МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Благовещенский государственный педагогический университет»

Физико-математический факультет

Кафедра информатики и методики преподавания информатики

КУРСОВАЯ РАБОТА

на тему: Программная реализация системы защиты данных на основе стандарта AES

по дисциплине: Современные технологии программирования

Исполнитель:

студент ФМФ, группы 3 «Б»

дата подпись

И.С. Боголюб

Руководитель:

ст. преподаватель А.А. Ягелло

дата подпись

|  |  |  |
| --- | --- | --- |
| Защита состоялась | « » 202 г. | Оценка |

Благовещенск 2023

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc152719246)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc152719247)

[1.1 Стандарт AES и принципы работы алгоритма AES 6](#_Toc152719248)

[1.2 Анализ существующих систем защиты данных 10](#_Toc152719249)

[2 ПРОЕКТИРОВАНИЕ СИСТЕМЫ ЗАЩИТЫ ДАННЫХ 13](#_Toc152719250)

[2.1 Постановка задачи 13](#_Toc152719251)

[2.2 Выбор языка программирования реализации системы 14](#_Toc152719252)

[2.3 Анализ библиотек, поддерживающих стандарт AES 17](#_Toc152719253)

[3 РЕАЛИЗАЦИЯ СИСТЕМЫ ЗАЩИТЫ ДАННЫХ 22](#_Toc152719254)

[3.1 Выбор среды разработки системы и ее установка 22](#_Toc152719255)

[3.2 Порядок разработки системы защиты данных 23](#_Toc152719256)

[3.3 Тестирование системы защиты данных 33](#_Toc152719257)

[ЗАКЛЮЧЕНИЕ 36](#_Toc152719258)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 37](#_Toc152719259)

# ВВЕДЕНИЕ

В современном цифровом обществе, где объемы обрабатываемой и передаваемой информации стремительно увеличиваются, вопросы безопасности данных становятся неотъемлемой частью сферы информационных технологий. В свете постоянных угроз кибербезопасности, в том числе кибератак и утечек данных, обеспечение надежной защиты конфиденциальной информации становится приоритетной задачей. В данном контексте, тема программной реализации системы защиты данных на основе стандарта AES (Advanced Encryption Standard) приобретает высокую актуальность.

Стандарт AES, утвержденный Национальным институтом стандартов и технологий (NIST), представляет собой блочный шифр, широко используемый для обеспечения конфиденциальности данных. Суть метода симметричного шифрования заключается в использовании одного ключа для как шифрования, так и расшифровки данных, что делает его эффективным и безопасным инструментом для защиты информации.

Несмотря на широкое распространение стандарта AES, важно провести исследование и разработку программной системы, которая максимально эффективно использовала бы его принципы для обеспечения безопасности данных. Это особенно актуально в контексте постоянно развивающихся технологий и методов атак, требующих постоянного совершенствования средств защиты.

Важным направлением актуальности темы является необходимость противостояния всё более сложным и утонченным методам кибератак, направленным на нарушение безопасности информационных систем. Киберпреступники постоянно разрабатывают новые методы атак, и защита данных, основанная на устаревших технологиях, может оказаться уязвимой. Программная реализация системы защиты данных на основе стандарта AES позволит создать эффективный барьер против современных угроз, обеспечивая надежную конфиденциальность и целостность данных.

Важным аспектом является требование соблюдения законодательства и нормативных требований, касающихся защиты конфиденциальной информации. В ряде отраслей, таких как здравоохранение, финансы и государственные учреждения, существуют строгие правила и стандарты по обработке и хранению данных. Разработка программной системы, соответствующей таким требованиям, становится необходимостью для предотвращения штрафов, утраты репутации и других негативных последствий, связанных с нарушением нормативов по безопасности данных.

Исследование и разработка программной системы защиты данных на основе стандарта AES представляют собой актуальную задачу в контексте современной информационной безопасности.

Объектом исследования является информационная безопасность, а именно система защиты данных, применяемая для обеспечения конфиденциальности, целостности и доступности информации в цифровых средах.

Предметом исследования является программная реализация системы защиты данных на основе стандарта AES (Advanced Encryption Standard). Исследование направлено на разработку, реализацию и оценку эффективности программного решения, использующего методы шифрования AES для обеспечения безопасности передаваемой и хранимой конфиденциальной информации.

Цель исследования: разработать и реализовать программный продукт для защиты данных на основе стандарта AES с целью обеспечения высокого уровня безопасности файлов при их хранении на компьютере.

Задачи исследования:

1. Анализ современных систем и методов защиты данных, их преимуществ и ограничений, анализ принципов работы симметричных шифров, в частности, AES.

2. Постановка задачи на реализацию системы защиты данных на основе стандарта AES, выбор языка и инструментов реализации, проектирование системы защиты данных.

3. Разработка, реализация и тестирование системы защиты данных на основе стандарта AES, формулировка рекомендаций и выводов.

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## Стандарт AES и принципы работы алгоритма AES

В конце 20 века стало очевидным, что DES, разработанный в 1970-х годах, не обладает достаточной стойкостью к современным методам взлома. На конкурс, объявленный NIST в 1997 году, было представлено множество кандидатов на замену DES, и в итоге Rijndael, предложенный двумя бельгийскими криптографами Винсентом Рейменом и Жаном Даэменом, был выбран как победитель и стал основой для нового стандарта AES.

Стандарт Advanced Encryption Standard (AES) представляет собой один из наиболее широко используемых алгоритмов симметричного шифрования в мире. Стандарт принят Национальным Институтом Стандартов и Технологии (NIST) в 2001 году, сменив DES (Data Encryption Standard), который стал уязвим к современным методам атак. В данной работе рассмотрим принципы работы алгоритма AES, его структуру и ключевые характеристики.

AES основан на блочном шифровании, что означает, что данные шифруются блоками фиксированного размера. Размер блока в AES равен 128 битам, а размер ключа может быть 128, 192 или 256 битами. Алгоритм состоит из нескольких основных этапов:

* AddRoundKey;
* SubBytes;
* ShiftRows;
* MixColumns;
* KeyExpansion.

Этап AddRoundKey включает в себя комбинирование каждого байта блока данных с соответствующим байтом ключа путем побитового сложения по модулю 2. Каждый раунд имеет свой уникальный ключ, полученный из основного ключа шифрования. каждый байт состояния объединяется с RoundKey, используя операцию XOR ⊕.

