповідним за індексом символом з таблиці підстановки. Блок-схему алгоритму шифрування наведено на рис. 4.3

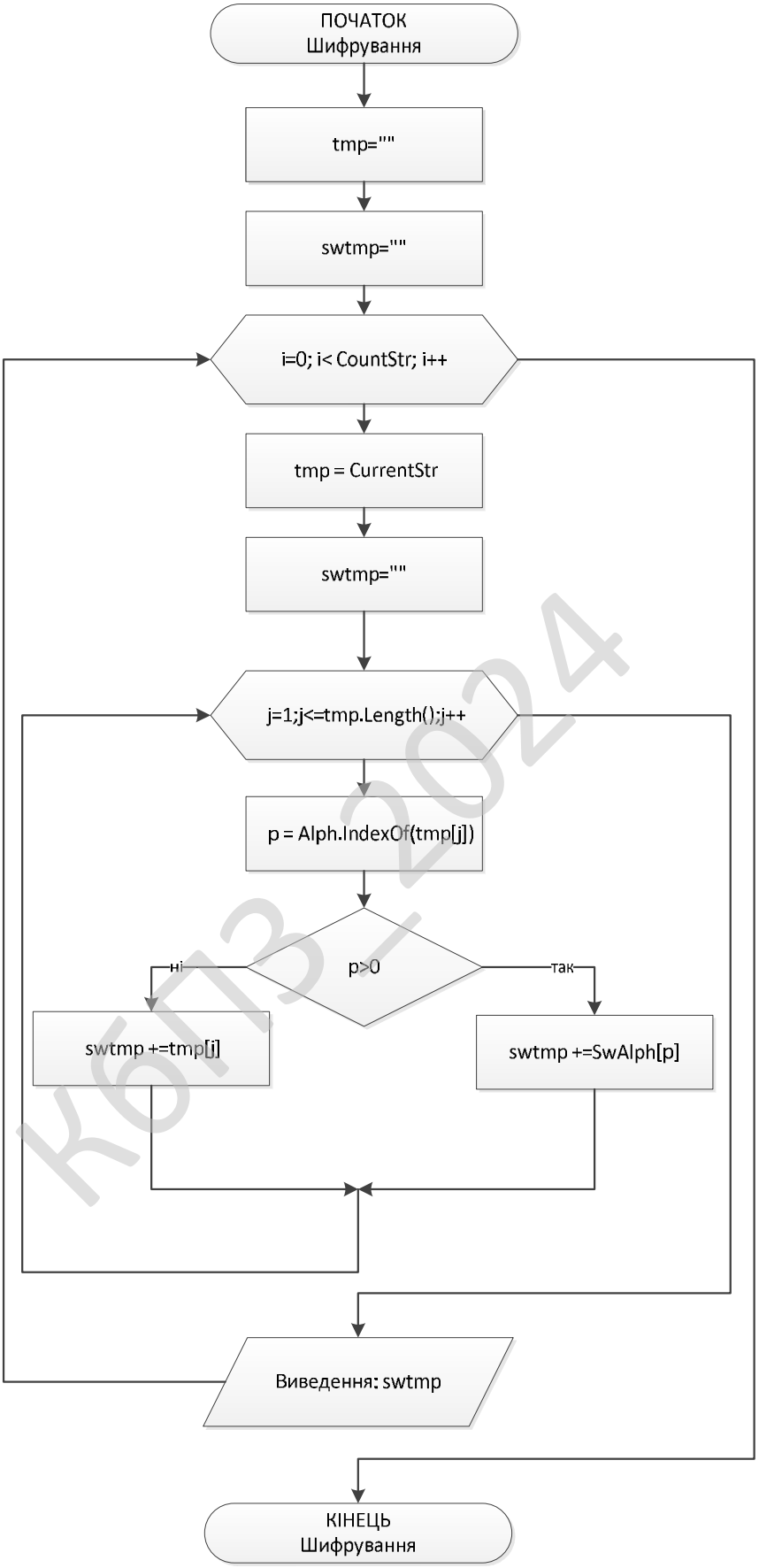


Рисунок 4.3 – Блок-схема "Шифрування"

```
   // функція шифрування
        private void button4_Click(object sender, EventArgs e)
        {
            string swtmp = "";
            int p;
            for (int i = 0; i <textBox3.Text.Length; i++)// прохід по всім символам
            {
              p = Alph.IndexOf(textBox3.Text[i]);// знайти входження (позицію)
символа в відкрит. алфавіті
              if (p >= 0)
               {
                 swtmp += SwAlph[p];// замінити на відповідний символ закритого
алфавіта
               }
              else
               {
                swtmp += textBox3.Text[i];// символи що не входять в Alph (осн.
алфавіт)
               }
            }
            textBox4.Text = swtmp;
              }
```

3) Функція дешифрування – виконує зворотний процес заміни символів: зіставляючи символи таблиці підстановки (закритий алфавіт) і замінює їх символами відкритого алфавіту. Алгоритмічно процес аналогічний попередньому з точністю до алфавітів, що використовуються. Блок схема процесу матиме вигляд аналогічний рис. 4.3.

Блок забезпечення функцій роботи з списком файлів

Далі опишемо методи, які будуть використані для реалізації. Наступний матеріал взятий із відкритих джерел документації з мови C#[4].

Оскільки в реалізації будуть використовуватися ці методи, варто їх коротко описати

Directory.GetFiles Method

Визначення

Простір імен: System.IO

Assemblies: System.IO.FileSystem.dll, mscorlib.dll, netstandard.dll

Повертає імена файлів, які відповідають зазначеним критеріям.

Метод має кілька перевантажень

Таблиця 4.1 – Перевантаження метода

`GetFiles(String)`	Повертає імена файлів (із зазначенням шляху до них) у вказаному каталозі.
`GetFiles(String, String)`	Повертає імена файлів (включаючи шляхи) із зазначеного каталогу, які відповідають умовам заданого шаблону пошуку.
`GetFiles(String, String, SearchOption)`	Повертає імена файлів (включаючи шляхи) у заданому каталозі, що відповідають умовам шаблону пошуку, використовуючи значення, яке визначає, чи виконувати пошук у підкаталогах.

GetFiles(String)

Повертає імена файлів (із зазначенням шляху до них) у вказаному каталозі.

public static string[] GetFiles (string path);

Параметри:

path: String відносний чи абсолютний шлях до каталогу для пошуку. У цьому рядку не враховується регістр знаків.

Повернення:

String[] масив повних імен (включаючи шляхи) файлів у вказаному каталозі або порожній масив, якщо файли не знайдені.

Винятки:

IOException: сталася мережна помилка або помилка вводу/виводу.

UnauthorizedAccessException: у об'єкта, що викликає, відсутній необхідний дозвіл.

ArgumentException: path є рядком нульової довжини, що містить тільки пробіли або один або кілька неприпустимих символів. Ви можете запросити неприпустимі символи за допомогою методу GetInvalidPathChars().

ArgumentNullException: path - null.

PathTooLongException: вказаний шлях, ім'я файлу або обидва значення перевищують максимальну довжину, задану в системі.

DirectoryNotFoundException: Вказаний шлях не знайдений або неприпустимий (наприклад, веде на неіснуючий диск).

EnumerateFiles та **GetFiles** методи мають такі відмінності: При використанні EnumerateFiles, можна запустити виконання перерахування колекції імен перед поверненням усієї колекції; При використанні GetFiles необхідно дочекатися весь масив імен, що повертається, щоб можна було відкрити масив. Таким чином, при роботі з багатьма файлами і папками, EnumerateFiles може виявитися більш ефективним.

За всіх переваг використання готових методів та побудови рішення на їх основі цей підхід має і суттєвий недолік. Недолік виявляється у проблемах та особливостях реалізації методу.

Так реалізація методу **EnumerateFiles** в .Net має таку проблему:

При спробі входу до прихованих системних папок (доступ до яких заборонено користувачеві, а для деяких і адміністратору системи) виникає помилка і програма «падає».

Простий підхід обертання методу в:

```
try
{}
catch
{}
```

не дає очікуваних результатів. Програма перестає «падати», оскільки передбачена реакція на винятки, але й очікуваних результатів не дає. При спробі звернутися до "закритої папки" виникає виняткова ситуація і метод повертає порожній список.

Якщо розглядати як параметри лише користувацькі (конкретно вказані) каталоги, то проблем не виникає, але якщо передавати як параметр кореневий диск (при цьому не важливо чи це буде системний диск, так як і на несистемних дисках, такі закриті каталоги присутні) просте використання методу не підходить.

Вивчивши документацію та досвід інших розробників, було знайдено вихід – використовувати комбінацію методів GetDirectories та GetFiles з обгорткою try{} catch {} та перевіркою винятків UnauthorizedAccessException, DirectoryNotFoundException

```
try
  {
     string[] subDirs= Directory.GetDirectories(currentDirPath);

  }
  catch (UnauthorizedAccessException)
  {
      continue;
  }
  catch (DirectoryNotFoundException)
  {
      continue;
 }
```

У ході тестів було виявлено, що такий підхід є найбільш вдалим і у всіх тестових запусках видавав очікувані коректні результати.

Під час роботи програми знадобиться також зручний метод отриманнярозміру файлу. Функція, що дозволяє отримати розмір файлу:

```
System.IO.FileInfo(findFile).Length
```

Ця функція повертає розмір у байтах, що дуже зручно з погляду виведення інформації для користувача. Необхідно реалізувати функцію перетворення розміру відповідні одиниці щодо розміру. Функція повинна приймати на вхід кількість байт (числове значення), а в результаті видавати рядкове значення розміру, перетворений на потрібні одиниці (байт, Кбт, Мбт, Гбт і т.д) і готове до виведення на форму.

Алгоритм такої функції досить простий, блок-схему алгоритму представлено на рис. 4.4.

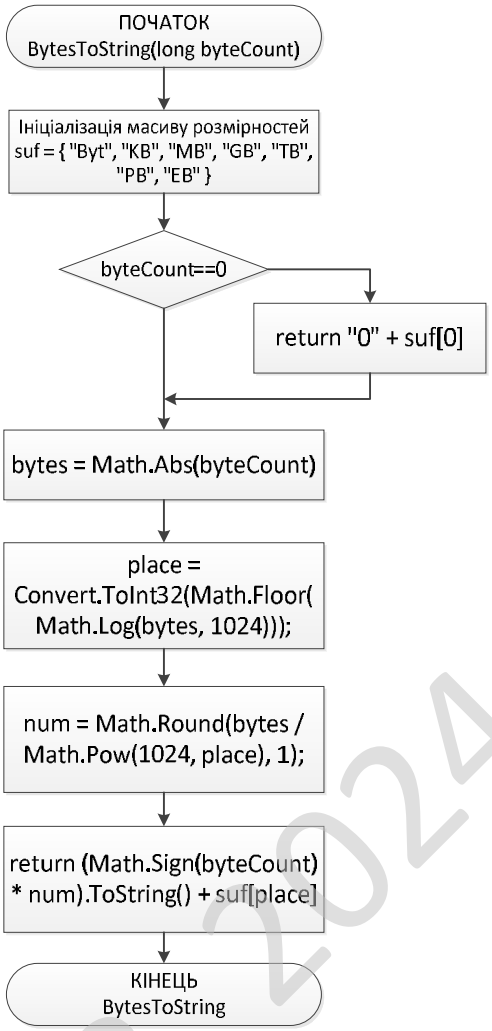


Рисунок 4.4 – Конвертація розмірності файлу

Реалізація цієї функції має вигляд:

```
static String BytesToString(long byteCount)
{
 string[] suf = {"Byt", "KB", "MB", "GB", "TB", "PB", "EB"};
//масив назв розмірностей
if (byteCount == 0) return "0" + suf [0];
long bytes = Math.Abs(byteCount);
int place = Convert.ToInt32(Math.Floor(Math.Log(bytes, 1024))));
double num = Math.Round(bytes / Math.Pow(1024, place), 1);
// Функція перетворення
 return (Math.Sign(byteCount) * num).ToString() + suf[place];
}
```

4.2 Реалізація окремих функцій ПЗ

Проектування інтерфейсу

При розробці проектів типу WinForms передбачається два основні етапи – проектування інтерфейсу та програмування алгоритмів та реакцій програми на події (користувач, система та ін.).

