Міністерство освіти і науки, молоді та спорту України

Харківський національний університет радіоелектроніки

Факультет Комп’ютерних наук

Кафедра Програмної інженерії

КУРСОВА РОБОТА

ПОЯСНЮВАЛЬНА ЗАПИСКА

Тема: Додаток для шифрування даних

з дисципліни “ Технології компонентного програмного забезпечення”

Керівник: к.т.н., доц. Лещинський В.О.

студент гр. ПІ-09-1 Горемикін І. Д.

Комісія:

Зав. каф. ПІ

к. т. н, проф. Дудар З.В.

к.т.н., доц. Лещинський В.О.

ас. Лещинська І. О.

Харків 2012

РЕФЕРАТ

Пояснювальна записка містить 28 сторінки, 7 рисунків.

Мета роботи: Спроектувати та розробити додаток для шифрування даних. Навчитися писати програмні продукти дотримуючись гайдлайнів.

Результат: Програма для шифрування розроблена у середовищі Microsoft Visual Studio з використанням мови C#, драйверу Dokan та бібліотеки шифрування Bouncy Castle Crypto.

C#, Dokan, Bouncy Castle Crypto, ДОДАТОК, ШИФРУВАННЯ, СЕРВІС ОН-ЛАЙН СХОВИЩА.

ЗМІСТ

ВСТУП...………….……………………………...………………….…………......................4

1 АНАЛІЗ ПРИДМЕТНОЇ ГАЛУЗІ……………………………………………………..….5

2 ПОСТАНОВКА ЗАДАЧІ……………………………………………………………….....6

3 МОДЕЛЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ…………… …………………7

4 ОПИС ПРОГРАМИ……………………………………………………………..................9

4.1 ЗАГАЛЬНІ ВІДОМОСТІ……………………………………………………..…9

4.2 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ …………………………………………..9

5 ОПИС ПРОЕКТУ..………………………………………………...……………………....13

ВИСНОВКИ………………………………………………………...……………………….27

ПЕРЕЛІК ПОСИЛАНЬ……………………………………………………………………..28

ВСТУП

В сучасному суспільстві місце, яке займає цифрова інформація, дуже велике. С кожним днем все більше людей користуються комп’ютерами, мобільними телефонами, планшетами та зберігають на цифрових носіях свої фотографії, документи та інші важливі персональні дані. Також с кожним днем все більше фірм здійснює документообіг у цифровому вигляді.

Збільшення швидкості доступу до Інтернету призвело до появи таких сервісів як: Dropbox, Google Drive, SkyDrive, Box, SugarSync та інших, що дозволяють зберігати персональні дані в мережі Інтернет. Це справді дуже зручно, тому що надає можливість синхронізувати дані між різними комп’ютерами та іншими цифровими пристроями. Або використовувати ці сервіси для резервного копіювання та збереження важливих документів, фотографій та іншого.

Але є інформація яка повинна залишатися конфіденційною. Це можуть бути як персональні дані окремої людини, так і важливі документи деякої фірми. Так, наприклад, якщо секретна інформація однієї фірми потрапить до конкурентів, це може призвести до дуже великих втрат, або навіть банкротства. Тому постає дуже важливе питання забезпечення безпеки. Не залежно від того використовуємо ми сторонні Інтернет сервіси або зберігаємо інформацію на жорсткому диску персонального комп’ютера.

Саме на забезпечення безпеки та конфіденційності даних, націлена, вибрана мною, тема курсового проекту.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

Предметна галузь являє собою комп’ютерний додаток, який дозволяє користувачу шифрувати необхідні дані.

В наш час стали дуже популярні он-лайн сховища, але їх проблема безпеки в тому, що вони зберігають дані в нешифрованому вигляді. Для виправлення цього недоліку створені такі програми, як BoxCryptor та SecretSync.

BoxCryptor дозволяє створювати віртуальний жорсткий диск. Кожному віртуальному жорсткому диску відповідає директорія-джерело. Після монтування, кожен файл в віртуальному жорсткому диску відповідає зашифрованому файлу в директорії.

Он-лайн сховища створюють на комп’ютері директорію, яку потім і синхронізують, тому директорію-джерело потрібно створювати саме в ній.

Після того я користувач додає файл в віртуальний жорсткий диск, BoxCryptor шифрує цей файл в директорію-джерело, а сервіс он-лайн сховища синхронізує його. Для доступу до цього файлу на іншому комп’ютері проходить зворотній процес – он-лайн сервіс синхронізує зашифрований файл у директорію-джерело на іншому комп’ютері, а BoxCryptor відображає його розшифровану копію в віртуальному жорсткому диску.

У програми BoxCryptor є корисна особливість, при роботі з таким сервісом он-лайн сховища, як Dropbox. Коли користувач вперше запустив BoxCryptor на даному комп’ютері, при створенні віртуального жорсткого диску, вона автоматично знаходить папку синхронізації та пропонує створити директорію-джерело саме в цій папці.