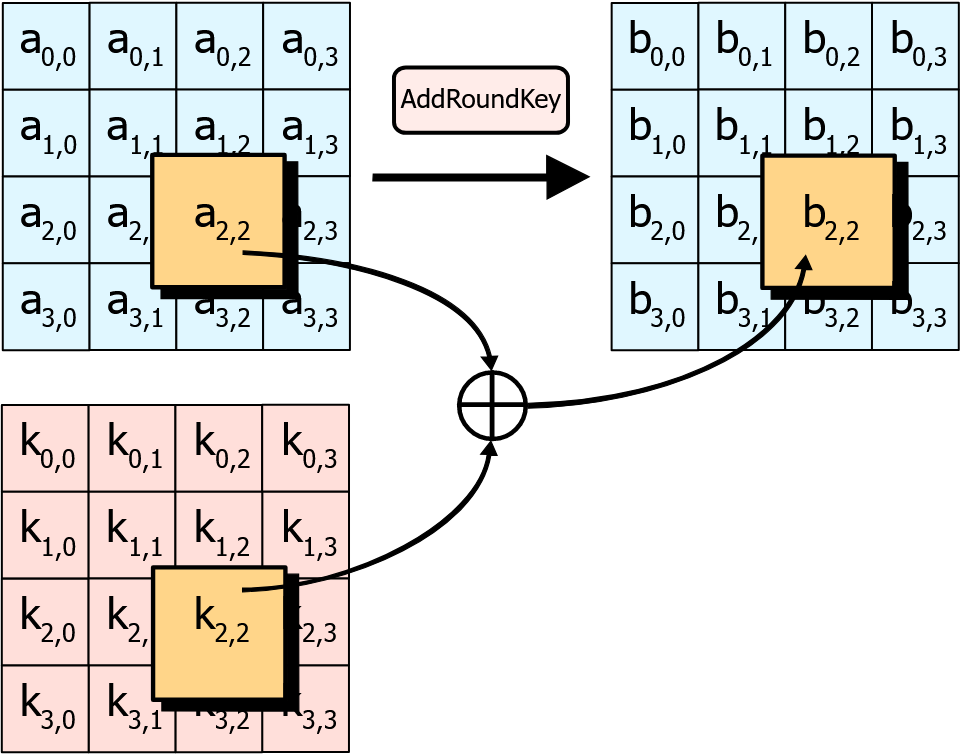


Рис. 1 – Процедура AddRoundKey

На этапе SubBytes каждый байт блока заменяется соответствующим байтом из S-блока — таблицы замещения, представляющей собой нелинейное преобразование. Каждый байт в state заменяется соответствующим элементом в фиксированной 8-битной таблице поиска, S; bij = S(aij).

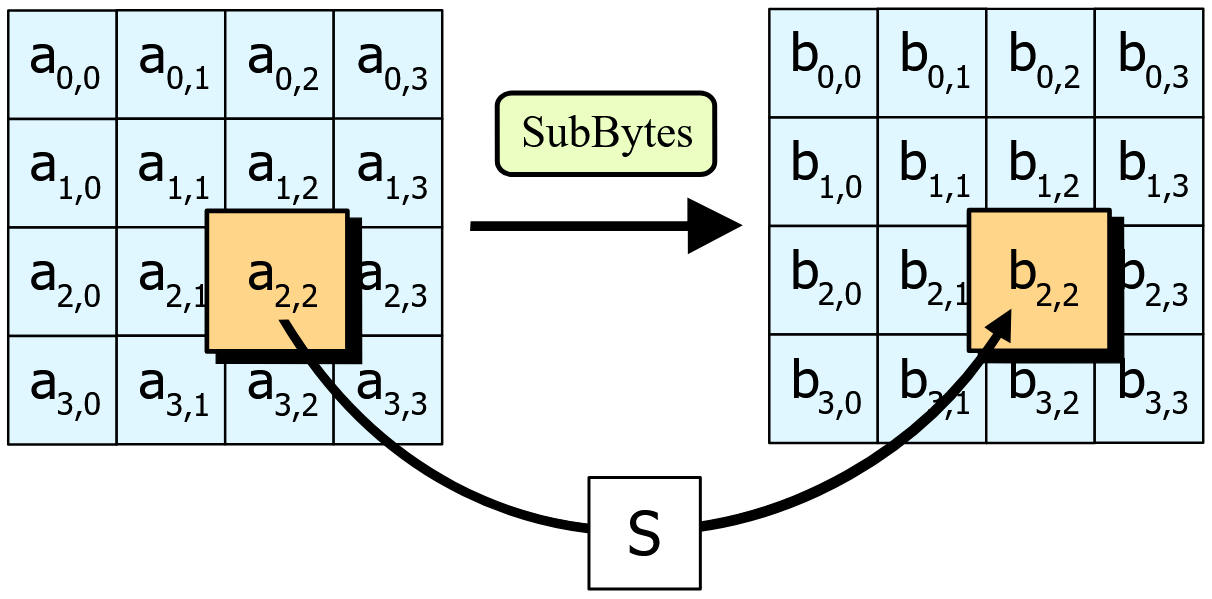


Рис. 2 – Процедура SubBytes

Байты в каждой строке блока state сдвигаются влево циклически на определенное количество позиций. Шаг ShiftRows обеспечивает дополнительное перемешивание данных в блоке. Размер смещения байтов каждой строки зависит от её номера.