Оскільки вибрано тип програми WinForms, розробка програми починається з проектування інтерфейсу. Середовище надає велику кількість готових візуальних компонентів. У ході реалізації використовувалися лише стандартні компоненти, тому їх детальний опис немає сенсу. Обмежимося лише переліком використаних компонентів та їх призначенням у проекті.

TButton – кнопки управління процесом пошуку («Каталог» вибір каталогу для пошуку, «Шукати» активація процесу пошуку);

TLabel – виведення текстових повідомлень на форму (компонент використовується для підпису призначення полів та елементів інтерфейсу)

TNumericUpDown – компонент для вибору меж за розміром. Налаштування цього компонента показано на рис 4.5 Використання цього компонента зручно так як не вимагає додаткових перевірок інформації, що вводиться. Користувач просто не може ввести неприпустимі, непередбачені параметри

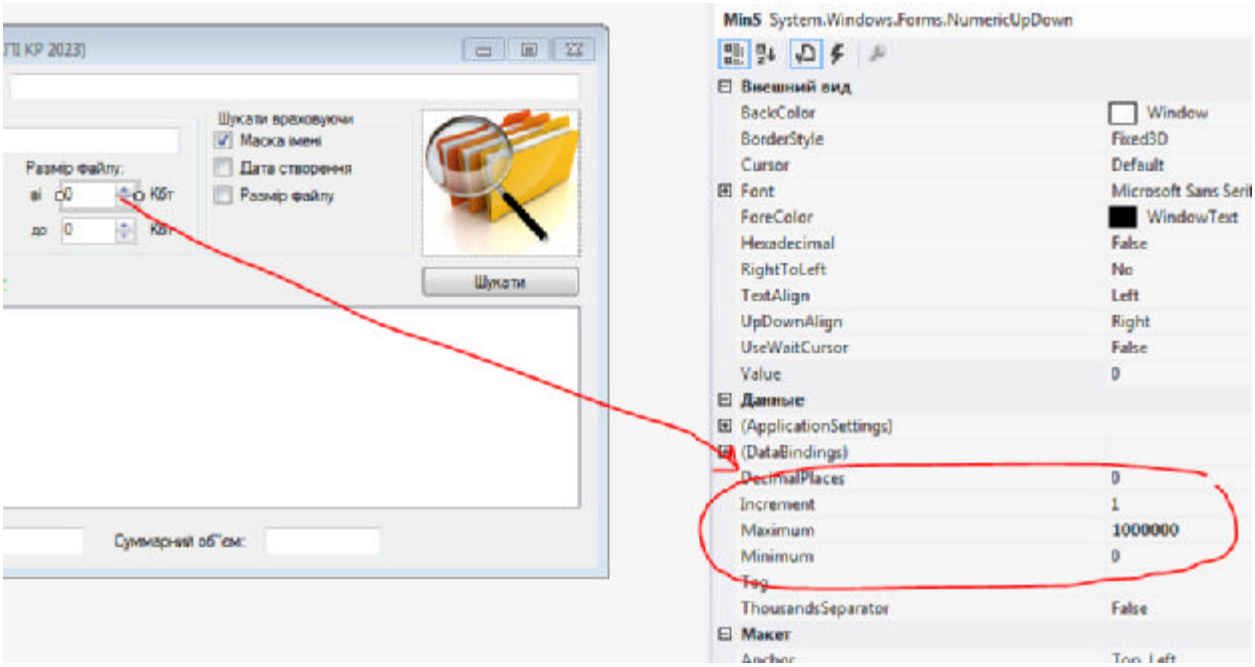


Рисунок 4.5 – Налаштування компонента, що задає межі розмірів

Повний макет форми (інтерфейсу) представлений на рис. 4.6

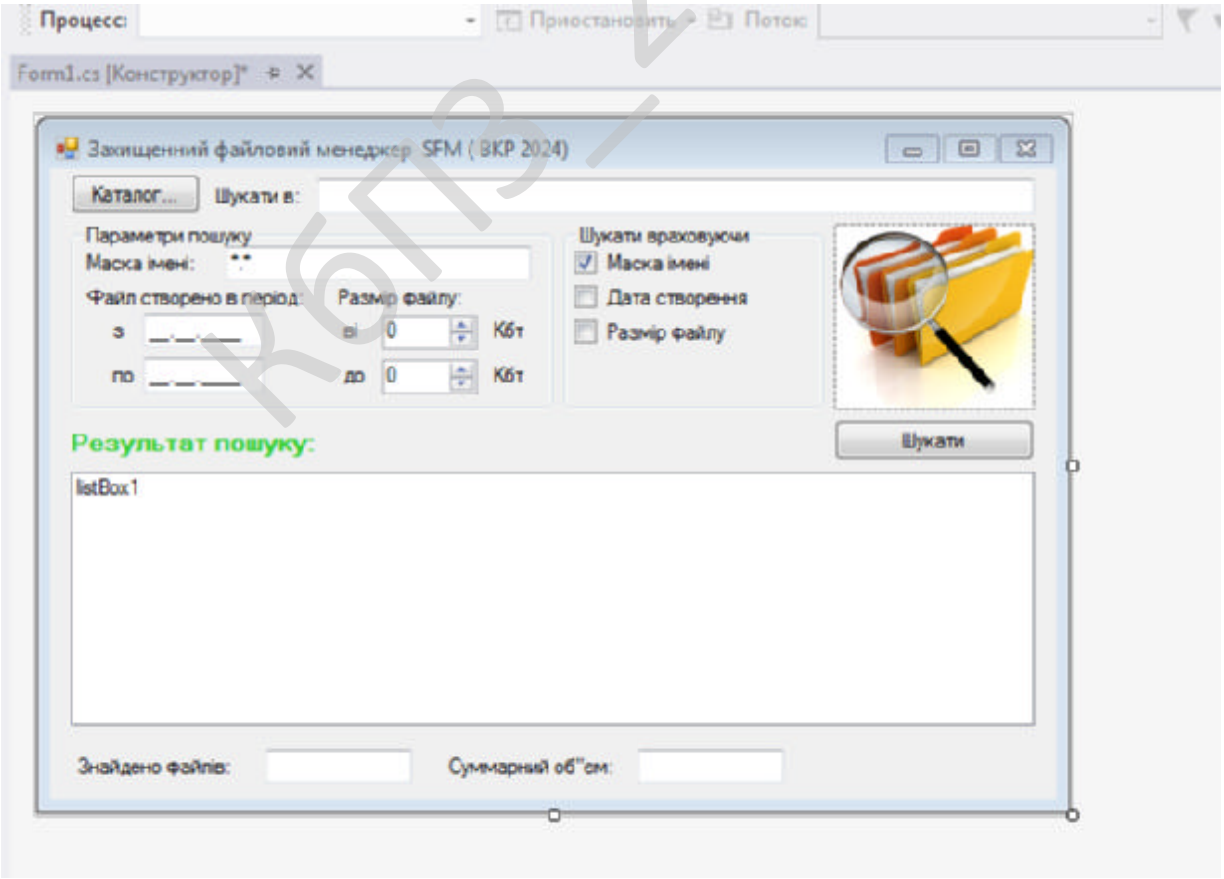


Рисунок 4.6 – Форма на етапі проектування

Як можна побачити на рис. 4.6, форма має два компоненти типу GroupBox. Перший для введення параметрів пошуку, другий з вибором (за допомогою CheckBox), які саме фільтри використовувати при пошуку.

Основний функціонал - отримання списку файлів реалізується (як було описано в попередньому пункті) на основі методів **GetDirectories** та **GetFiles**.

Використовуючи ці методи отримуємо:

```
IEnumerable<string> findFiles
```

Якщо маска файлів не була задана (за замовчуванням використовується маска *.*).

Далі з колекції потрібно відібрати файли за вказаними фільтрами (якщо такі встановлені) та вивести імена файлів до списку на формі. За це відповідає цикл

```
foreach (string findFile in findFiles)
{}
```

У цьому циклі перебираються елементи колекції та перевіряються додаткові умови. Наприклад, перевірка на належність до зазначеного періоду створення файлу:

```
if (checkDate.Checked)//Фільтр за датою
  {
      DateTime creationD = File.GetCreationTime(findFile);
      if ((creationD >= min_d) && (creationD <= max_d))
      {
         fdata = true;
      }
  }
```

Тут перевіряються дані файлу і лише встановлюється прапорець – файл відповідає за діапазоном дат, якщо фільтр за датами не активований то цей прапор завжди true.

Перед додаванням файлу до списку та підрахунком статистики перевіряються усі прапори:

```
if (fdata & fsize)
   {
       listBox1.Items.Add(findFile);
```

```
            SumSize + = fileSize;
        }
```

Підсумкова кількість файлів визначається за розміром списку та виводиться на форму.

При виборі файлу викликається модуль дешифрації, функції якого роглянуто вище. Після дешифрування файл відкривається для перегляду.

4.3 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використати фіналіста конкурсу AES – шифр Rijndael. Він є нетрадиційним блоковим шифром, оскільки не використовує мережу Фейштеля для криптоперетворень. Алгоритм представляє кожний блок даних, що кодуються, у вигляді двовимірного масиву байт розміром 4x4, 4x6 або 4x8 залежно від встановленої довжини блоку. Далі на відповідних етапах перетворення відбуваються або над незалежними стовпцями, або над незалежними рядками, або взагалі над окремими байтами в таблиці.

Всі перетворення в шифрі мають строге математичне обґрунтування. Сама структура й послідовність операцій дозволяють виконувати даний алгоритм ефективно як на 16-бітних так і на 64-бітних процесорах. У структурі алгоритму закладена можливість паралельного виконання деяких операцій, що на багатопроцесорних робочих станціях може ще підняти швидкість шифрування в 4 рази.

Алгоритм складається з деякої кількості раундів (від 10 до 14 – це залежить від розміру блоку й довжини ключа), у яких послідовно виконуються наступні операції :

ByteSub – Таблична підстановка 8x8 біт (рисунок 4.7).

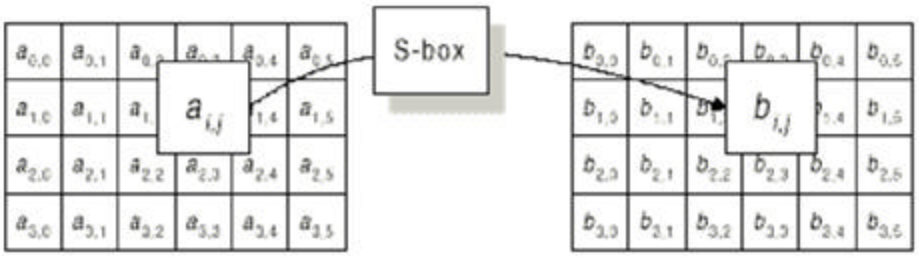


Рисунок 4.7 – Таблична підстановка 8x8 біт

ShiftRow – зрушення рядків у двовимірному масиві на різні зсуви (рисунок 4.8).