Програма SecretSync працює аналогічно, за одним винятком. Замість створення віртуального жорсткого диску, вона просто створює спеціальну папку в якій відображаються розшифровані файли з директорії-джерела.

Обидва додатки використовують Advanced Encryption Standard (AES), також відомий під назвою Rijndael — симетричний алгоритм блочного шифрування (розмір блока 128 біт, ключ 128/192/256 біт), фіналіст конкурсу AES і прийнятий в якості американського стандарту шифрування урядом [США](http://uk.wikipedia.org/wiki/%D0%A1%D0%A8%D0%90).

Недоліком програми BoxCryptor є те, що безкоштовна версія дозволяє створювати тільки один віртуальний жорсткий диск об’ємом 2 Гб.

2 ПОСТАНОВКА ЗАДАЧІ

В даній роботі необхідно реалізувати програму для шифрування персональних даних. Програмний продукт повинен бути реалізований за допомогою мови програмування C# та платформи .NET. Також, для створення віртуальних жорстких дисків, повинен використовуватися драйвер Dokan, а для шифрування – бібліотека Bouncy Castle Crypto.

Зробивши аналіз аналогічних систем, був зроблений висновок, що додаток повинен мати наступні функціональні вимоги:

* давати можливість користувачам створювати віртуальний жорсткий диск;
* давати можливість призначати точку монтування;
* давати можливість обирати мітку тому;
* давати можливість знищувати віртуальний жорсткий диск;
* давати можливість захищати віртуальний жорсткий диск паролем;
* давати можливість змінювати пароль диску;
* давати можливість тимчасово відключати диск;
* давати можливість підключати віртуальний жорсткий диск до вже існуючої директорії-джерела.

Не функціональні вимоги:

* приємне поєднання кольорів в інтерфейсі;
* розташування елементів управління згідно принципів юзабіліті;
* додаток повинен працювати з платформою .NET Framework версії 2.0 або більше.

3 МОДЕЛЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Діаграма прецедентів (рисунок 3.1) - це діаграма, на якій зображено відношення між акторами та прецедентами. В даному проекті акторами є користувачі системи. Прецедентами є ті дії, які користувачі можуть виконувати в системі.



Рисунок 3.1 - Діаграма прецедентів (Use Case)

Дана діаграма повністю відображає функціональні вимоги до системи. Користувач може створювати віртуальні жорсткі диски. Для цього йому потрібно буде призначити директорію-джерело, обрати точку монтування, призначити мітку тому, призначити пароль для захисту диску. Також користувач може відключати, підключати, видаляти диски та змінювати пароль для доступу кожному з них.

Діаграма послідовності — це діаграма, яка відображає взаємодії об'єктів впорядкованих за часом. Діаграма послідовності показана на рисунку 3.2

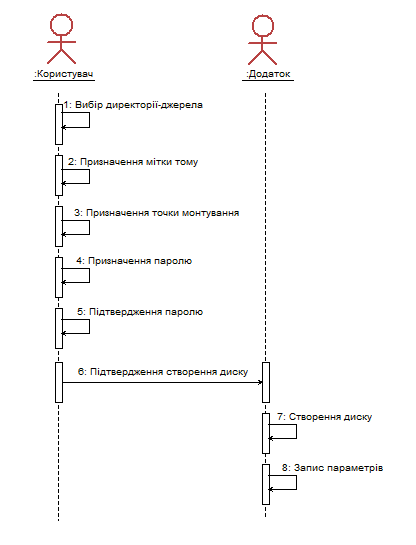


Рисунок 3.2 - Діаграма послідовності

На даній діаграмі зображено процес створення нового віртуального жорсткого диску. Користувач вводить всі необхідні дані, як шлях до директорії-джерела, мітку тому, точку монтування, пароль, підтвердження паролю та підтверджує свій намір створити диск. Після чого додаток створює новий віртуальний жорсткий диск на основі введених даних користувачем.

4 ОПИС ПРОГРАМИ

4.1 Загальні відомості

Для написання додатку для шифрування даних була обрана середа розробки програмного забезпечення Microsoft Visual Studio, мова програмування C#, драйвер файлової системи Dokan та бібліотека шифрування Bouncy Castle Crypto.

Основними критеріями вибору даної мови програмування стало те, що C# - мова програмування, яка активно розвивається на платформі .NET та на даний момент є найкращою мовою для створення надійного та стійкого програмного забезпечення, з максимальною швидкодією та мінімальною кількістю помилок і збоїв.

Драйвер Dokan був обраний тому, що це єдиний драйвер для платформи Windows функціональність, якого повністю забезпечує всі потреби розробки даного додатку. Також він повністю безкоштовний.

Бібліотека Bouncy Castle Crypto була обрана тому, що вона має дуже багатий вибір різноманітних алгоритмів шифрування та вона також повністю безкоштовна.

4.2 Опис програмної реалізації

Головне вікно програми зображено на рисунку 4.1. В даному вікні знаходиться панель інструментів, що забезпечує функції створення віртуальних жорстких дисків, видалення, підключення та відключення дисків. Також за допомогою панелі інструментів можна змінити пароль диску та відкрити вікно з інформацією про програму та її автора.