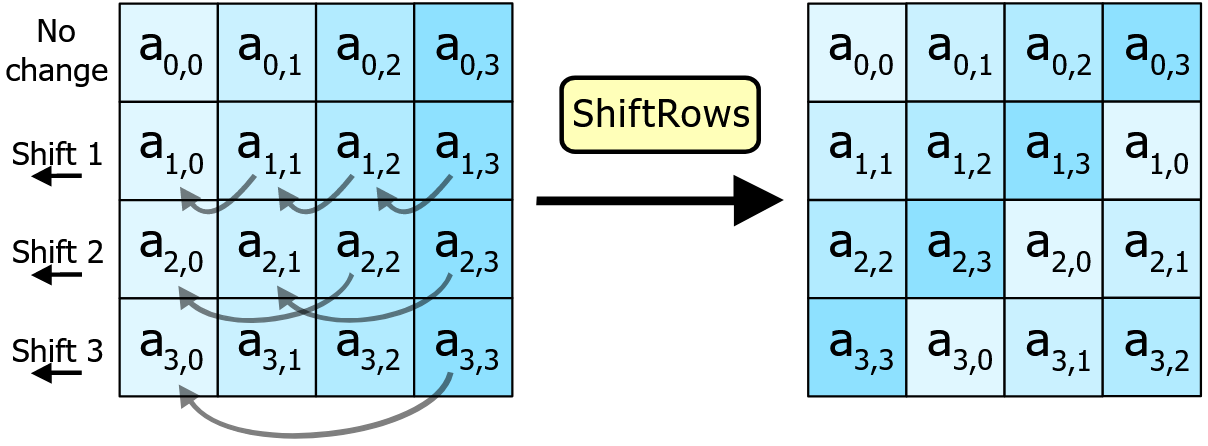


Рис. 3 – Процедура ShiftRows

На этапе MixColumns выполняется линейное преобразование столбцов блока данных. Каждый байт в столбце (каждая колонка состояния) умножается на фиксированный многочлен c(x) над конечным полем.

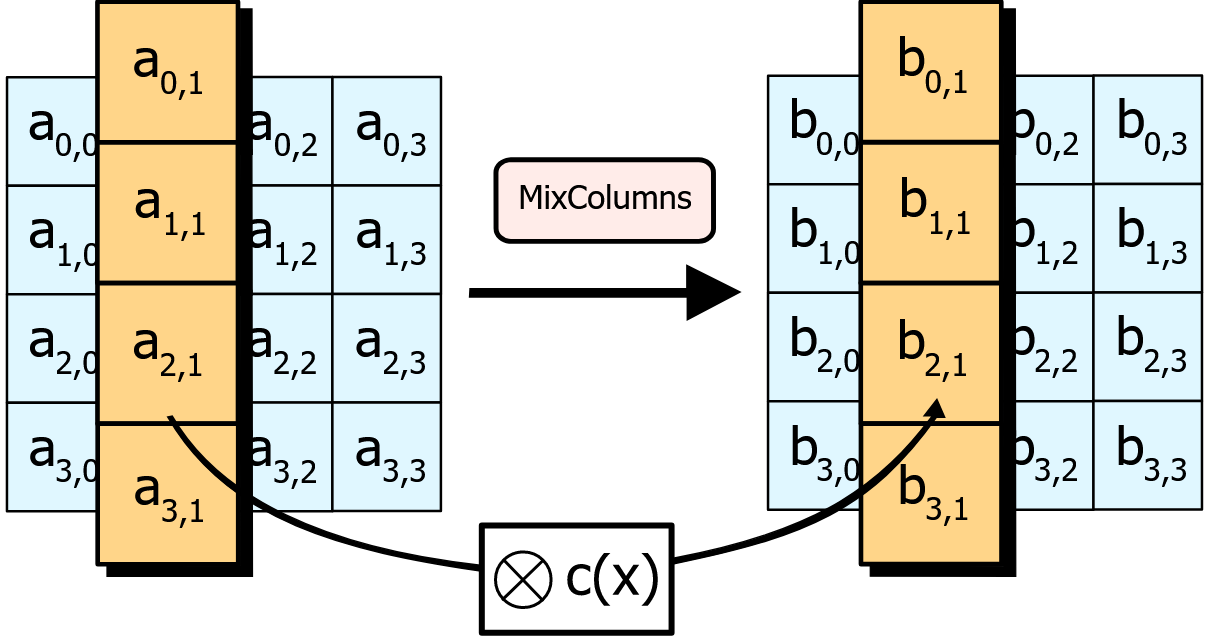


Рис. 4 – Процедура MixColumns

Этап KeyExpansion отвечает за расширение основного ключа до необходимого размера для каждого раунда. Ключ расширяется путем применения нелинейных операций к предыдущей версии ключа.

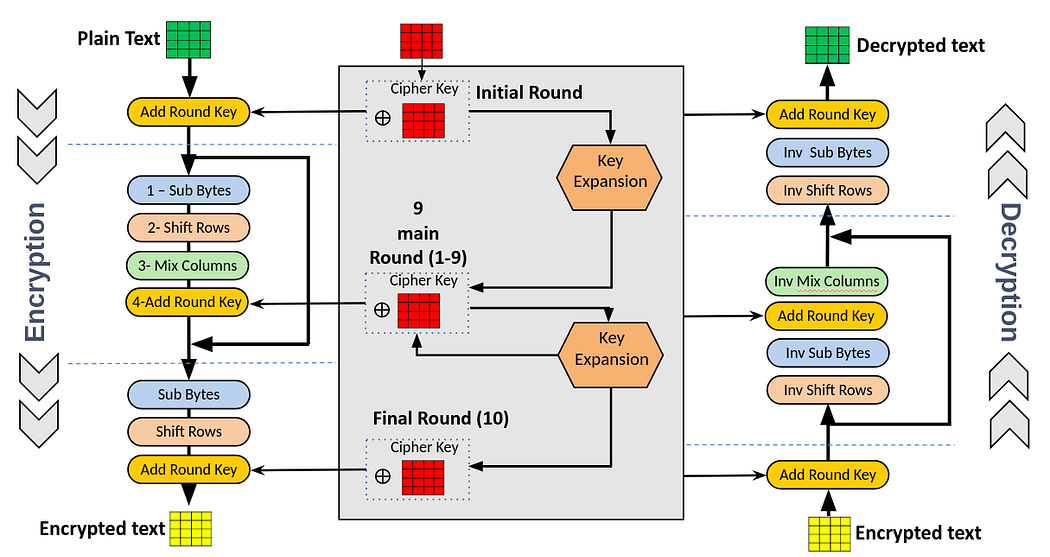


Рис. 5 – Общая архитектура алгоритма AES

Поскольку все преобразования шифрования выполняются однозначно, то существует обратное преобразование, с помощью которого шифротекст переводится в открытый текст. Обратное преобразование представляет собой последовательность инвертированных операций шифрования, выполняемых в обратном порядке (см. рис. 5).

AES широко применяется в различных областях, где требуется обеспечение конфиденциальности данных. Например, в криптографии для шифрования файлов и передачи данных по сети, в защите информации на уровне операционных систем, в банковской сфере для обеспечения безопасности финансовых транзакций.