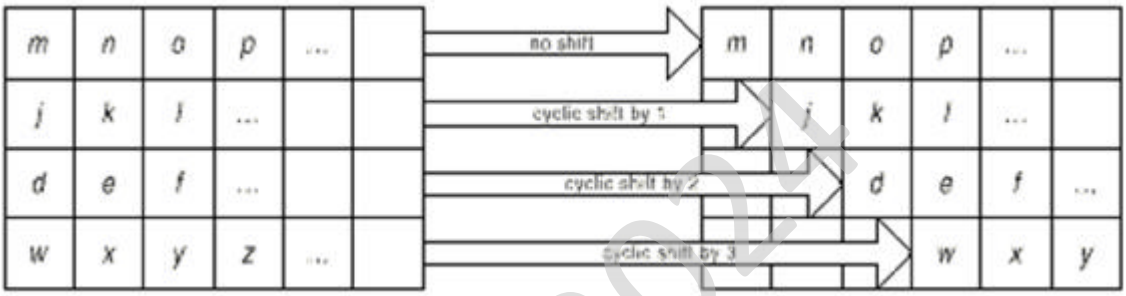


Рисунок 4.8 – Зрушення рядків у двовимірному масиві на різні зсуви

MixColumn – математичне перетворення, що перемішує дані усередині стовпця (рисунок 4.9).

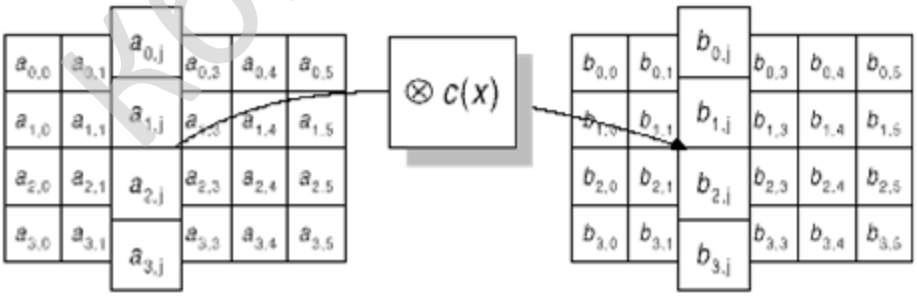


Рисунок 4.9 – Математичне перетворення, що перемішує дані усередині стовпця

AddRoundKey – додавання матеріалу ключа операцією XOR (рисунок 4.10).

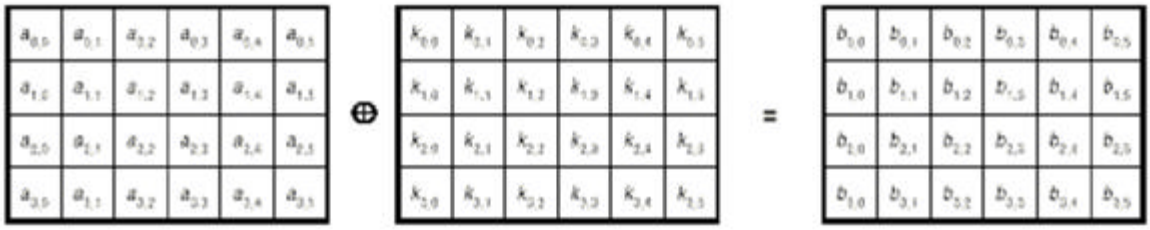


Рисунок 4.10 – Додавання матеріалу ключа операцією XOR

В останньому раунді операція перемішування стовпців відсутня, що робить всю послідовність операцій симетричною. Окрім розглянутого програмного способу реалізації шифрування існує і апаратний підхід.

Апаратне шифрування AES, як і програмне, існує вже давно.

Різниця між апаратним та програмним шифруванням

У випадку флеш-накопичувачів (а також HDD і SSD) для здійснення апаратного шифрування даних використовується спеціальний чіп, розташований на друкованій платі пристрою. У нього вбудований генератор випадкових чисел, що створює ключі шифрування. Дані шифруються автоматично і миттєво дешифруються при введенні пароля користувача. У цьому сценарії доступ до даних без пароля практично неможливо.

При використанні програмного шифрування "блокування" даних на накопичувачі забезпечується зовнішнім програмним забезпеченням, яке виступає у ролі бюджетної альтернативи апаратним методам шифрування. Недоліки такого ПЗ можуть полягати в банальній вимозі регулярних оновлень, щоб пропонувати стійкість до методик злому, що постійно покращуються. До того ж для дешифрування даних використовуються потужності комп'ютерного процесу (а не окремого апаратного чіпа), і по суті рівень захисту ПК визначає рівень захисту накопичувача.

Головна ж особливість накопичувачів з апаратним шифруванням полягає в окремому криптографічному процесорі, наявність якого говорить нам про те, що ключі шифрування ніколи не залишають USB-накопичувач, на відміну від програмних ключів, які можуть зберігатися в оперативній пам'яті комп'ютера або

на жорсткому диску. А оскільки програмне шифрування використовує пам'ять ПК для збереження кількості спроб входу в систему, воно не може зупинити атаки грубої сили на пароль або ключ. Лічильник спроб входу може постійно скидатися зловмисником доти, доки програма автоматичного злому паролів не знайде потрібну комбінацію.

До речі, в коментарях до статті Kingston DataTraveler: нове покоління захищених флешок користувачі також відзначили, що, наприклад, у програми TrueCrypt є портативний режим роботи. Однак це не є великою перевагою. Справа в тому, що в цьому випадку програма шифрування зберігається в пам'яті флеш-накопичувача, і це робить її більш уразливою для атак.

Як результат: програмний підхід не забезпечує настільки ж високий рівень безпеки, як AES-шифрування. Це, скоріше, базовий захист. З іншого боку – програмне шифрування важливих даних все ж таки краще, ніж взагалі відсутність шифрування. І цей факт дозволяє нам чітко розмежувати ці типи криптографії: апаратне шифрування флеш-накопичувачів – необхідна швидше для корпоративного сектора (наприклад, коли співробітники компанії користуються накопичувачами, виданими на роботі); а програмне – більше підходить для потреб користувача.

Наприклад, Kingston поділяє свої моделі накопичувачів (наприклад, IronKey S1000) на версії Basic (базові) і Enterprise (корпоративні). За функціональністю та властивостями захисту вони практично ідентичні один одному, але корпоративна версія пропонує можливість керування накопичувачем за допомогою SafeConsole/IronKey EMS. Завдяки цьому програмному забезпеченню, накопичувач працює або з хмарними, або з локальними серверами для дистанційного здійснення політик парольного захисту та доступу. Користувачам при цьому надаються можливості відновлення втрачених паролів, а адміністраторам – перемикання накопичувачів, що більше не використовуються, на нові завдання.

Kingston використовує 256-бітове апаратне шифрування AES-XTS (з використанням додаткового повнорозмірного ключа) для всіх безпечних

накопичувачів. Як ми вже зазначили вище, флешки містять у своїй компонентній базі окремий чіп для шифрування та дешифрування даних, який виступає в ролі постійно активного генератора випадкових чисел.

Коли користувач підключає пристрій до USB-порту вперше, майстер налаштування ініціалізації пропонує вам встановити майстер-пароль для доступу до пристрою. Після активації накопичувача алгоритми шифрування автоматично почнуть свою роботу відповідно до користувальницьких уподобань.

При цьому для користувача принцип роботи флеш-накопичувача залишиться незмінним - він, як і раніше, зможе завантажувати та розміщувати файли в пам'яті пристрою, як при роботі зі звичайною USB-флешкою. Єдина відмінність полягає в тому, що при підключенні флешки до нового комп'ютера вам потрібно буде вводити встановлений пароль для отримання доступу до інформації.

Для організацій, у яких конфіденційні дані є частиною бізнесу (фінансові, медичні чи державні установи), шифрування є найбільш надійним засобом захисту. У цьому плані флеш-накопичувачі з підтримкою 256-бітного апаратного шифрування AES – масштабоване рішення, яке може використовуватися будь-якими компаніями: від приватних осіб та малих підприємств до великих корпорацій, а також військових та урядових організацій. Якщо розглядати це питання трохи конкретніше, використання зашифрованих USB-накопичувачів необхідно:

• Для забезпечення безпеки конфіденційних даних компанії

• Для захисту інформації про клієнтів

• Для захисту компаній від втрати прибутку та лояльності клієнтів

Варто зазначити, що деякі виробники захищених флеш-накопичувачів (у тому числі і компанія Kingston) надають для корпорацій індивідуальні рішення, розроблені під потреби та завдання замовників. Але і масові лінійки (в числі яких флешки DataTraveler) чудово справляються зі своїми завданнями та здатні забезпечити корпоративний клас безпеки.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

При реалізації програмного комплексу враховувались основні стандарти з точки зору захищеного доступу та загальні інтерфейсні вимоги. Реалізація програми передбачає мінімальні вимоги до навчання персоналу. Інтуїтивність програмних діалогів дозволяє взаємодіяти з програмою без додаткових інструкцій для впевненого користувача ПК, для персоналу (що підлягає атестації) система передбачає інструкції та довідникову систему.

Діючим стандартом в Україні, що визначає вимоги до якості програмного забезпечення, є ДСТУ ISO/IEC 25010:2017 "Системи та програмне забезпечення. Вимоги до якості та оцінка". Цей стандарт є національним адаптованим варіантом міжнародного стандарту ISO/IEC 25010:2011.

ДСТУ ISO/IEC 25010:2017 визначає модель якості програмного забезпечення, яка містить вимоги до його властивостей та характеристик. Ці вимоги поділяються на дві групи: функціональні та нефункціональні вимоги. Функціональні вимоги відображають властивості, пов'язані зі здатністю програмного забезпечення виконувати функції, які відповідають вимогам користувачів. Нефункціональні вимоги відображають властивості, які не пов'язані зі здатністю програмного забезпечення виконувати функції, але впливають на його якість.

В якості практичної частини роботи реалізовано програмний прототип, що демонструє розмежування доступу користувачів до даних на зовнішньому носії. Програма частково реалізує функції, розглянуті в ході аналізу тематики, але ілюструє основну ідею реалізації захищеного доступу до даних на зовнішніх носіях. Враховуючи вище сказане, програмна розробка може впроваджуватись в запропонованому вигляді, але для реального використання все ж потребує певних функціональних доопрацювань.

Взаємодія із програмою відбувається через графічний інтерфейс, а саме

діалогові вікна WindowsForm. Приклади можливих програмних вікон представлені рисунках (нижче) даного розділу. Розроблений інтерфейс опирається на стандарти ОС Windows, що спрощує взаємодію з системою для оператора. Як видно на представлених нижче екранних копіях основних вікон інтерфейсу, вікна не перенасичені елементами керування, інтерфейс простий у використанні й інтуїтивно зрозумілий.

Програма має простий інтерфейс, що досягається використанням мінімальної кількості компонент.

Загальні відомості

Розроблена програма представляє модуль захищеного файлового менеджеру для пошуку файлів на диску (або в обраному каталозі користувача) з здатністю додаткових параметрів фільтрації. Програма розроблена серед VisualStudio 2022 мовою C#.

Функціональне призначення

Модуль призначено для захищеної взаємодії з зовнішнім носієм та здійснення основних файлових операцій. Зокрема реалізовано: пошук файлів на зовнішньому носії, фільтрація за різними критеріями.