Головне вікно програми дозволяє продивлятися список створених віртуальних жорстких дисків, а також інформацію по кожному з них, а тобто: мітку тому, точку монтування, підключених чи відключений конкретний диск та шлях до директорії-джерела.

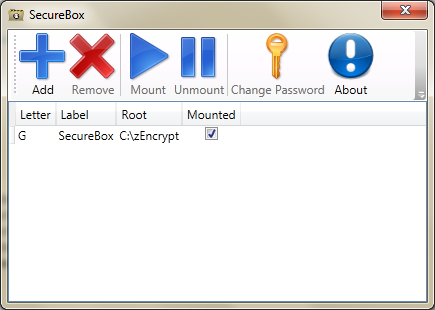
. 

Рисунок 4.1 – Головне вікно додатку

Коли користувач натисне на панелі інструментів кнопку створення нового віртуального жорсткого диску, він побачить вікно показане на рисунку 4.2.

Для того, щоб створити нових віртуальний жорсткий диск користувачеві потрібно, в поле під надписом «Choose the location of the new encrypted folder», ввести шлях до нової директорії-джерела, або натиснувши кнопку «Browse…» вибрати його за допомогою вікна вибору каталогу. Треба зауважити, що корінь вибраного каталогу буде служити директорією-джерелом, а отже саме в корні цього каталогу буде створений файл з настройками та саме в корні цього каталогу будуть знаходитися зашифровані файли.

Далі користувачеві потрібно ввести мітку тому в поле після надпису «Label for the SecureBox drive». Потім з випадаючого списку обрати точку монтування нового диску. Щоб створити новий жорсткий диск користувачеві ще потрібно ввести пароль та підтвердження паролю. Якщо всі дані введені вірно, після натискання кнопки «OK», програма створить новий жорсткий диск, автоматично примонтує його та додасть до списку дисків у головному вікні.

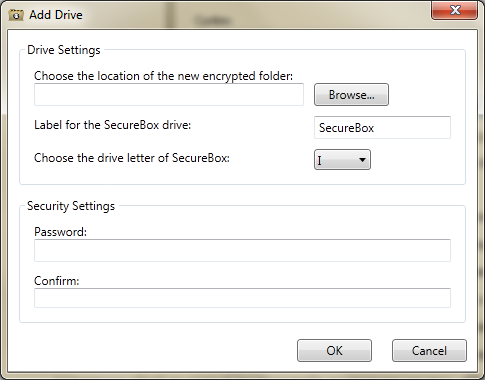


Рисунок 4.2 – Вікно створення диску

Якщо, при введені були недопустимі дані, після натискання кнопки «OK», додаток висвітить нове вікно з описом допущених помилок користувачем. Якщо користувач під час введення даних з якихось причин передумає, він може перервати операцію створення диску, натиснувши кнопку «Cancel».

Якщо користувач захоче змінити пароль для вже існуючого диску, він може в головному вікні на панелі інструментів натиснути кнопку «Change Password». Відкриється вікно зображене на рисунку 4.3.

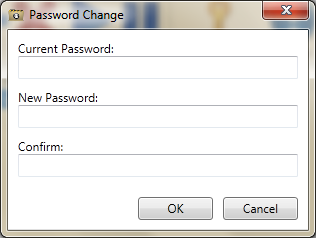


Рисунок 4.3 – Вікно для зміни паролю

Потрібно ввести поточний пароль, потім ввести бажаємий новий пароль та підтвердити його, після чого натиснути кнопку «OK». Якщо всі дані були введені правильно відбудеться зміна паролю.

Якщо користувач захоче отримати інформацію про автора програми або її версію, йому потрібно на панелі інструментів натиснути кнопку «About», після чого відкриється вікно зображене на рисунку 4.4.



Рисунок 4.4 – Вікно з інформацією о програмі

В даному вікні міститься інформація з назвою програми, версією та іменем автора.

5 ОПИС ПРОЕКТУ

Структура даного проекту складається з двох слоїв:

* Business Layer;
* User Interface Layer.

В каталозі Business Layer знаходяться класи бізнес логіки, а саме:

* CryptoMirror – клас, який відповідає за функціонування віртуального жорсткого диску;
* DriveInfo – клас-сутність, який відображає інформацію об існуючім диску;
* SecureBox – цей клас представляє собою фасад бізнес-логіки, саме з ним спілкується слой User Interface Layer;
* Encryptor – надає засоби для шифрування та дешифрування даних, а також для генерації ключа;
* XmlAppSettings – відповідає за запис та зчитування налаштування додатку;
* XmlDriveSettings – відповідає за запис та зчитування налаштування каталогу-джерела.

В каталозі User Interface Layer знаходяться класи слою користувацького інтерфейсу, а саме:

* MainWindow – головне вікно програми;
* AddDrive – вікно створення нового віртуального жорсткого диску;
* PasswordChange – вікно зміни паролю;
* About – вікно з інформацією про додаток.