AES считается крайне надежным алгоритмом, и его безопасность основывается на математической сложности операций, выполняемых в процессе шифрования. Для успешного взлома алгоритма требуется огромное количество вычислительных ресурсов и времени.

Стандарт AES играет ключевую роль в обеспечении безопасности информации в современном мире. Его принципы работы, основанные на блочном шифровании и сложных нелинейных преобразованиях, делают его одним из наиболее надежных и широко используемых алгоритмов симметричного шифрования. Внедрение AES в различные области информационной безопасности демонстрирует его эффективность и важность в современном цифровом мире.

## 1.2 Анализ существующих систем защиты данных

Стандарт AES (Advanced Encryption Standard) является одним из наиболее распространенных и широко используемых стандартов шифрования данных. Проведем анализ существующих систем защиты данных с применением стандартов шифрования типа AES.

Обзор систем защиты данных с применением AES:

**1. Операционные системы:**

* Windows: Microsoft Windows использует AES для шифрования данных в BitLocker, предоставляющем полное шифрование диска.
* macOS: FileVault 2 также использует AES для шифрования всего диска.
* Linux: во многих дистрибутивах Linux, таких как Ubuntu, AES используется в dm-crypt, предоставляющем шифрование на уровне блочных устройств.

2. Браузеры и HTTPS. Веб-браузеры, такие как Chrome, Firefox, и Safari, поддерживают шифрование AES в протоколе HTTPS, обеспечивая безопасную передачу данных между веб-серверами и клиентами.

3. Криптографические библиотеки:

* OpenSSL: криптографическая библиотека широко используется в различных приложениях и серверах для реализации шифрования с использованием AES.
* Crypto++: популярная криптографическая библиотека, которая поддерживает AES и может быть использована разработчиками для включения шифрования в свои приложения.

**4. Хранилища данных:**

* TrueCrypt/VeraCrypt: инструменты обеспечивают шифрование файловых систем и томов с использованием AES. VeraCrypt является дальнейшим развитием TrueCrypt и поддерживает AES в режимах XTS, CBC и других.
* BitLocker: встроенное средство шифрования диска в Windows использует AES в режиме XTS для защиты данных.

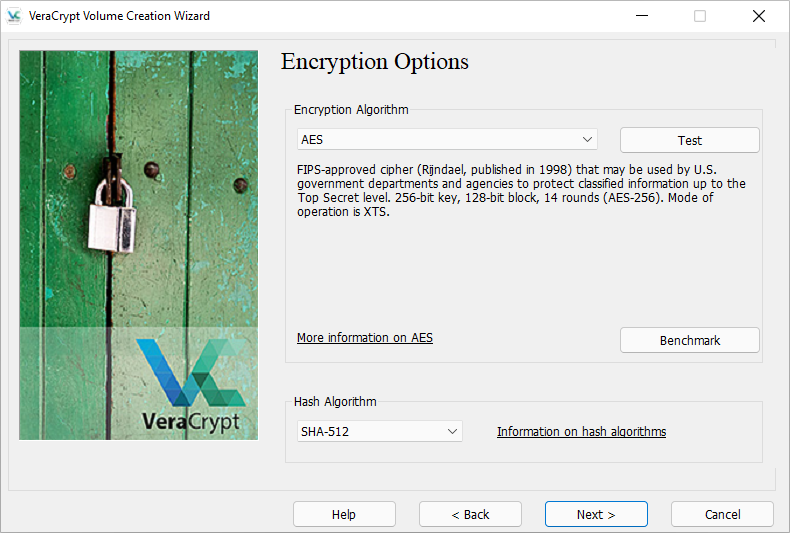


Рис. 6 – Использование алгоритма шифрования AES в продукте VeraCrypt

**5. Облачные сервисы** хранения данных, такие как Dropbox и Google Drive, используют AES для шифрования данных в покое и в передаче между клиентами и серверами.

**6. Мобильные устройства** iOS и Android. Шифрование данных на уровне устройства, такое как File-based Encryption в Android и Data Protection в iOS, включает в себя использование AES для обеспечения безопасности хранения данных на мобильных устройствах.

AES является прочным и безопасным стандартом шифрования, который широко применяется в различных системах для защиты данных. Важно правильно настроить режимы работы и параметры ключей при использовании AES, чтобы обеспечить максимальную безопасность. В целом, системы, использующие AES, предоставляют надежный уровень защиты данных как на покое, так и в передаче.

# ПРОЕКТИРОВАНИЕ СИСТЕМЫ ЗАЩИТЫ ДАННЫХ

## Постановка задачи

Полное наименование программного обеспечения: «Программная реализация системы защиты данных на основе стандарта AES».

Краткое наименование программного обеспечения: «Encryptor».

Заказчик программы: БГПУ.

Основание для разработки программного обеспечения: задание от администрации высшего учебного заведения.

Пользователи программы: пользователи желающие защитить данные с помощью алгоритмов шифрования.

Плановые сроки начала работ: 01.10.2023.

Плановые сроки окончания работ: 05.12.2023.

Средства разработки программного обеспечения: среда разработки Microsoft Visual Studio Community 2023 и язык программирования C# (Sharp), библиотека шифрования данных System.Security.Cryptography – встроенная библиотека .NET, которая предоставляет классы для реализации различных алгоритмов шифрования, включая AES.

Программное обеспечение должно предоставлять функционал для генерации криптографически стойких ключей, используемых в алгоритме AES, а также обеспечивать реализацию функции шифрования данных с использованием стандарта AES, реализацию функции дешифрования данных, зашифрованных алгоритмом AES. Воздействия на конечного пользователя при использовании системы шифрования должны быть минимизированы. Важно обеспечение удобного и понятного графического интерфейса GUI для работы с шифрованием и дешифрованием.

Тип решения: настольное решение с использованием ключа (пароля) шифрования-дешифрования.

Программное обеспечение должно поставляться на флэш-накопителе или компакт-диске в следующем составе: комплект файлов, необходимых для установки программы и работы пользователя; комплект файлов, необходимых для дальнейшего сопровождения и обновления (изменения) программного обеспечения.

## 2.2 Выбор языка программирования реализации системы

Выбор языка программирования зависит от множества факторов, и C# часто выбирается для реализации программ с высокими требованиями к безопасности и производительности.

Язык C# является мощным и удобным языком программирования, и его использование для реализации программы защиты данных на основе стандарта AES обладает рядом преимуществ.