Безпосередні показники програми:

- інсталяція не потрібна, достатньо скопіювати програму на жорсткий диск;

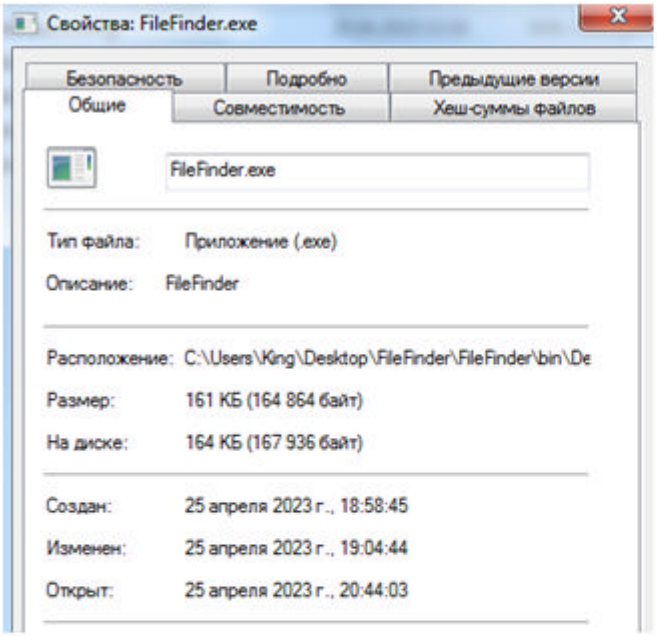
- розмір файлу запуску (FileFinder.exe) – 164 Kbt

- Додаток не використовує жодних додаткових файлів або бібліотек (зрозуміло .Net не враховується, додаток буде працювати тільки за наявності попередньо встановленого фраймворку);

- Об'єм в оперативній пам'яті під час роботи 23 264 Кбт

Об'єм суттєво залежить від потенційної кількості файлів у цільовому каталозі пошуку, тому що формується список у пам'яті.

Запускний файл



Процес

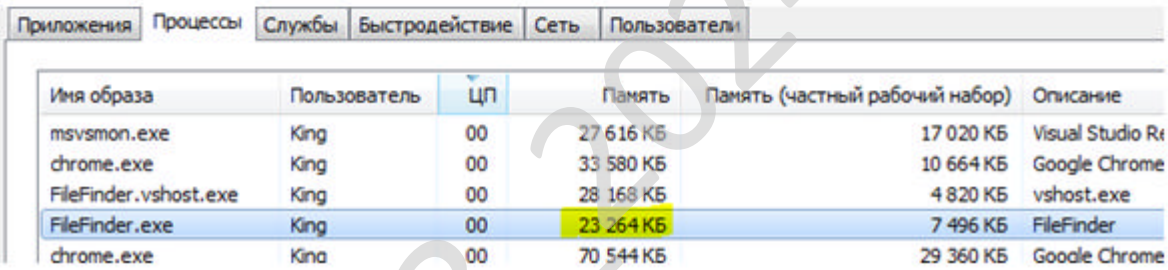


Рисунок 5.1 – Розміри у пам'яті розробленого програмного додатку

Виклик та завантаження

Програма запускається на виконання або подвійним клацанням лівої кнопки миші по файлу FileFinder.exe у провіднику Windows, або натисканням правої кнопки миші за вказаним файлом та вибором пункту меню “Відкрити”.

Тестування розробленої програми. Інструкція користувача

Тестування програми показало адекватну реакцію на різноманітні ситуації та дії користувача. Навігація по інтерфейсним елементам працює адекватно, про це свідчать скріни форм, що представлені нижче.

Після запуску програми користувач бачить головне вікно програми (рис. 5.2)

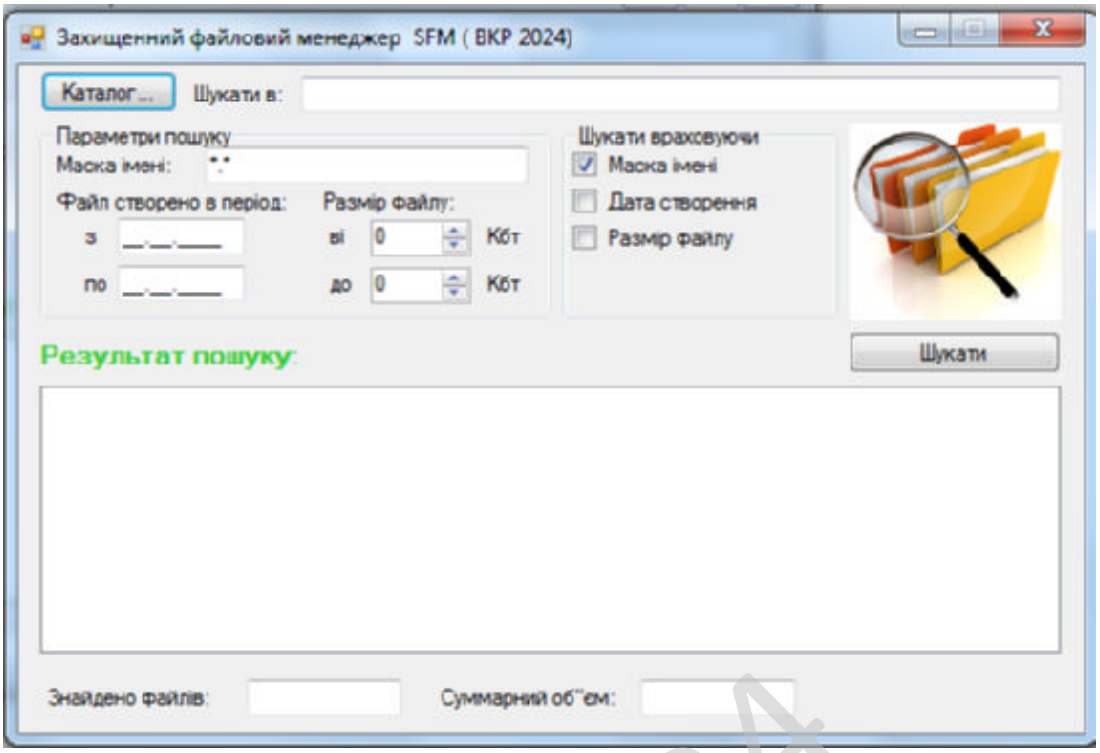


Рисунок 5.2 – Головне вікно програми

При натисканні на кнопку "Каталог" створюється системний діалог "Вибір каталогу"

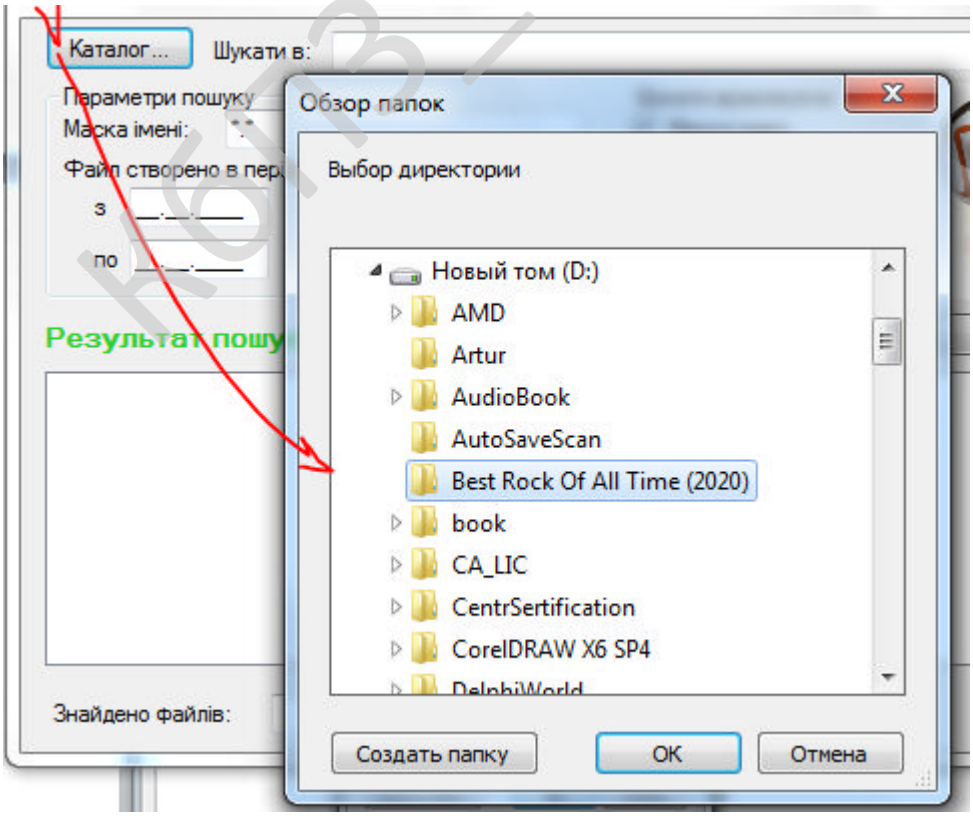


Рисунок 5.3 – Вибір каталогу

Налаштування пошуку (фільтри та їх параметри) доступні користувачеві у спеціально відведених блоках форми.

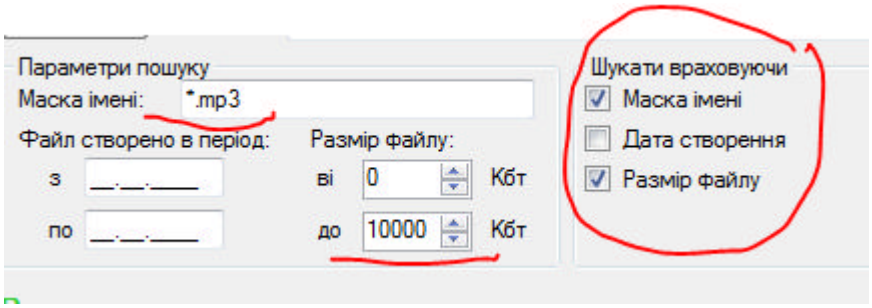


Рисунок 5.4 – Налаштування параметрів пошуку

На наступному рисунку показаний результат пошуку із зазначеними параметрами: пісні (файли *.mp3) з каталогу «D:\Best Rock Of All Time (2020)», що мають розмір до 10 Mbt.

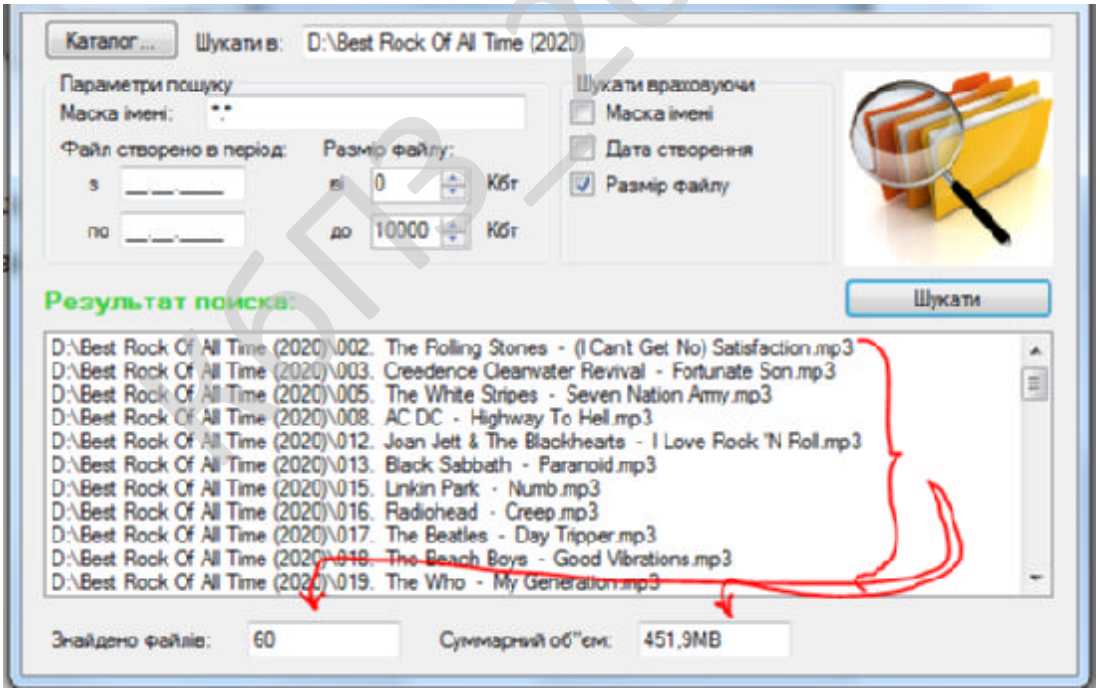


Рисунок 5.5 – Результат1: отримання списку файлів після фільтрації

На рис. 5.6 приведено приклад отримання вмісту файлу через програму(зліва) та через провідник Windows(зправа).