Схема бізнес-логіки представлена на рисунку 5.1.

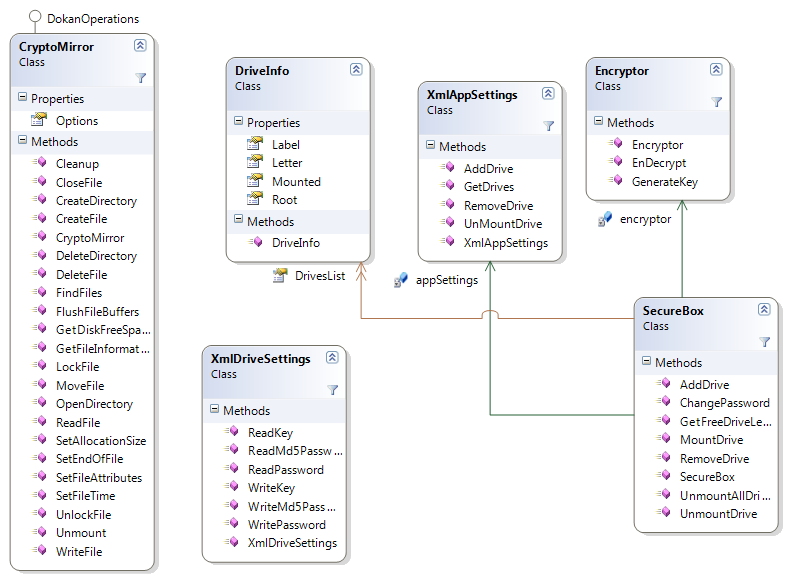


Рисунок 5.1 – Схема бізнес-логіки

Але на ній показані не всі зв’язки між класами. Так екземпляр класу CryptoMirror створюється кожний раз класом SecureBox, при запиті на створення нового віртуального жорсткого диску. В цей же час створюється екземпляр класу XmlDriveSettings, який створює файл «.settings.xml» в директорії-джерелі. В цьому файлі зберігаються пароль та ключ шифрування файлів в директорії.

Для створення віртуального жорсткого диску використовується драйвер Dokan та .NET обгортка DokanNet, яка дозволяє створювати свою файлову систему в managed-коді. Для цього потрібно реалізувати інтерфейс DokanOperations та передати його реалізацію, як параметр методу DokanMain бібліотеки DokanNet.

Реалізація інтерфейсу DokanOperations відповідає класу CryptoMirror та зводиться до реалізації наступних методів:

* CreateFile – метод який викликається для створення файлів та директорій. Його реалізація наведена на наступній сторінці.

public int CreateFile(String filename, FileAccess access, FileShare share,

FileMode mode, FileOptions options, DokanFileInfo info)

{

string path = GetPath(filename);

try

{

if (Directory.Exists(path))

{

info.IsDirectory = true;

}

else

{

FileStream fs = new FileStream(path, mode, access, share,

bufferSize, options);

fs.Close();

}

return DokanNet.DOKAN\_SUCCESS;

}

catch (Exception e)

{

return -DokanNet.DOKAN\_ERROR;

}

}

Параметр методу filename – це назва файлу, який необхідно створити. Інші параметри представляють собою параметри створення файлу;

* + OpenDirectory – відкриття директорії. В цьому методі необхідно реалізувати функцію відкриття директорії відповідно до того, як вона повинна бути реалізована в вашій файловій системі. Реалізація, для даного проекту, наведена нижче:

public int OpenDirectory(String filename, DokanFileInfo info)

{

if (Directory.Exists(GetPath(filename)))

{

return DokanNet.DOKAN\_SUCCESS;

}

return -DokanNet.ERROR\_PATH\_NOT\_FOUND;

}

В функцію передається параметр з ім’ям директорії, яка повинна бути відкрита. Параметр info типу DokanFileInfo – це параметр для внутрішніх потреб драйверу, тому його можна не використовувати при реалізації методів інтерфейсу DokanOperations;

* CreateDirectory – створення директорії. Код наведений нижче:

public int CreateDirectory(String filename, DokanFileInfo info)

{

if (!Directory.Exists(GetPath(filename)))

{

try

{

Directory.CreateDirectory(GetPath(filename));

}

catch (Exception)

{

return -DokanNet.DOKAN\_ERROR;

}

return DokanNet.DOKAN\_SUCCESS;

}

return -DokanNet.ERROR\_ALREADY\_EXISTS;

}

Той же самий параметр, що й в попередньому методі;

* Cleanup – видалення файлу або директорії. Для нашої файлової системи метод завжди повертає значення успішного завершення роботи;
* CloseFile – закриття файлового дескриптору. Повертаємо значення таке ж саме, як і для попереднього методу;
* ReadFile – зчитування частини файлу певної довжини з певним зміщенням. Реалізація:

public int ReadFile(String filename, Byte[] buffer, ref uint readBytes,

long offset, DokanFileInfo info)

{

string path = GetPath(filename);

try

{

long encOffset = offset - offset % cryptBlockSize;

if (encOffset >= cryptBlockSize && encOffset % writeBlockSize != 0)