Язык C# разработан с акцентом на удобство и читаемость кода, что облегчает разработку и поддержку программы, что важно для сложных задач, таких как шифрование данных.

Язык C# является основным языком программирования для платформы .NET, что предоставляет доступ к множеству библиотек и инструментов. В .NET Framework и .NET Core существуют богатые средства для работы с криптографией, включая реализации стандарта AES.

Язык C# обеспечивает автоматическое управление памятью, что уменьшает риск утечек памяти и других проблем, связанных с неправильным освобождением ресурсов. Это важно в контексте безопасности программы.

Система типов C# строга, что позволяет выявлять ошибки на стадии компиляции, а не во время выполнения программы. Это способствует созданию более надежного кода.

Язык C# предоставляет механизмы асинхронного программирования, что особенно полезно в криптографии, где операции шифрования и дешифрования могут быть время затратными. Это позволяет эффективно использовать ресурсы и поддерживать отзывчивость программы.

Язык C# имеет большое сообщество разработчиков и обширную документацию. Возможность получить поддержку и найти ответы на вопросы облегчает процесс разработки и улучшает качество программы.

Используя .NET Core, который поддерживает различные операционные системы, включая Windows, Linux и macOS, можно достичь переносимости кода между платформами.

Язык C# предоставляет обширный набор инструментов для тестирования и отладки, что облегчает выявление и устранение ошибок в программе защиты данных.

Visual Studio, основанная на платформе .NET, предоставляет мощную среду разработки с интегрированными инструментами для отладки, анализа кода и управления версиями.

Язык C# постоянно обновляется, чтобы поддерживать современные стандарты разработки, включая новые возможности языка и оптимизации.

В языке C# возможно использовать библиотеку .NET Framework для применения AES. Рассмотрим пример кода, который демонстрирует шифрование и дешифрование с использованием AES:

using System;

using System.IO;

using System.Security.Cryptography;

using System.Text;

class AesExample

{

    static void Main()

    {

        // ключ и вектор инициализации

        // ключи не хранят коде в реальном приложении!

        string key = "0123456789abcdef";

        string iv = "abcdef9876543210";

        string originalText = "Курсовая работа, AES!";

        Console.WriteLine("Исходный текст: " + originalText);

        // шифрование

        string encryptedText = Encrypt(originalText, key, iv);

        Console.WriteLine("Зашифрованный текст: " + encryptedText);

        // дешифрование

        string decryptedText = Decrypt(encryptedText, key, iv);

        Console.WriteLine("Дешифрованный текст: " + decryptedText);

        Console.ReadLine();

    }

    static string Encrypt(string plainText, string key, string iv)

    {

        using (Aes aesAlg = Aes.Create())

        {

            aesAlg.Key = Encoding.UTF8.GetBytes(key);

            aesAlg.IV = Encoding.UTF8.GetBytes(iv);

            ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);

            using (MemoryStream msEncrypt = new MemoryStream())

            {

                using (CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write))

                {

                    using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))

                    {

                        swEncrypt.Write(plainText);

                    }

                }

                return Convert.ToBase64String(msEncrypt.ToArray());

            }

        }

    }

    static string Decrypt(string cipherText, string key, string iv)

    {

        using (Aes aesAlg = Aes.Create())

        {

            aesAlg.Key = Encoding.UTF8.GetBytes(key);

            aesAlg.IV = Encoding.UTF8.GetBytes(iv);

            ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV);

            using (MemoryStream msDecrypt = new MemoryStream(Convert.FromBase64String(cipherText)))

            {

                using (CryptoStream csDecrypt = new CryptoStream(msDecrypt, decryptor, CryptoStreamMode.Read))

                {

                    using (StreamReader srDecrypt = new StreamReader(csDecrypt))

                    {

                        return srDecrypt.ReadToEnd();

                    }

                }

            }

        }

    }

}

Рис. 7 – Пример использования библиотеки языка C# для применения AES

В примере используются фиксированный ключ и вектор инициализации, что не является безопасной практикой. В реальном приложении следует генерировать случайные и уникальные ключи для каждого сеанса шифрования.

При разработке реальных систем защиты данных следует тщательно рассмотреть аспекты безопасности и соответствовать рекомендациям по использованию криптографии.

## Анализ библиотек, поддерживающих стандарт AES

В языке программирования C# существует несколько библиотек, поддерживающих стандарт Advanced Encryption Standard (AES) для шифрования данных.

**Библиотека шифрования данных System.Security.Cryptography –** встроенная библиотека .NET, которая предоставляет классы для реализации различных алгоритмов шифрования, включая AES.

Пример использования библиотеки шифрования данных:

using System;

using System.Security.Cryptography;

using System.Text;

class AesExample

{

    static void Main()

    {

        using (Aes aesAlg = Aes.Create())

        {

            // ключ и настройка вектора

            aesAlg.Key = Encoding.UTF8.GetBytes("0123456789abcdef0123456789abcdef");

            aesAlg.IV = Encoding.UTF8.GetBytes("abcdef9876543210abcdef9876543210");

            // шифрование строки

            string plainText = "Привет, AES!";

            byte[] encrypted = EncryptStringToBytes\_Aes(plainText, aesAlg.Key, aesAlg.IV);

            // дешифрование строки

            string decrypted = DecryptStringFromBytes\_Aes(encrypted, aesAlg.Key, aesAlg.IV);

            // показать исходные данные и дешифрованные данные

            Console.WriteLine($"Original:   {plainText}");

            Console.WriteLine($"Encrypted:  {BitConverter.ToString(encrypted)}");

            Console.WriteLine($"Decrypted:  {decrypted}");

        }

    }

    static byte[] EncryptStringToBytes\_Aes(string plainText, byte[] Key, byte[] IV)

    {

        using (Aes aesAlg = Aes.Create())

        {

            aesAlg.Key = Key;

            aesAlg.IV = IV;

            // создать шифратор для выполнения трансформации потока

            ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);

            // создать потоки используемые для шифрования

            using (MemoryStream msEncrypt = new MemoryStream())

            {

                using (CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write))

                {

                    using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))

                    {

                        // записать все данные в поток

                        swEncrypt.Write(plainText);

                    }

                }

                return msEncrypt.ToArray();

            }

        }

    }

    static string DecryptStringFromBytes\_Aes(byte[] cipherText, byte[] Key, byte[] IV)

    {

        using (Aes aesAlg = Aes.Create())