Ілларіон Павлюк "Я бачу, вас цікавить пітьма" 2022 (фрагмент) Сніг обрушився на Буськів Сад, наче хтось тряс над селищем величезне сито з борошном. Сутінки вмить згусли, і червоний сигнальний маячок загубився в імлі. Маленька дівчинка, стоячи в заводській брамі, вертіла головою, та вогника ніде не було. Навколишній світ ураз розсипався мов картковий будиночок. У незнайомому місці так було завжди, варто лишень відпустити мамину руку. Надя вже не знала, куди йти. Не знала, як повернутися додому, не розуміла, де вона і як вирватися із плаского простору, що огорнув її, наче рулон паперу. Какофонія звуків і власних думок наразі поглинула дівчинку. Плесь! Плесь-плесь!	=1×i ćçTG éq[ñ 1H fG[Ù¿×q =HDB 1Hj Ñzo[Ù zTù[1HD [j-ë'j^ qj-éj©1¡ [ézó ® çóTó 1ú^ã –Gz×l¿é q^ézÙ ©íG[-éB × ¡Tqó1éO [ei1H-Ù1éO ^Hñ ó¿ ®HíGçéç[ñ ç ×^–×ã H-j1U¿H D×q ël¿HB [zóñ é q ®HqéD[Ù¿×O çTH℅B qj'Tz×-H ió-ćçó¥B zH çóí1é¿H l×Dj 1j çG-óã ÆHq¿ó-ś 1×O [q×z GTH® Tó®[é¼Hq[ñ °óq ¿HTz¿óqéO çGDél ó ó¿ã ü lj©1HOó°ó°G ℅[Ã× zH¿ çG-ò ©HqŠDëB qHTzò -é j1Ù q×D½G[zézé °H°él G TG¿Gã ÆHD℅ qŠj !j ©1H-HB ¿GDé Ozéª Æj ©1H-HB ñ¿ ¼óqjT1Gzé[ñ DóDó^GB !j Tó®G^×-HB Dj çó1H × ñ¿ qéIçHzé[ñ ×© ¼-H[¿óió ¼Tó[zóTGB \ć óíöT1Gq ww B lIIj TG-ó1 ¼II½jTGã {II¿ó⊥ó1>ñ ®qC¿×q × ç-II[1éÑ DG°ó¿ 1HTH® ¼ói-ć1G-H D×ç ć1¿Gã -j[Ù^ -j[Ù ¼-j[Ù^ U¡×TH q×DíG¿1G-H[ñ çć-j°B — zH¿ [ə Ù1ò ÆHDÑ çDHTe H Dò ò1Ù¿H°eã ɟ Ãj ©HqŠDë Dó¼ò°HiH-óÏ ®qG¿ -ñ½H1Â×q [zHqHq ió-ćq·ë°B ww q-H[1× TG¿ë — ç-é©Ù¿é°ëB ćzSjB qlj Tj zH — DH-j¿é°B × qó1H ½óqjTzH-H[ñ q TjH-Ù1×[zÙã sjTj® ¿òTúz¿G Ñqé-Ù¿G [q×z ©1óçG që zóqÑ1j ww 1II ¼óqjTÑ1¥B O zòD× qò1II ¼-j[1j ®1óqCã -j[Ù^ -j[Ù ¼-j[Ù^ ñG-ć O ×1 × ÑćzTó\×ã ÆEH¼Té¿-HDB ã¡\ò Dëçézé[ñ 1H \c[Ù ¿'1×©Ù [¼ jzj1× ¼H ÙÂ×B zò ɟı ɟ [zHð ç×Dò¿'Ij° ŧqHzé =ò1°G q×D =ò1Gã ɟ çG ë O zH¿× Tj ×B ñ¿× lj ¼-GzH-é[ñ O lj ©-éçH-é[ñ 1×¿ć-ë

Рисунок 5.6 – Відкритий та зашифрований текст

Звичайно, при пошуку на локальних дисках час роботи програми істотно збільшиться. Важко дати конкретну оцінку, оскільки залежить від багатьох параметрів (швидкість роботи HDD, складність структури каталогів, параметри фільтрів, глибина вкладеності каталогів та інших.). У тестовому варіанті запуску пошук усіх файлів по масці *.png на диску системному диску C: зайняв близько 14 хв.

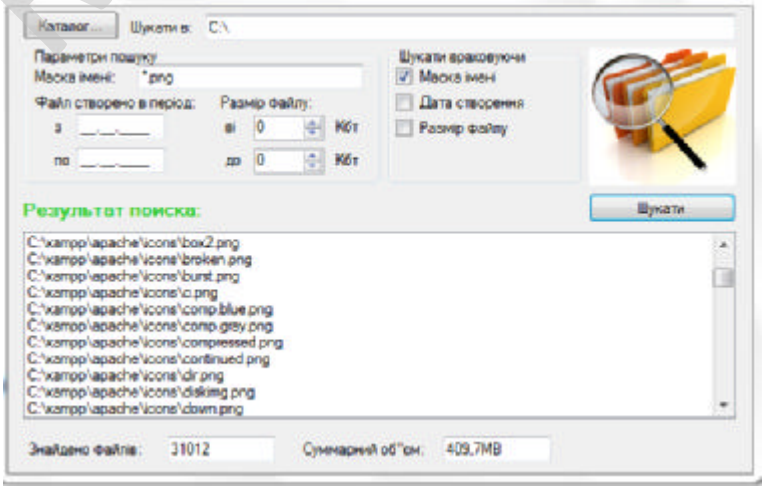


Рисунок 5.7 – Результат 2 (пошук на локальному диску)

При цьому обсяг диска C: на ПК, де проводилася тестування, дорівнює 100 Гб. На рис. 5.8 представлена статистика за каталогами та файлами тільки відкритих доступних користувачеві каталогів

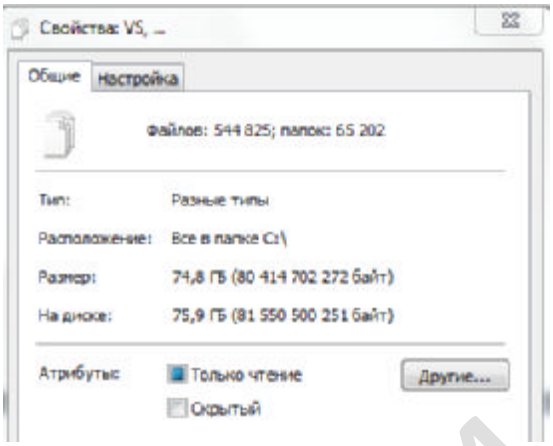


Рисунок 5.8 – Статистика (папки, файли) на диску C:\

6 ОСНОВНІ ВИСНОВКИ

В ході даної роботи проведено дослідження технологій та підходів захисту даних на зовнішніх носіях.

Дипломна робота включає 5 розділів основної частини.

У першому розділі проводиться короткий аналіз основних понять, типів та підходів до реалізації аутентифікації.

Другий розділ розглядає сферу застосування. Тут розглянуто різні типи носіїв, приведено їх порівняльну характеристику В останньому пункті розділу подається словесна постановка задачі з деталізацією функцій, які потрібно реалізувати.

В третьому розділі здійснюється проектування програмного комплексу. В цьому розділі розроблено різноманітні діаграми та моделі. Після цього було розглянуто кілька сценаріїв, у яких потрібна система захисту доступу до файлів на зовнішніх носіях. Були виявлені найбільш важливі в кожному випадку критерії, а також проведено порівняльний аналіз методів. В третьому розділі також розглянуто структуру програмного комплексу, будовано функціональну діаграму та діаграму процесів.

Четвертий розділ присвячений практичній реалізації програмної системи забезпечення захищеного доступу до файлів на зовнішніх носіях. На початку розділу будуються блок-схеми алгоритмів, далі описуються прийняті алгоритмічні рішення та використані технології.

В п’ятому розділі наведено докладну інструкцію користувача з ілюстрацією екранними копіями форм користувача. Програмна реалізація передбачає наявність основних функцій для взаємодії з файлами, що притаманні іншим реалізаціям файлових менеджерів: пошук файлів за різними критеріями, фільтрація, отримання даних та специфікацій файлу.

У додатках роботи наведено фрагменти програмного коду.

Файловий менеджер є одним з невід'ємних елементів інструментального набору користувача для взаємодії з файлами в ОС. Програми подібного типу з'явились одними з перших, бо файлові операції відносяться до базових.

Розробка повнофункціонального захищеного файлового менеджеру задача досить об'ємна, тому в практичній частині вирішено зосередитись лише на основних функціях: шифрування файлу, базові файлові операції.

В практичній частині роботи вирішена задача розробки програми для пошуку файлів з набором додаткових параметрів. Розроблений додаток може в подальшому розширюватись за рахунок додавання нового функціоналу або інтегруватись як модуль в іншу програму.

У ході тестування були отримані результати роботи реалізованого ПЗ, якими видно, що програма працює адекватно, процес пошуку відповідає обраним фільтрам і постановці завдання. Помічені під час тестування недоліки та помилки було усунено.

Практична реалізація може мати ряд подальших удосконалень: як з точки зору інтерфейсу, так і з точки зору алгоритмів та підходів. В той же час програма повністю працездатна, що підтверджується результатами її тестування. Завдання, поставлені у роботі, виконані повному обсязі.

Структура та зміст роботи повністю відповідають вимогам та тематиці кваліфікаційної роботи, завдання, поставлені в роботі, виконані у повному обсязі.