{

encOffset -= cryptBlockSize;

}

long offsetDif = offset - encOffset;

FileStream fs = new FileStream(path, FileMode.Open, FileAccess.Read, FileShare.ReadWrite);

fs.Seek(encOffset, SeekOrigin.Begin);

Byte[] crpBuff = new Byte[buffer.Length + offsetDif];

readBytes = (uint)fs.Read(crpBuff, 0, crpBuff.Length);

readBytes -= (uint)offsetDif;

fs.Close();

Array.Copy(encryptor.EnDecrypt(false, crpBuff), offsetDif, buffer, 0, buffer.Length);

return DokanNet.DOKAN\_SUCCESS;

}

catch (Exception e)

{

Console.Error.WriteLine("ReadFile exception \"{0}\": {1}", filename, e.Message);

return DokanNet.DOKAN\_ERROR;

}

}

Реалізація цього методу досить складна, тому що тут відбувається не тільки зчитування файлу з файлової системи. Після того, як буде зчитаний необхідна частина файлу, потрібно зробити розшифровування за допомогою методу EnDecrypt класу Encryptor;

* WriteFile – метод аналогічний методу ReadFile, але служить для запису даних в файл. Реалізація:

public int WriteFile(String filename, Byte[] buffer,

ref uint writtenBytes, long offset, DokanFileInfo info)

{

string path = GetPath(filename);

try

{

FileStream fs = new FileStream(path, FileMode.OpenOrCreate, FileAccess.Write, FileShare.ReadWrite);

if (info.WriteToEndOfFile)

{

fs.Seek(0, SeekOrigin.End);

}

else

{

fs.Seek(offset, SeekOrigin.Begin);

}

Byte[] crpBuff = encryptor.EnDecrypt(true, buffer);

fs.Write(crpBuff, 0, crpBuff.Length);

writtenBytes = (uint)crpBuff.Length;

fs.Close();

return DokanNet.DOKAN\_SUCCESS;

}

catch (Exception e)

{

Console.Error.WriteLine("Write exception \"{0}\": {0}", filename, e.Message);

return DokanNet.DOKAN\_ERROR;

}

}

Перед тим, як записати дані до файлу, відбувається їх зашифровування за допомогою того ж самого методу EnDecrypt, але вже з параметром який вказує на те що необхідно зробити шифрування;

* FlushFileBuffers – метод в якому відбувається очистка буферу файла;
* GetFileInformation – повертає інформацію про файл або директорію. Реалізація:

public int GetFileInformation(String filename, FileInformation fileinfo,

DokanFileInfo info)

{

string path = GetPath(filename);

if (File.Exists(path))

{

FileInfo f = new FileInfo(path);

fileinfo.Attributes = f.Attributes;

fileinfo.CreationTime = f.CreationTime;

fileinfo.LastAccessTime = f.LastAccessTime;

fileinfo.LastWriteTime = f.LastWriteTime;

fileinfo.Length = f.Length;

return DokanNet.DOKAN\_SUCCESS;

}

else if (Directory.Exists(path))

{

DirectoryInfo f = new DirectoryInfo(path);

fileinfo.Attributes = f.Attributes;

fileinfo.CreationTime = f.CreationTime;

fileinfo.LastAccessTime = f.LastAccessTime;

fileinfo.LastWriteTime = f.LastWriteTime;

fileinfo.Length = 0;// f.Length;

return DokanNet.DOKAN\_SUCCESS;

}

return DokanNet.DOKAN\_ERROR;

}

Якщо файл знайдений, то метод просто зчитує його атрибути. Якщо файл не існує, перевіряється наявність каталогу з таким самим ім’ям, при його наявності також зчитуються його атрибути. Якщо ж ні файл, ні директорія не існують, метод повертає помилку;

* FindFiles – отримання списку файлів та каталогів в директорії. Його реалізація:

public int FindFiles(String filename, ArrayList files, DokanFileInfo info)

{

string path = GetPath(filename);

if (Directory.Exists(path))

{

DirectoryInfo d = new DirectoryInfo(path);

FileSystemInfo[] entries = d.GetFileSystemInfos();

foreach (FileSystemInfo f in entries)

{

if (f.Name == settingsFile)

{

continue;

}

FileInformation fi = new FileInformation();

fi.Attributes = f.Attributes;

fi.CreationTime = f.CreationTime;

fi.LastAccessTime = f.LastAccessTime;

fi.LastWriteTime = f.LastWriteTime;

fi.Length = (f is DirectoryInfo) ? 0 : ((FileInfo)f).Length;

fi.FileName = f.Name;

files.Add(fi);

}

return DokanNet.DOKAN\_SUCCESS;

}

return DokanNet.DOKAN\_ERROR;

}

Передається ім’я директорії в якій необхідно зробити пошук, та масив в який необхідно занести об’єкти класу FileSystemInfo для знайдених файлів або директорій;