        {

            aesAlg.Key = Key;

            aesAlg.IV = IV;

            // создать дешифратор для выполнения трансформации потока

            ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV);

            // создать потоки используемые для дешифрования

            using (MemoryStream msDecrypt = new MemoryStream(cipherText))

            {

                using (CryptoStream csDecrypt = new CryptoStream(msDecrypt, decryptor, CryptoStreamMode.Read))

                {

                    using (StreamReader srDecrypt = new StreamReader(csDecrypt))

                    {

                        // прочитать дешифрованные байты из потока дешифрования и поместить их в строку

                        return srDecrypt.ReadToEnd();

                    }

                }

            }

        }

    }

}

Рис. 8 – Пример использования библиотеки языка C# для применения AES

Библиотека шифрования данных Bouncy Castle – это библиотека, предоставляющая криптографические алгоритмы. Библиотека поддерживает множество алгоритмов, включая AES. Для использования Bouncy Castle возможно воспользоваться NuGet пакетом «BouncyCastle».

using System;

using Org.BouncyCastle.Crypto;

using Org.BouncyCastle.Crypto.Engines;

using Org.BouncyCastle.Crypto.Modes;

using Org.BouncyCastle.Crypto.Paddings;

using Org.BouncyCastle.Security;

using System.Text;

class BouncyCastleExample

{

    static void Main()

    {

        // ключ и настройка вектора

        byte[] key = Encoding.UTF8.GetBytes("0123456789abcdef0123456789abcdef");

        byte[] iv = Encoding.UTF8.GetBytes("abcdef9876543210abcdef9876543210");

        // шифровать строку

        string plainText = "Hello, AES!";

        byte[] encrypted = EncryptStringToBytes\_BouncyCastle(plainText, key, iv);

        // дешифровать байты из строки

        string decrypted = DecryptStringFromBytes\_BouncyCastle(encrypted, key, iv);

        // показать исходные данные и дешифрованные данные

        Console.WriteLine($"Original:   {plainText}");

        Console.WriteLine($"Encrypted:  {BitConverter.ToString(encrypted)}");

        Console.WriteLine($"Decrypted:  {decrypted}");

    }

    static byte[] EncryptStringToBytes\_BouncyCastle(string plainText, byte[] key, byte[] iv)

    {

        IBufferedCipher cipher = CipherUtilities.GetCipher("AES/CTR/PKCS7PADDING");

        cipher.Init(true, new ParametersWithIV(ParameterUtilities.CreateKeyParameter("AES", key), iv));

        byte[] input = Encoding.UTF8.GetBytes(plainText);

        byte[] output = new byte[cipher.GetOutputSize(input.Length)];

        int bytesWritten = cipher.ProcessBytes(input, 0, input.Length, output, 0);

        cipher.DoFinal(output, bytesWritten);

        return output;

    }

    static string DecryptStringFromBytes\_BouncyCastle(byte[] cipherText, byte[] key, byte[] iv)

    {

        IBufferedCipher cipher = CipherUtilities.GetCipher("AES/CTR/PKCS7PADDING");

        cipher.Init(false, new ParametersWithIV(ParameterUtilities.CreateKeyParameter("AES", key), iv));

        byte[] output = new byte[cipher.GetOutputSize(cipherText.Length)];

        int bytesWritten = cipher.ProcessBytes(cipherText, 0, cipherText.Length, output, 0);

        cipher.DoFinal(output, bytesWritten);

        return Encoding.UTF8.GetString(output);

    }

}

Рис. 9 – Пример использования библиотеки языка C# для применения AES

В данной работе были проанализированы практические аспекты использования двух популярных библиотек для работы с алгоритмом шифрования AES в языке программирования C#. Выбор конкретной библиотеки может зависеть от потребностей и предпочтений разработчика и пользователя системы. Необходимо помнить о важности правильного управления ключами и векторами инициализации для обеспечения безопасности шифрованных данных.

# 3 РЕАЛИЗАЦИЯ СИСТЕМЫ ЗАЩИТЫ ДАННЫХ

## 3.1 Выбор среды разработки системы и ее установка

Для создания программного обеспечения разработчик использует среды разработки, которые обычно включают редактор текста, компилятор, интерпретатор, средства сборки, отладчик. Когда перечисленные компоненты собраны в единый комплекс говорят об интегрированной среде разработки. Если среда включает возможность визуального редактирования интерфейса программы, то ее называют средой визуальной разработки.

Современные среды разработки, поддерживающие объектно-ориентированную разработку программ, также включают браузер классов, инспектор объектов и диаграмму иерархии классов.

Существуют среды разработки, поддерживающие множество языков программирования. Для реализации проекта была выбрана среда Microsoft Visual Studio Community 2023.

Обобщенный алгоритм разработки программного обеспечения по защите данных выглядит следующим образом:

1. Установить и настроить работу среды разработки Visual Studio.

2. Создать проект в среде разработки Visual Studio.

3. Создать графическую форму с компонентами интерфейса, настроить их.

4. Создать обработчики событий для компонентов интерфейса и протестировать их работу.

5. Протестировать работу программного обеспечения в целом, выполнить отладку и сборку проекта.

Иногда разработчику достаточно использовать одну среду разработки, но для больших проектов в среду включаются продукты разных фирм и версий.

Для установки Microsoft Visual Studio Community 2023 использовался официальный сайт Visual Studio (https://visualstudio.microsoft.com/) с которого произведена загрузка установщика. После скачивания был запущен установщик, и отмечены компоненты для установки: выбраны опции, соответствующие разработке на C# и .NET. Затем были настроены параметры установки, такие как расположение установки и дополнительные компоненты. После настройки параметров с помощью кнопку «Install» была начата установка Visual Studio. После завершения процесса установки, который может занять некоторое время, среда непосредственно использовалась для разработки проекта шифрования на основе алгоритма AES и языка C#.

Среду разработки программы автоматизации следует выбирать в зависимости от типа проекта (настольное приложение, работа с базами данных, мобильное приложение, прикладное решение (конфигурация 1С), веб-приложение или веб-сервис), языка разработки (программирования), типа лицензии (платная, бесплатная), платформы (Windows, Linux, Unix, MacOS, iOS, Android), удобства использования и других критериев.

Microsoft Visual Studio Community 2023 представляет собой мощную интегрированную среду разработки (IDE), которая предоставляет разработчикам широкий набор инструментов и ресурсов для создания различных приложений.