СПИСОК ЛІТЕРАТУРИ

1. ДСТУ ISO/IEC/IEEE 16326:2015 Розроблення систем та програмного забезпечення. Процеси життєвого циклу. Керування проектами (ISO/IEC/IEEE 16326:2009, IDT)
2. Котляров В. П., Т. В. Колікова, Основи тестування програмного забезпечення, Видавництва: Інтернет-університет інформаційних технологій, Біном. Лабораторія знань, 2012, 288 стор.
3. Красовська, Л. В. Методи захисту даних в базах даних [Текст] / Л. В. Красовська // Вісник Київського національного університету імені Тараса Шевченка. Економіка. - 2022. - № 1(193). - С. 45-49.
4. Культін А.В. Реляційні бази даних: практичні прийоми ефективних рішень. - К.: ВД «Грифон», 2020 - 400с.;
5. Основи_дизайну_інтерфейсу [Електронний ресурс].– URL:http://froland.org/samodel/vbguide/ch6_8_1.html (дата звернення: 26.02.24)
6. Ріхтер, Дж. CLR via C #. Програмування на платформі Microsoft .NET Framework 4.5 мовою C#. Майстер клас. Пров. з англ. / Дж. Ріхтер. - К.: ПРОФІ, 2019. - 656 с.: Іл. - ISBN 978-5-91180-303-2.
7. Сулейманов Р.Р. Методика вирішення навчальних завдань засобами програмування. Мова C# //Методичний посібник - К: 2021, с. 112
8. Троєлсон Е. C # 6.0 and the .NET 4.6 Framework,- К.: Вільямс, 2016.
9. Ульман Дж. Основи систем баз даних. К.: Махаон, 2019.-334 с.
10. Яровий М.К. Якісний програмний продукт // Проектування та розробка ПЗ. 2019 . №2
11. Баранов А.П., Борисенко Н.П., "Математичні основи криптографії", перекл. – К.: 2018.
12. Задірака В.К. МЕТОДИ ЗАХИСТУ БАНКІВСЬКОЇ ІНФОРМАЦІЇ / (В.К. Задірака, О.С.Олексюк, М.О.Недашковський), Навчальний посібник. — К.:

Вища школа, 2021. — 405 с.

13. Курбатов В.М. Політики безпеки інформації при взаємодії компаній череж мережу//Захист інформації в мережах №11, 2016

14. Dumitrescu, S., W. Xiaolin and Z. Wang, 2003. Detection of LSB steganography via sample pair analysis. In: LNCS, Vol. 2578, Springer-Verlag, New York, pp: 355–372.

15. Fionov A. N., Eltysheva (Merzlyakova) C. U. Stegosystem construction on the basis of raster images by permutations. // 2010 IEEE Region 8 international conference on computational technologies in electrical and electronics engineering. Sibircon-2010. Volume 1. Irkutsk , July 11-15 2010. IEEE. P. 137-140.

16. Security and Privacy in Electronic Health Records: A Systematic Literature Review" (IEEE Access, 2019)

17. Verizon 2022 Data Breach Investigations Report [Електронний ресурс]// URL: https://www.verizon.com/business/resources/reports/dbir/2022/master-guide/ (дата звернення: 25.04.2024)

18. Widup, Suzanne & Pinto, Alex & Hylender, David & Bassett, Gabriel & Langlois, Philippe. (2021). 2021 Verizon Data Breach Investigations Report.

19. Аутентифікація. Теорія і практика забезпечення захищеного доступу до інформаційних ресурсів: навч. посібник / А. А. Афанасій, Л. Т. Веденів, А. А. Воронцов . – К.: Гаряча лінія – Телеком, 2017. – 552 с.

20. Velsquez, I., Caro, A., & Rodrнguez, A. (2017). Identifying Comparison and Selection Criteria for Authentication Schemes and Methods.

21. Velsquez, Ignacio & Caro, Angelica & Rodrнguez, Alfonso. (2019). Multifactor Authentication Methods: A Framework for Their Comparison and Selection.

22. Foong Annie, H. F. Storage as fast as rest of the system / H. F. Foong Annie // Memory Workshop (IMW). — 2016 IEEE 8th International. — IEEE. 2016. — С. 1—4.

23. Dirik, C. The Performance of PC Solid-state Disks (SSDs) As a Function

of Bandwidth, Concurrency, Device Architecture, and System Organization / C. Dirik, B. Jacob // SIGARCH Comput. Archit. News. — New York, NY, USA, 2009. — T. 37, № 3. — C. 279—289.

24. Park, S. FIOS: A Fair, Efficient Flash I/O Scheduler / S. Park, K. Shen // Proceedings of the 10th USENIX Conference on File and Storage Technologies. — San Jose, CA : USENIX Association, 2012. — C. 13 — URL: http://dl.acm.org/citation.cfm?id=2208461.2208474 (дата звернення 15.04.2024).

25. Extending SSD Lifetimes with Disk-based Write Caches / G. Soundararajan [и др.] // Proceedings of the 8th USENIX Conference on File and Storage Technologies. — San Jose, California : USENIX Association, 2010. — C. 8—8. — (FAST'10). — URL: http://dl.acm.org/citation.cfm?id=1855511. 1855519 (дата звернення 15.12.2017).

26. A 72% error reduction scheme based on temperature acceleration for long-term data storage applications: Cold flash and millennium memories / S. Yamazaki [et al.] // Solid-State Electronics. — 2016. — July. — Vol. 121. — P. 25—33. — URL: http://linkinghub.elsevier.com/retrieve/pii/ S0038110116000459

27. Chen, F. CAFTL: A Content-aware Flash Translation Layer Enhancing the Lifespan of Flash Memory Based Solid State Drives / F. Chen, T. Luo, X. Zhang // Proceedings of the 9th USENIX Conference on File and Stroage Technologies. — San Jose, California : USENIX Association, 2011. — C. 6—6. — (FAST'11). — URL: http://dl.acm.org/citation.cfm?id=1960475. 1960481 (дата звернення 15.03.2024).

28. A Fresh Perspective on Total Cost of Ownership Models for Flash Storage in Datacenters / Z. Yang [и др.] // Cloud Computing Technology and Science (CloudCom), 2016 IEEE International Conference on. — IEEE, 2016. — C. 245—252.

29. Mittal, S. A Survey of Software Techniques for Using Non-Volatile Memories for Storage and Main Memory Systems / S. Mittal, J. S. Vetter // IEEE Transactions on Parallel and Distributed Systems. — 2016. — Май. — Т. 27, № 5. — С. 1537—1550. — URL: http : / / ieeexplore . ieee . org / document / 7120149/ (дата звернення 15.03.2024).

30. Bonneau, J., Herley, C., Oorschot, P. C. van, & Stajano, F. (2012). The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. 2012 IEEE Symposium on Security and Privacy. Cutting, Danie&Landfeldt, Bjurn & Quigley, Aaron. (2006). Implicit Group Messaging over Peer-to-Peer Networks. 125–132. 10.1109/P2P.2006.20.

31. OAuth 2.0 [Електронний ресурс]//URL: https://oauth.net/2/ (дата звернення: 20.01.2024).

32. Auzy [Електронний ресурс]// URL:https://github.com/alexey-detr/auzy (дата звернення: 20.01.2024).

33. Neerja Bhatnagar. Security in Relational databases / N. Bhatnagar // - In: Handbook of Information and Communication Security. ed. by P. Stavroulakis, M. Stamp.- Springer. – 2010. - pp. 257 - 272.

34. SoK: Cryptographically Protected Database Search / Fuller B., Varia M., Yerukhimovich A., Shen E., Hamlin A., Gadepally V., Shay R., Mitchell J. D., Cunningham R. K. / 2017

35. Захист програмного забезпечення. Частина 2: навчальний посібник / В. А. Каплун, О. В. Дмитришин, Ю. В. Баришев – Вінниця : ВНТУ, 2014. – 105 с.

36. НД ТЗІ 2.5-004-99. Критерії оцінки захищеності інформації в комп'ютерних системах від несанкціонованого доступу.

37. Сторожук А., Software Engineer в Cossack Labs/ Шифрування в базах даних SQL з можливістю пошуку

38. 2015 Cyber Security Survey: Major Australian Business // Australian Government, Australian Cyber Security Centre.

39. Дж. Брассар "Сучасна криптологія" pdf-кіопія.

40. Жельников В. Криптографія від папіруса до комп'ютера. – М.: ABF, 2011. – 335 с.

41. Криптографічний захист інформації і системи розподілу ключів. Електронний ресурс [режим доступу]: http://pidruchniki.com, час доступу 28.01.2018

42. КРИПТОГРАФІЧНИЙ ЗАХИСТ ФІНАНСОВОЇ ІНФОРМАЦІЇ / А.І. Пасько, Електронний ресурс [режим доступу]: https://habrahabr.ru/post/114875/, час доступу 12.04.2024

43. Франчук В.М. Захист інформаційних ресурсів: криптографічні та стеганографічні методи захисту даних. Посібник для викладачів, вчителів та студентів інформатичних спеціальностей. – К.: НПУ імені М.П. Драгоманова, 2012. – 120 с.

44. Blockchain Technology for Healthcare: Facilitating the Transition to Patient-Driven Interoperability" (Computational and Structural Biotechnology Journal, 2018) -

45. Clemens Scott Kruse Challenges and Opportunities of Big Data in Health Care: A Systematic Review" (JMIR Medical Informatics, 2016)

46. Chandramouli, R. and N. Memon, 2001. Analysis of LSB based image steganography techniques. Proc.ofICIP, Thessaloniki, Greece. Электронный документ [режим доступа: http://vanilla47.com/PDFs/Cryptography/Steganography], время доступа: 11.04.2024

47. Diffie W., Hellman M. E. New Directions in Cryptography // IEEE Trans. Inf. Theory / F. Kschischang — IEEE, 1976.

48. Dumitrescu, S., W. Xiaolin and Z. Wang, 2003. Detection of LSB steganography via sample pair analysis. In: LNCS, Vol. 2578, Springer-Verlag, New York, pp: 355–372.

49. Ethical Considerations in Digital Health: Focus on Healthcare Data Sharing and Interoperability" (Healthcare Informatics Research, 2019)

50. Privacy-Preserving Data Sharing in Healthcare: Current Solutions and Challenges" (BMC Medical Informatics and Decision Making, 2019)

51. Raymond B. Wolfgang and Edward J. Delp. A watermark for digital images. In International Conference on Images Processing, pages 219-222, Lausanne, Switzer-land, September 1996. IEEE. http://www.autex.spb.ru/wavelet/books.htm

52. Security and Privacy in Electronic Health Records: A Systematic

Literature Review" (IEEE Access, 2019)

53. Simmons G.J. The prisoner`s problem and the subliminal channel, Proc. Workshop on Communications Security (Crypto`83), 1984, 51-67.

54. Walter Bender, Daniel Gruhl, Norishige Morimoto, and Anthony Lu. Techniques for data hiding. IBM Systems Journal, 35(3 & 4):313{336, 1996}.

55. Zollner J., Federrath H.,Klimant H., Pfitzmann A., Piotraschke R., Westfeld A., Wicke G., Wolf G. Modeling the security of steganographic system, Proc. 2nd International Workshop on Information Hiding, 1998, LNCS, v.1525, 344-354.

56. Seafile, a Self-Hosted Dropbox Alternative https://gofoss.net/cloud-storage/

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку програмного забезпечення системи кібербезпеки комплексу захищеної взаємодії з інформаційними носіями.

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 (№135-02, від 01.04.24).

3 Мета та призначення розробки

Метою бакалаврської дипломної роботи є дослідження підходів та методів реалізації захищеного доступу до даних (файлів) на зовнішніх носіях.

4 Джерела розробки

Джерелом цієї бакалаврської дипломної роботи є відносна 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 з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Об’єкт ВКР – технології та підходи захисту даних та обмеження несанкціонованого доступу до даних на зовнішніх носіях.

Предмет дослідження ВКР – застосування технологій для реалізації програмного комплексу обмеження доступу до файлів на флеш-накопичувачі чи зовнішньому жорсткому диску.

Метою випускної бакалаврської кваліфікаційної роботи є реалізація програмного додатку, що використовує ефективні технології захисту даних на зовнішніх накопичувачах.