* SetFileAttributes – установка атрибутів файлу або директорії. Реалізація:

public int SetFileAttributes(String filename, FileAttributes attr, DokanFileInfo info)

{

string path = GetPath(filename);

try

{

if (File.Exists(path))

{

File.SetAttributes(path, attr);

return DokanNet.DOKAN\_SUCCESS;

}

else if (Directory.Exists(path))

{

DirectoryInfo di = new DirectoryInfo(path);

di.Attributes = attr;

return DokanNet.DOKAN\_SUCCESS;

}

}

catch (Exception e)

{

return -DokanNet.DOKAN\_ERROR;

}

return DokanNet.DOKAN\_SUCCESS;

}

Відбувається пошук файлу або директорії з заданим ім’ям та встановлення заданих атрибутів;

* SetFileTime – встановлення часу створення, останнього доступу, модифікації файлу або директорії. Реалізація:

public int SetFileTime(String filename, DateTime ctime,

DateTime atime, DateTime mtime, DokanFileInfo info)

{

string path = GetPath(filename);

try

{

if (File.Exists(path))

{

if (ctime == DateTime.MinValue)

ctime = File.GetCreationTime(path);

if (atime == DateTime.MinValue)

atime = File.GetLastAccessTime(path);

if (mtime == DateTime.MinValue)

mtime = File.GetLastWriteTime(path);

File.SetCreationTime(path, ctime);

File.SetLastAccessTime(path, atime);

File.SetLastWriteTime(path, mtime);

return DokanNet.DOKAN\_SUCCESS;

}

else if (Directory.Exists(path))

{

if (ctime == DateTime.MinValue)

ctime = Directory.GetCreationTime(path);

if (atime == DateTime.MinValue)

atime = Directory.GetLastAccessTime(path);

if (mtime == DateTime.MinValue)

mtime = Directory.GetLastWriteTime(path);

Directory.SetCreationTime(path, ctime);

Directory.SetLastAccessTime(path, atime);

Directory.SetLastWriteTime(path, mtime);

return DokanNet.DOKAN\_SUCCESS;

}

return DokanNet.DOKAN\_ERROR;

}

catch (Exception e)

{

Console.Error.WriteLine("FileTime exception: {0}", e.Message);

return -DokanNet.DOKAN\_ERROR;

}

}

Відбувається пошук необхідного файлу або директорії а потім встановлення необхідних атрибутів;

* DeleteFile – метод для видалення файлу. Реалізація:

public int DeleteFile(String filename, DokanFileInfo info)

{

string path = GetPath(filename);

if (File.Exists(path))

{

try

{

File.Delete(path);

}

catch (Exception e)

{

Console.WriteLine(e.Message);

return DokanNet.DOKAN\_ERROR;

}

return DokanNet.DOKAN\_SUCCESS;

}

return -DokanNet.ERROR\_FILE\_NOT\_FOUND;

}

Відбувається пошук необхідного файлу та його видалення;

* DeleteDirectory – метод видалення директорії. Реалізація:

public int DeleteDirectory(String filename, DokanFileInfo info)

{

string path = GetPath(filename);

if (Directory.Exists(path))

{

try

{

Directory.Delete(path, true);

}

catch (Exception e)

{

Console.WriteLine(e.Message);

return DokanNet.DOKAN\_ERROR;

}

return DokanNet.DOKAN\_SUCCESS;

}

return -DokanNet.ERROR\_PATH\_NOT\_FOUND; ;

}

Працює аналогічно попередньому методу, але для директорії;

* MoveFile – метод в якому відбувається переміщення файлу або директорії. Реалізація:

public int MoveFile(String filename, String newname, bool replace, DokanFileInfo info)

{

string sPath = GetPath(filename);

string dPath = GetPath(newname);

if (!replace && File.Exists(dPath))

{

return -DokanNet.ERROR\_ALREADY\_EXISTS;

} else if (!replace && Directory.Exists(dPath))

{

return -DokanNet.ERROR\_PATH\_NOT\_FOUND;

}

try

{

if (File.Exists(sPath))

{

if (replace)

{

File.Replace(sPath, dPath, null);

}

else

{

File.Move(sPath, dPath);

}

return DokanNet.DOKAN\_SUCCESS;

}

else if (Directory.Exists(sPath))

{

DirectoryInfo k = new DirectoryInfo(sPath);

k.MoveTo(dPath);

return DokanNet.DOKAN\_SUCCESS;

}

}

catch (Exception e)

{

Console.Error.WriteLine("Move exception: {0}", e.Message);

return -DokanNet.DOKAN\_ERROR;

}

return DokanNet.DOKAN\_ERROR;

}

Спочатку відбувається пошук файлу або директорії, а потім відбувається його переміщення на нове розташування. Також цей метод використовується для перейменування файлів або директорій;

* SetEndOfFile – метод встановлює розмір файлу. Реалізація:

public int SetEndOfFile(String filename, long length, DokanFileInfo info)

{

string path = GetPath(filename);

try

{

FileStream fs = new FileStream(path, FileMode.Open, FileAccess.Write, FileShare.ReadWrite);

fs.SetLength(length);

fs.Close();