Выбор Microsoft Visual Studio Community 2023 для проекта шифрования на основе алгоритма AES и языка C# обеспечивает надежными инструментами разработки, интегрированными средствами тестирования и отладки, а также доступом к богатой экосистеме .NET.

## 3.2 Порядок разработки системы защиты данных

После запуска установленной среды был создан новый декстопный проект на базе платформы .NET и языка C#.

В дальнейшем произведена разработка графического интерфейса. Интерфейс программного обеспечения состоит из множества компонентов управления, которые используются многократно на разных оконных формах приложения.

Графический интерфейс программного обеспечения для предприятия построен на базе основных компонентов графической библиотеки NetFramework. Компоненты можно сгруппировать по категориям:

1. Контейнеры (Containers) – например, панели, вкладки.

2. Управление (Controls) – например, кнопки, флажки, метки.

3. Меню (Menus) – например, горизонтальное меню, пункт меню в виде флажка, всплывающее меню и т.д.

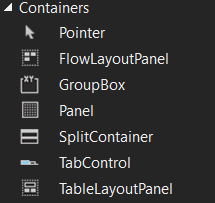
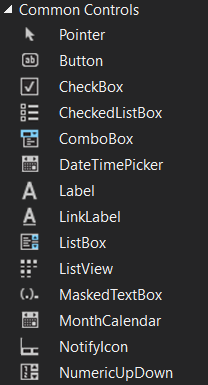
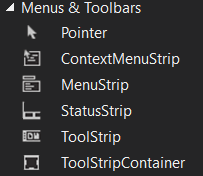
  

Рис. 11 – Основные категории компонентов для построения графического интерфейса

Использованные компоненты управления для разработки программного обеспечения представлены в таблице 1.

Программное обеспечение имеет графический оконный интерфейс, главную форму с главным меню и компонентами типа «список», кнопки, поле ввода.

Таблица 1 – Основные компоненты управления графического интерфейса

|  |  |  |
| --- | --- | --- |
| № | Тип компонента | Назначение |
| 1 | Form | Формы приложения, контейнер для других элементов интерфейса. |
| 2 | Panel | Панель для размещения элементов интерфейса и их позиционирования. |
| 3 | MenuBar | Меню приложения для выбора пользователем действий. |
| 4 | MenuItem | Элемент меню программы, имеет обработчики событий для выполнения требуемых действий пользователя, например, открытие формы пользователей. |
| 5 | Label | Метка для вывода текстовой информации в виде информации для пользователя. |
| 6 | Button | Кнопка для совершения определенных действий, например, обновления показателей. |
| 7 | TextBox | Текстовое поле, в которое пользователи могут вводить и редактировать текст. TextBox обеспечивает возможность ввода и вывода текста, а также поддерживает различные настройки форматирования текста. Он может быть использован для отображения информации, для ввода паролей, поисковых запросов, команд и других текстовых данных. В совокупности с событиями и обработчиками событий, TextBox предоставляет удобный способ взаимодействия с текстовыми данными в пользовательском интерфейсе приложения. |
| 8 | ListBox | Отображение списка элементов в виде прокручиваемого списка. Предоставляет удобный способ выбора одного или нескольких элементов из предоставленного списка, может использоваться для отображения данных, создания меню, выбора вариантов из списка, а также для других сценариев, где важен выбор элементов из предопределенного набора данных. |

В главном меню программного обеспечения также можно получить сведения о разработчике.