В випускній кваліфікаційній бакалаврській роботі визначено наступні **задачі**:

1) провести аналіз технологій захисту даних на зовнішніх накопичувачах;

2) вибрати найефективніші та прийнятні, відносно постановки задачі, технології для реалізації;

3) розробити програму, що використовує технології захисту даних на зовнішніх накопичувачах;

4) підготувати технічну документацію.

Важливим аспектом є забезпечення зручності використання розподілених систем. Це має на увазі навіть наявність зручних у застосуванні систем аутентифікації, інтуїтивно зрозумілих для пересічного користувача. Все це робить проблему доступу у розподілених системах актуальною темою дослідження.

Практична значимість дослідження полягає в можливості впровадження розробленого програмного комплексу в експлуатацію для організацій, що потребують обмеження доступу до файлів на зовнішніх носіях (державні

установи, комерційні організації, дослідницькі центри, організації, що оперують персональними даними).

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення у вигляді WinForms додатку на мові програмування C#. Має забезпечувати можливість захищеного доступу до файлів на зовнішніх носіях інформації, зокрема флеш-накопичувачах.

5.4 Вимоги до архітектури

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

Для функціонування системи необхідний наступний набір: системний блок, монітор, клавіатура, миша.

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: 18-20 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 архітектури IBM PC, працювати в ОС Windows 7/8/10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Мобільність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 7/8/10/11, встановлений фраймворк .NetFrameWork 4.5.

5.8.1 Обладнання

Комп'ютер Intel® CoreI3/16 Гб/1.2 Gb/AGP 256 Mb або сумісні з ним.

5.8.2 Мова програмування

Програмний модуль реалізовано як WinForms на мові програмування C# VisualStudio, тип інтерфейсу – WinForms. FrameWork 4.5 і вище.

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 – 76 аркушів.

8 Етапи розробки

8.1 Збір і обробка інформації по темі бакалаврської дипломної роботи. Постановка задачі на виконання бакалаврської дипломної роботи (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень бакалаврської дипломної роботи.

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 бакалаврської дипломної роботи на попередній захист 19.05.2024 р.

9.2 Подання бакалаврської дипломної роботи на захист 03.06.2024 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник бакалаврської дипломної роботи

_____ Улічев О.С.

Програмне забезпечення системи кібербезпеки комплексу захищеної взаємодії з інформаційними носіями

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 12

Літера: РП

Кропивницький – 2024 року

Основна програма

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Text;
using System.Windows.Forms;


namespace FileFinder
{

    public partial class Form1 : Form
    {
        String chDir = "";
        public Form1()
        {
            InitializeComponent();
        }
            //Перетворення розмірності файлу в рядкове значення
        static String BytesToString(long byteCount)
        {
            string[] suf = { "Byt", "KB", "MB", "GB", "TB", "PB", "EB" };
//масив назв розмірностей
            if (byteCount == 0)
 return "0" + suf[0];
            long bytes = Math.Abs(byteCount);
            int place = Convert.ToInt32(Math.Floor(Math.Log(bytes, 1024)));
            double num = Math.Round(bytes / Math.Pow(1024, place), 1);
            return (Math.Sign(byteCount) * num).ToString() +
suf[place];//функція перетворення
        }
            //Обробник події "Вибір каталогу"
        private void button1_Click(object sender, EventArgs e)
        {

            FolderBrowserDialog DirDialog = new
FolderBrowserDialog();//створення діалогу
            DirDialog.Description = "Вибір директорії";
            DirDialog.SelectedPath = @"C:\";//Начальний каталог (по замовченню)

            if (DirDialog.ShowDialog() == DialogResult.OK)
            {
 PathTextBox.Text = DirDialog.SelectedPath;//виведення каталогу на форму
 chDir = DirDialog.SelectedPath;//запамятовуємо каталог в змінній
            }


        }
//Обробник кнопки "Пошук"
        private void button2_Click(object sender, EventArgs e)
        {
            long SumSize = 0;//Сумарний розмір
            String mascf = "*.*";//Маска файлу можна використовувати символи "?"
и "*"
            decimal max_size=0, min_size=0;//границі розміру файлу під час
пошуку
            DateTime max_d = DateTime.Now, min_d = DateTime.Now;//границі дат
            bool fdata = true, fsize = true;//прапори критеріїв пошуку
            listBox1.Items.Clear();//список для виведення результатів
            TotalSizeBox.Text = "";
            CountFile.Text = "";
            if (checkSize.Checked)//обрано пошук за розміром
            {
 max_size=MaxS.Value;
```

```
 min_size=MinS.Value;
 fsize = false;
            }
            if (checkDate.Checked)// обрано пошук по даті
            {
 max_d = DateTime.Parse(maskedTextBox2.Text);
 min_d = DateTime.Parse(maskedTextBox1.Text);
 fdata = false;
            }
            if (checkMasc.Checked)// обрано пошук за іменем (маска)
            { mascf = textBox1.Text; }
                  //Отримати всі файли за маскою
            IEnumerable<string> findFilesList =
System.IO.Directory.EnumerateFiles(chDir, mascf,
System.IO.SearchOption.AllDirectories);
            foreach (string findFile in findFilesList)
            {
 var fileSize = new System.IO.FileInfo(findFile).Length;

 if (checkSize.Checked)//Фільтр по розміру
 {
    if ((fileSize / 1024 > min_size) && (fileSize / 1024 < max_size))
    {
    fsize = true;
    }
 }
 if (checkDate.Checked)//Фільтр по даті
 {
    DateTime creationD = File.GetCreationTime(findFile);
    if ((creationD >= min_d) && (creationD <= max_d))
    {
    fdata = true;
    }
 }
 if (fdata && fsize)
 {
    listBox1.Items.Add(findFile);
    SumSize += fileSize;
 }
 if (checkSize.Checked)//Скидання прапорів відповідності
 {
   fsize = false;
 }
 if (checkDate.Checked)// Скидання прапорів відповідності
 {
   fdata = false;
 }

            }
            CountFile.Text =  listBox1.Items.Count.ToString();
            TotalSizeBox.Text = BytesToString(SumSize);
        }
    }
}

public partial class ProgressWindow : Form
    {
        bool wasClickCancel = false;
        public pictureStegonography steg;
        public ProgressWindow()
        {
            InitializeComponent();
        }

       public void BackgroundWorker_ProgressChanged(object sender,
ProgressChangedEventArgs e)
        {
            this.Text = "Виконано " + e.ProgressPercentage.ToString()+"%";
            label1.Text = "Виконано " + e.ProgressPercentage.ToString() + "%";
```

```
            progressBar1.Value = e.ProgressPercentage;
        }

       public void BackgroundWorker_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
        {
            if (e.Cancelled)
            {
                MessageBox.Show("Операція перервана!,
                 MessageBoxButtons.OK, MessageBoxIcon.Stop);
            }
            else
            {
                MessageBox.Show("Текст вдало приховано! ", "Операцію завершено",
                         MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
            this.Close();
        }


           public void ExtractBackgroundWorker_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
        {
            if (e.Cancelled)
            {
                if (wasClickCancel)
                {
                    MessageBox.Show("Операція вилучення закінчилась невдало",
"Помилка", MessageBoxButtons.OK, MessageBoxIcon.Stop);
                }
                else
                MessageBox.Show("Операція вилучення перервана! ", "Помилка",
                 MessageBoxButtons.OK, MessageBoxIcon.Stop);
            }
            else
            {
                MessageBox.Show("Текст вдало вилучено! ", "Операцію завершено",
                        MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
            this.Close();
        }

        private void button6_Click(object sender, EventArgs e)
        {
           wasClickCancel = true;
            steg.BackgroundWorker.CancelAsync();
        }


    }
}

private void Button_Click(object sender, RoutedEventArgs e)
        {
            if(dgShow.SelectedItem==null) return;
            var settings = dbWorker.GetSettigsFromDb(dgShow.SelectedItem as
TargetModel);
            //Математика
            settings.CriterionVectors =
settings.CalculateVectors(settings.CriterionMatrixDouble);
            foreach (var alternativesMatrix in settings.AlternativeMatrixDouble)
            {

settings.AlternativesVectors.Add(settings.CalculateVectors(alternativesMatrix));
            }
            settings.AlphaList =
settings.CalculateAlphaList(settings.CriterionVectors,
settings.AlternativesVectors, settings.Alternatives.Count);
```

```
            //Створюємо звіт
            var excelWorker = new ExcelWorker();
            excelWorker.CreateReport(settings);
        }

        private void Button_Click_1(object sender, RoutedEventArgs e)
        {
            if (dgShow.SelectedItem == null) return;
            var settings = dbWorker.GetSettigsFromDb(dgShow.SelectedItem as
TargetModel);
            dbWorker.DeleteTarget(dgShow.SelectedItem as TargetModel);
            dgShow.ItemsSource = null;
            dgShow.ItemsSource = dbWorker.GetAllTarget();
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography.X509Certificates;
using System.Text;
using System.Threading.Tasks;

namespace MAI.Classes
{
    public class DbWorker
    {
        public void CreateTargetInDb(Settings settings)
        {
            using (var db = new MAIContext())
            {
                List<Alternatives>addedAlt = new List<Alternatives>();
                var target = new Target(){TargetName = settings.Target};
                db.Target.Add(target);
                db.SaveChanges();
                foreach (var criterionName in settings.Criterion)
                {
                    var criterion = new Criterion(){CriterionName =
criterionName, TargetId = target.Id};
                    db.Criterion.Add(criterion);
                    db.SaveChanges();
                }
                foreach (var alternativeName in settings.Alternatives)
                {
                    var alternatives = new Alternatives() { AlternativesName =
alternativeName, TargetId = target.Id };
                    db.Alternatives.Add(alternatives);
                    db.SaveChanges();
                    addedAlt.Add(alternatives);
                }
                for (int i = 0; i < settings.CriterionMatrixDouble.Count; i++)
                {
                    for (int j = 0; j <settings.CriterionMatrixDouble[i].Count;
j++)
                    {
                        var matrix = new Matrix() { TargetId = target.Id,
RowIndex = i, ColumnIndex = j, Value = settings.CriterionMatrixDouble[i][j] };
                        db.Matrix.Add(matrix);
                        db.SaveChanges();
                    }
                }
                for (int i = 0; i < settings.AlternativeMatrixDouble.Count; i++)
                {
                    var alternativeMatrix = new AlternativesMatrixList() {
TargetId = target.Id, IndexValue = i };
                    db.AlternativesMatrixList.Add(alternativeMatrix);
                    db.SaveChanges();
```

```
                    //var alternatives = new Alternatives() { AlternativesName =
settings.Alternatives[i], TargetId = target.Id };
                    //db.Alternatives.Add(alternatives);
                    //db.SaveChanges();
                    for (int j = 0; j <
settings.AlternativeMatrixDouble[i].Count; j++)
                    {
                        for (int k = 0; k <
settings.AlternativeMatrixDouble[i][j].Count; k++)
                        {
                            var matrix = new Matrix() { AlternativeListId =
alternativeMatrix.Id, RowIndex = j, ColumnIndex = k, Value =
settings.AlternativeMatrixDouble[i][j][k] };
                            db.Matrix.Add(matrix);
                            db.SaveChanges();
                        }
                    }
                }