}

catch (Exception e)

{

Console.Error.WriteLine("SetEndOfFile exception: {0}", e.Message);

return -DokanNet.DOKAN\_ERROR;

}

return DokanNet.DOKAN\_SUCCESS;

}

Відбувається відкриття файлового потоку та встановлення кінця файлу;

* SetAllocationSize – встановлюється розмір виділення пам’яті для файлу;
* LockFile – метод для блокування файлу. Реалізація:

public int LockFile(String filename, long offset, long length, DokanFileInfo info)

{

string path = GetPath(filename);

try

{

FileStream fs = new FileStream(path, FileMode.Open, FileAccess.Write, FileShare.ReadWrite);

fs.Lock(offset, length);

fs.Close();

}

catch (Exception e)

{

Console.Error.WriteLine("SetEndOfFile exception: {0}", e.Message);

return -DokanNet.DOKAN\_ERROR;

}

return DokanNet.DOKAN\_SUCCESS;

}

Відбувається відкриття файлового потоку та блокування файлу;

* UnlockFile – метод для зняття блокування з файлу. Реалізація:

public int UnlockFile(String filename, long offset, long length, DokanFileInfo info)

{

string path = GetPath(filename);

try

{

FileStream fs = new FileStream(path, FileMode.Open, FileAccess.Write, FileShare.ReadWrite);

fs.Unlock(offset, length);

fs.Close();

}

catch (Exception e)

{

Console.Error.WriteLine("SetEndOfFile exception: {0}", e.Message);

return -DokanNet.DOKAN\_ERROR;

}

return DokanNet.DOKAN\_SUCCESS;

}

Метод працює аналогічно попередньому, але знімає блокування з файла;

* GetDiskFreeSpace – отримання інформації про кількість усього та доступного місця в файловій системі;
* Unmount – відключення віртуального жорсткого диску. В даній реалізації завжди повертає успішне значення.

Клас DriveInfo – відображає собою сутність віртуального диску. Його реалізація наведена нижче:

public class DriveInfo

{

public DriveInfo(char letter, string label,

string root, bool mounted)

{

Letter = char.ToUpper(letter);

Label = label;

Root = root;

Mounted = mounted;

}

public char Letter

{

get;

private set;

}

public string Label

{

get;

private set;

}

public string Root

{

get;

private set;

}

public bool Mounted

{

get;

private set;

}

}

Свойство Letter зберігає букву диску, свойство Label відображає мітку віртуального диску, свойство Root – директорію-джерело в якій знаходяться зашифровані файли. Свойство Mounted показує чи підключений на даний момент віртуальній жорсткий диск. Даний клас є загальним для слоїв Business Layer та User Interface Layer. Коли класи слою User Interface Layer посилають запити на підключення, відключення або видалення віртуального диску, вони передають в параметрі запиту саме клас DriveInfo для визначення конкретного диску.

Клас SecureBox виступає фасадом слою бізнес логіки. Саме цьому класу посилаються запити з слою користувацького інтерфейсу. Фактично тільки цей клас (за виключення класу DriveInfo, який являється загальним) видимий для користувача бізнес логіки. В ньому реалізовані такі методи:

* AddDrive – метод додавання нового віртуального диску. Його реалізація наведена нижче:

public bool AddDrive(DriveInfo drive, string password)

{

foreach (DriveInfo dInfo in DrivesList)

{

if (dInfo.Root == drive.Root ||

dInfo.Letter == drive.Letter)

{

return false;

}

}

Directory.CreateDirectory(drive.Root);

CreateDriveSettings(drive, password);

MountDrive(drive);

appSettings.AddDrive(drive);

DrivesList = appSettings.GetDrives();

return true;

}

Спочатку відбувається пошук серед вже існуючих дисків, якщо вже існує диск з подібною буквою або тією самою директорією-джерелом, метод повертає помилку. Якщо все добре, тоді метод створює файл з на лаштунками в директорії-джерелі, а потім створює новий віртуальний диск та додає його до списку вже існуючих;

* RemoveDrive – метод видаляє створений диск. Реалізація:

public bool RemoveDrive(DriveInfo drive)

{

if (ValidPassword(drive))

{

UnmountDrive(drive);

appSettings.RemoveDrive(drive);

DrivesList = appSettings.GetDrives();

return true;

}

return false;

}

Спочатку перевіряється справжність паролю, потім відключення диску, та видалення його зі списку існуючих;

* MountDrive – підключення диску. Реалізація:

public bool MountDrive(DriveInfo drive)

{

if (ValidPassword(drive))

{

CryptoMirror crptMirr = new CryptoMirror(drive.Root, drive.Letter, drive.Label);

Thread newThread = new Thread(CreateDrive);

newThread.Start(crptMirr);

threads.Add(drive.Letter, newThread);

appSettings.UnMountDrive(drive, true);

DrivesList = appSettings.GetDrives();

return true;

}

return false;

}

Створюється об’єкт класу CryptoMirror і передається як параметр функції при створенні нового потоку для нового віртуального диску;