            }

        }
public void DeleteTarget(TargetModel targetModel)
        {
            using (var db = new MAIContext())
            {
                var target = db.Target.FirstOrDefault(x => x.Id ==
targetModel.Id);
                db.Target.Remove(target);
                db.SaveChanges();
            }
        }

private int GetCriterionSizeMatrix(List<Matrix> list)
        {
            int count = 0;
            while (list[count+1]!= list[count])
            {
                count++;
            }
            return count;
        }
 private List<List<Double>> GetVoidMatrix(int size)
        {
            List<List<double>> result = new List<List<double>>();
            for (int i = 0; i < size; i++)
            {
                var list = new List<double>();
                for (int j = 0; j < size; j++)
                {
                    list.Add(0);
                }
                result.Add(list);
            }
            return result;
        }

        public List<TargetModel> GetAllTarget()
        {
            using (var db = new MAIContext())
            {
                var targetList = db.Target;
                var result = new List<TargetModel>();
                foreach (var target in targetList)
                {
                    var targetModel = new TargetModel(){Id=target.Id, TargetName
= target.TargetName};
                    result.Add(targetModel);
```

```
                }
                return result;
            }
        }
 public Settings GetSettigsFromDb(TargetModel targetModel)
        {
            using (var db = new MAIContext())
            {
                //var target = db.Target.FirstOrDefault(x => x.Id ==
targetModel.Id);
                var settings = new Settings();
                var criterionNameList = db.Criterion.Where(x => x.TargetId ==
targetModel.Id).ToList();
                foreach (var criterion in criterionNameList)
                {
                    settings.Criterion.Add(criterion.CriterionName);
                }
                var alternativesNameList = db.Alternatives.Where(x => x.TargetId
== targetModel.Id);
                foreach (var alternative in alternativesNameList)
                {
                    settings.Alternatives.Add(alternative.AlternativesName);
                }
                //Инициализируем матрицу критериев
                var criterionList = db.Matrix.Where(x => x.TargetId ==
targetModel.Id).ToList();
                var size = criterionNameList.Count();  //
GetCriterionSizeMatrix(criterionList);
                settings.CriterionMatrixDouble = GetVoidMatrix(size);
                foreach (var criterion in criterionList)
                {

settings.CriterionMatrixDouble[criterion.RowIndex.Value][criterion.ColumnIndex.V
alue] =
                        criterion.Value.Value;
                }

                //Инициализируем матрицу альтернатив
                var alternativeMatrixList = db.AlternativesMatrixList.Where(x =>
x.TargetId == targetModel.Id);
                foreach (var alternatives in alternativeMatrixList)
                {
                    var matrix = GetVoidMatrix(alternativesNameList.Count());
                    var alternativesMatrix = db.Matrix.Where(x =>
x.AlternativeListId == alternatives.Id).ToList() ;
                    foreach (var alternative in alternativesMatrix)
                    {

matrix[alternative.RowIndex.Value][alternative.ColumnIndex.Value] =
alternative.Value.Value;
                    }
                    settings.AlternativeMatrixDouble.Add(matrix);

                }
                return settings;

            }
        }
    }
}

using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
```

```
namespace MAI
{
    /// <summary>
    /// Логіка взаємодії для WinElementName.xaml
    /// </summary>
    public partial class WinElementName : Window
    {

        public class ObjectForDataGrid
        {
            public ObjectForDataGrid(string dataString)
            {
                DataString = dataString;
            }
            public string DataString { get; set; }
        }
        List<ObjectForDataGrid> dataToGrid = new List<ObjectForDataGrid>();
        List<string> _dataList = new List<string>();

        public WinElementName(List<string> dataList, string elementString)
        {
            InitializeComponent();
            txtText.Content = elementString;
            _dataList = dataList;
            foreach (var dataString in dataList)
            {
                dataToGrid.Add(new ObjectForDataGrid(dataString));
            }
            dgShow.ItemsSource = dataToGrid;

        }


        private void btContinue_Click(object sender, RoutedEventArgs e)
        {
            if (dataToGrid.Any(x=>x.DataString ==""))
            {
                MessageBox.Show("Заповніть список для продовження!");
                return;
            }

            _dataList.Clear();
            foreach (var data in dataToGrid)
            {
                _dataList.Add(data.DataString);
            }
            Close();
        }

        private void dgShow_SelectionChanged(object sender,
SelectionChangedEventArgs e)
        {

        }

        private void dgShow_MouseDoubleClick(object sender, MouseButtonEventArgs
e)
        {


        }

        private void WinElementName_OnLoaded(object sender, RoutedEventArgs e)
        {

        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.Office.Interop.Word;
using Excel = Microsoft.Office.Interop.Excel;


    internal class ExcelWorker
    {


        private void PrintMatrix(Excel.Worksheet worksheet, ref int row,
List<List<double>> matrix, List<string>header, List<double>vectorList, string
symbol)
        {

            row++;
            int column = 1;
            //Печатаем критерии
            for (int i = 0; i < header.Count; i++)
            {
                worksheet.Cells[row, i + 2] = header[i];
                column = i + 2;
            }
            worksheet.Cells[row, column+3] = "Вектори матриці";
            row++;
            for (int i = 0; i < matrix.Count; i++)
            {
                worksheet.Cells[row, 1] = header[i];

                for (int j = 0; j < matrix[i].Count; j++)
                {
                    worksheet.Cells[row, j+2] = matrix[i][j];
                }
                worksheet.Cells[row, column+3] = $"{symbol}{i}=
{vectorList[i]}";
                row++;
            }
        }

    public partial class Form1 : Form
    {
        //Відкритий алфавіт
        string Alph =
"1234567890абвгдеєжзиіїйклмнопрстуфхцчшщьюяАБВГДЕЄЖЗИІЇЙКЛМНОПРСТУФХЦЧШЩЬЮЯ -
,.:;”!?";
        //змінна для збереження закритого алфавіту
        string SwAlph = "";
        public Form1()
        {
            InitializeComponent();
        }

        private void button2_Click(object sender, EventArgs e)
        {
            OpenFileDialog openFileDialog1 = new OpenFileDialog();
            openFileDialog1.Filter = "txt files (*.txt)|*.txt|All files
(*.*)|*.*";
            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                textBox3.Text = File.ReadAllText(openFileDialog1.FileName,
Encoding.Default);
            }
        }
```

```
        private void button7_Click(object sender, EventArgs e)
        {
            OpenFileDialog openFileDialog1 = new OpenFileDialog();
            openFileDialog1.Filter = "txt files (*.txt)|*.txt|All files
(*.*)|*.*";
            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                textBox4.Text = File.ReadAllText(openFileDialog1.FileName,
Encoding.Default);
            }
        }


        private void PrintMatrixAlternatives(Excel.Worksheet worksheet, ref int
row, Settings settings)
        {
            //Печатаем критерии
            int column = 1;
            for (int i = 0; i < settings.Criterion.Count; i++)
            {
                worksheet.Cells[row, i + 2] = settings.Criterion[i];
                worksheet.Cells[row+1, i + 2] = settings.CriterionVectors[i];
                column = i + 2;
            }
            worksheet.Cells[row, column + 3] = "Загальні приорітети";
            row += 2;
            for (int i = 0; i < settings.Alternatives.Count; i++)
            {
                worksheet.Cells[row, 1] = settings.Alternatives[i];
                for (int j = 0; j < settings.AlternativesVectors.Count; j++)
                {
                    worksheet.Cells[row, j + 2] =
settings.AlternativesVectors[j][i];

                }
                worksheet.Cells[row, column + 3] = $"λ{i}  =
{settings.AlphaList[i]}";
                row++;
            }
   internal void CreateReport(Settings settings)
        {

            Excel._Application app = new
Microsoft.Office.Interop.Excel.Application();
            Excel._Workbook workbook = app.Workbooks.Add(Type.Missing);
            app.DisplayAlerts = false;
            app.Interactive = false;
            app.Visible = true;
            var row = 1;

            var worksheet = (Excel.Worksheet)workbook.Sheets.Add();
}


namespace CriptoSwap
{
    public partial class Form1 : Form
    {
        //Відкритий алфавіт
        string Alph =
"1234567890абвгдеєжзиіїйклмнопрстуфхцчшщьюяАБВГДЕЄЖЗИІЇЙКЛМНОПРСТУФХЦЧШЩЬЮЯ -
,.:;”!?";
        //змінна для збереження закритого алфавіту
        string SwAlph = "";
        public Form1()
        {
            InitializeComponent();
        }
```

```
        private void button2_Click(object sender, EventArgs e)
        {
            OpenFileDialog openFileDialog1 = new OpenFileDialog();
            openFileDialog1.Filter = "txt files (*.txt)|*.txt|All files
(*.*)|*.*";
            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                textBox3.Text = File.ReadAllText(openFileDialog1.FileName,
Encoding.Default);
            }
        }

        private void button7_Click(object sender, EventArgs e)
        {
            OpenFileDialog openFileDialog1 = new OpenFileDialog();
            openFileDialog1.Filter = "txt files (*.txt)|*.txt|All files
(*.*)|*.*";
            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                textBox4.Text = File.ReadAllText(openFileDialog1.FileName,
Encoding.Default);
            }
        }

        private void button3_Click(object sender, EventArgs e)
        {
            SaveFileDialog saveFileDialog = new SaveFileDialog();
            saveFileDialog.Filter = "Текстовый документ (*.txt)|*.txt|Все файлы
(*.*)|*.*";

            if (saveFileDialog.ShowDialog() == DialogResult.OK)
            {
                StreamWriter streamWriter = new
StreamWriter(saveFileDialog.FileName);
                streamWriter.WriteLine(textBox3.Text);
                streamWriter.Close();
            }
        }

        private void button6_Click(object sender, EventArgs e)
        {
            SaveFileDialog saveFileDialog = new SaveFileDialog();
            saveFileDialog.Filter = "Текстовый документ (*.txt)|*.txt|Все файлы
(*.*)|*.*";

            if (saveFileDialog.ShowDialog() == DialogResult.OK)
            {
                StreamWriter streamWriter = new
StreamWriter(saveFileDialog.FileName);
                streamWriter.WriteLine(textBox4.Text);
                streamWriter.Close();
            }
        }

       //генерація підстановочної таблиці
        private void button1_Click(object sender, EventArgs e)
        {
            int cod;
            int key = 0;
            string key_st = textBox1.Text;//отримати ключ

            for (int i = 0; i < key_st.Length; i++)
                key += (int)key_st[i];
            Random rnd = new Random(key);
            SwAlph = "";
            // Генерація таблиці (набір символів для підстановки)
            for (int i = 0; i < Alph.Length; i++)
            {
```

```
                cod = 32 + rnd.Next(223);
                while (SwAlph.IndexOf((char)cod) >= 0)
                {
                    cod = 32 + rnd.Next(223);
                }
                SwAlph += (char)cod;
            }
            // вивести отриману таблицю
            textBox2.Text = SwAlph;
        }
        // функція шифрування
        private void button4_Click(object sender, EventArgs e)
        {
            string swtmp = "";
            int p;
            for (int i = 0; i <textBox3.Text.Length; i++)// прохід по всім
символам
            {
              p = Alph.IndexOf(textBox3.Text[i]);// знайти входження (позицію)
символа в відкрит. алфавіті
              if (p >= 0)
               {
                 swtmp += SwAlph[p];// замінити на відповідний символ закритого
алфавіта
               }
              else
               {
                swtmp += textBox3.Text[i];// символи що не входять в Alph (осн.
алфавіт)
               }
            }
            textBox4.Text = swtmp;
        }

        private void button5_Click(object sender, EventArgs e)
        {
            string swtmp = "";
            int p;
            for (int i = 0; i < textBox4.Text.Length; i++)
            {
                p = SwAlph.IndexOf(textBox4.Text[i]);
                if (p >= 0)
                {
                    swtmp += Alph[p];
                }
                else
                {
                    swtmp += textBox4.Text[i];// символи що не входять в Alph
(осн. алфавіт)
                }
            }
            textBox3.Text = swtmp;
        }
    }
}
```