* UnmountDrive – відключення диску. Реалізація:

public bool UnmountDrive(DriveInfo drive)

{

try

{

if (drive.Mounted)

{

DokanNet.DokanUnmount(drive.Letter);

threads[drive.Letter].Abort();

threads.Remove(drive.Letter);

appSettings.UnMountDrive(drive, false);

DrivesList = appSettings.GetDrives();

}

}

catch (Exception e)

{

Console.Error.WriteLine(e.Message);

return false;

}

return true;

}

Перевіряється чи підключений диск, потім викликається метод відключення диску, завершується поток, що відповідав за цей диск та видаляється зі списку потоків;

* UnmountAllDrives – метод відмонтовує всі жорсткі диски;
* ChangePassword – слугує для зміни паролю, яким захищений жорсткий диск; Реалізація:

public bool ChangePassword(DriveInfo drive, string oldPassword, string newPassword)

{

if (ValidPassword(drive))

{

string password = GetDrivePassword(drive);

if (password == oldPassword)

{

XmlDriveSettings dSettings = new XmlDriveSettings(drive.Root);

byte[] encPassword = encryptor.EnDecrypt(true, Encoding.Default.GetBytes(newPassword));

password = Encoding.Default.GetString(encPassword);

dSettings.WritePassword(password);

string md5Password = GetMd5Hash(newPassword);

dSettings.WriteMd5Password(md5Password);

return true;

}

}

return false;

}

Зчитується пароль з файлу налаштування директорії-джерела, зашифровується та зберігається пароль та його хеш-сума;

* GetFreeDriveLetters – метод повертає список літер, які можна використати для створення нового віртуального диску. Реалізація:

public char[] GetFreeDriveLetters()

{

System.IO.DriveInfo[] drives = System.IO.DriveInfo.GetDrives();

List<char> letters = new List<char>(driveLetters.ToCharArray());

foreach (System.IO.DriveInfo dInfo in drives)

{

letters.Remove(dInfo.Name[firstLetter]);

}

foreach (DriveInfo dInfo in DrivesList)

{

letters.Remove(dInfo.Letter);

}

return letters.ToArray();

}

Спочатку перевіряється чи не занята літера системою, а потім серед віртуальних дисків. Якщо все добре, то вона залишається серед списку доступних літер.

ВИСНОВКИ

В результаті розробки даного курсового проекту був отриманий додаток для шифрування даних на основі моделі системи представленої у третьому розділі. Даний програмний продукт був реалізований дотримуючись гайдлайнів.

При розробці даного продукту були зроблені наступні етапи:

* аналіз предметної галузі. Тут були проаналізовані подібні за предметною галуззю програми, де були винайдені їх недоліки та переваги;
* постановка задачі. Тут були описані основні функціональні та не функціональні вимоги розроблюваної системи;
* моделювання програмного забезпечення. Тут були описані дві діаграми. Перша діаграма (use case) показує взаємодію користувачів системи з самою системою, тобто які функції можуть виконувати користувачі в системі. Друга діаграма (діаграма послідовності) показує, яким чином відбувається процес створення нового жорсткого диску;
* опис програми. Цей етап призначений для описання розробленої системи. Він включає в себе опис всіх можливостей програми з представленням зображень усіх вікон;
* опис проекту. Тут була описана архітектура даного проекту, його класи, а також взаємодія класів в бізнес-логіки.

Розроблена система виявиться дуже корисною всім хто хоче зберігати свої дані конфіденційно та в безпеці. Хоча створений продукт виявився дуже якісний та виконує всі заплановані функції, є ще дуже багато шляхів його розвитку. Один з таких – автоматичне знаходження папок синхронізації для усіх відомих сервісів он-лайн сховищ, при умові їх знаходження в системі користувача, та запропонування встановлення директорії-джерела в корні однієї з цих папок.

ПЕРЕЛІК ПОСИЛАНЬ

1. Мак-Дональд, Мэтью, Фримен, Адам, Шпушта, Марио. Microsoft ASP.NEТ 4 с примерами на С# 2010 для профессионалов, 4-е изд. : [Текст]/ Мэтью Мак-Дональд, Адам Фримен, Марио Шпушта - М.: 000 "И.Д. Вильямс", 2011.- 1424 с.
2. Програмна специфікація Bouncy Castle Crypto [Электронный ресурс] – Режим доступа: www/URL: <http://www.bouncycastle.org/specifications.html>
3. Нейгел, Кристиан, Ивьен, Билл, Глинн, Джей, Уотсон, Карл, Скиннер, Морган. С# 2008 и платформа .NET 3.5 для профессионалов.: [Текст]/ Кристиан Нейгел, Билл Ивьен, Джейн Глинн, Карл Уотсон, Морган Скиннер — М.: ООО "И.Д. Вильяме", 2009. - 1392 с.
4. Леоненко, Александр. Самоучитель UML: [Текст]/ Александр Леоненко – СПБ.: БХВ-Питербург, 2004. – 373 с.