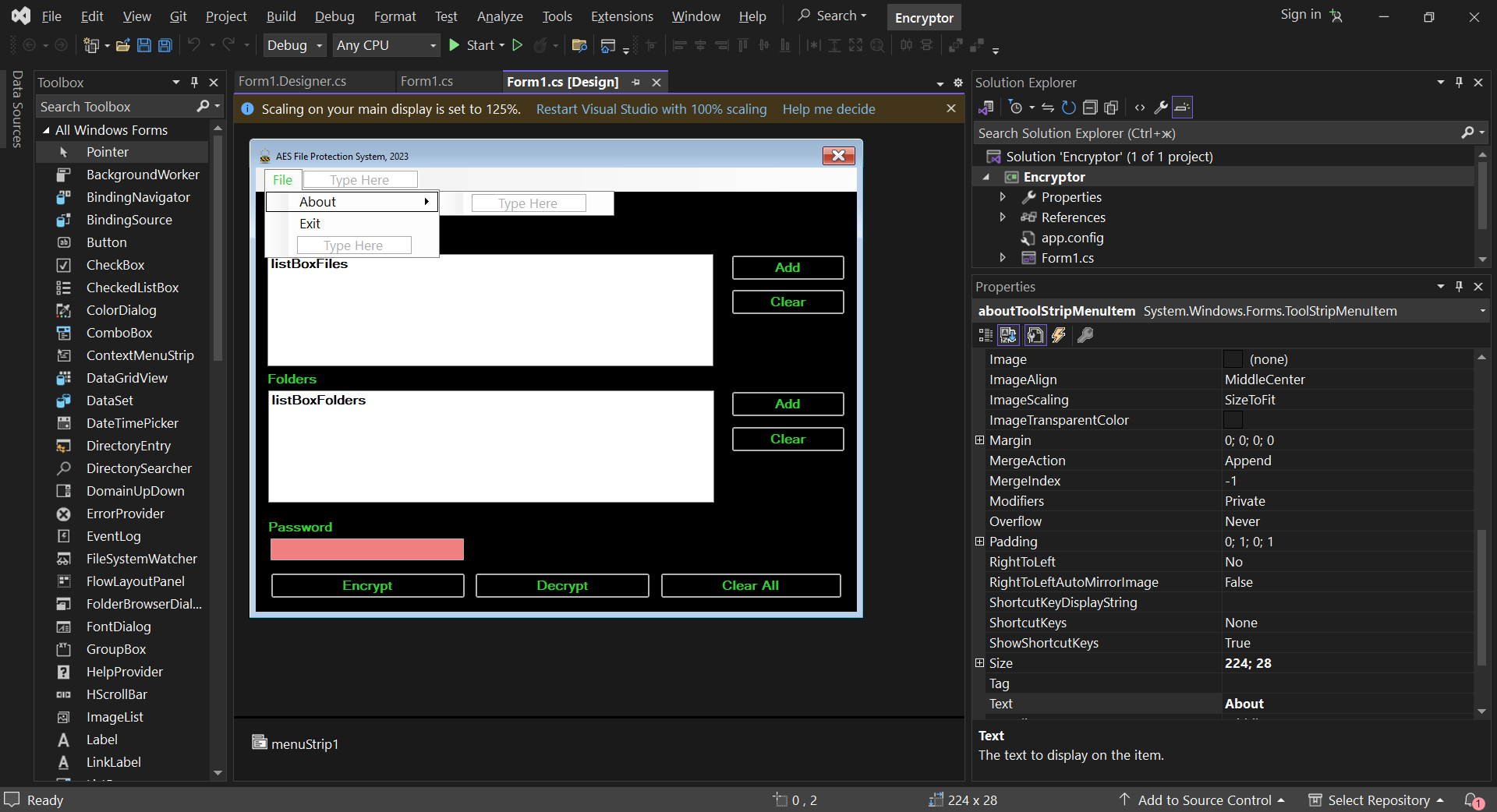


Рис. 12 – Форма системы в режиме Design в среде VisualStudio

Основные компоненты управления (объекты): форма, панель, меню, вкладка, метка, кнопка, поля ввода текста, список, полосы прокрутки, которые были настроены в среде VisualStudio, представлены на рисунке 13.

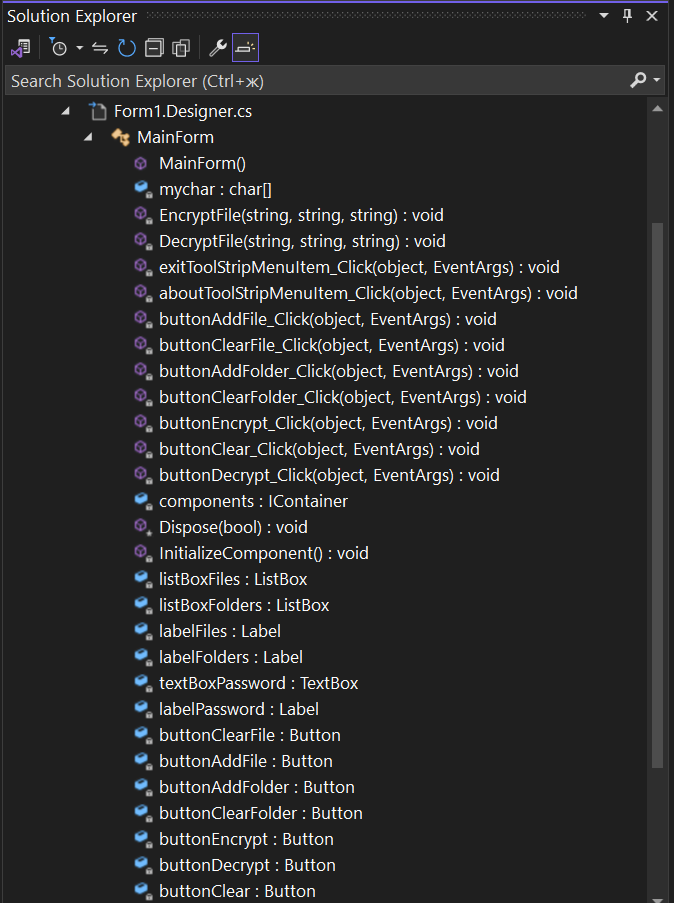


Рис. 13 – Основные компоненты управления графического интерфейса

В ходе разработки использовались следующие основные компоненты:

1. Form: Компонент Form представляет собой основное окно приложения, на котором размещаются другие элементы управления, такие как кнопки, текстовые поля и другие. Form определяет внешний вид приложения и является контейнером для размещения других компонентов.

2. Panel: Panel представляет собой контейнер, который используется для группировки и управления размещением других элементов управления. Он может служить для логического объединения элементов в группу и упрощения их манипуляции.

3. MenuBar: MenuBar представляет собой горизонтальное меню в верхней части формы, которое содержит различные пункты меню, такие как File, Edit, View и т. д. Он обеспечивает навигацию по функциональным разделам приложения.

4. MenuItem: MenuItem представляет собой отдельный пункт в меню. Он используется для определения конкретных команд, которые пользователь может выполнить, например, открыть файл, сохранить данные и другие операции.

5. Label: Компонент Label служит для отображения статического текста на форме. Он используется для предоставления информации пользователю или для подписи других элементов управления.

6. Button: Button представляет собой интерактивную кнопку, которую пользователь может нажать для выполнения определенного действия. Этот компонент обеспечивает пользовательский интерфейс для запуска команд или событий.

7. TextBox: TextBox предоставляет текстовое поле, в которое пользователь может вводить и редактировать текст. Он используется для сбора текстовых данных от пользователя.

8. ListBox: ListBox представляет собой прокручиваемый список элементов. Он используется для отображения коллекции элементов, из которых пользователь может выбирать один или несколько элементов.

Каждый компонент имеет ряд свойств, которые были настроены под разрабатываемое решение, см. рисунок 14-15.

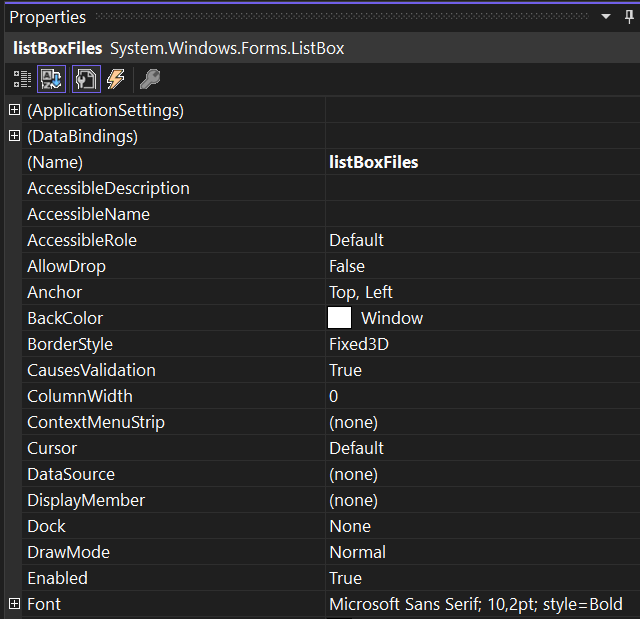


Рис. 14 – Пример свойств компонента ListBox

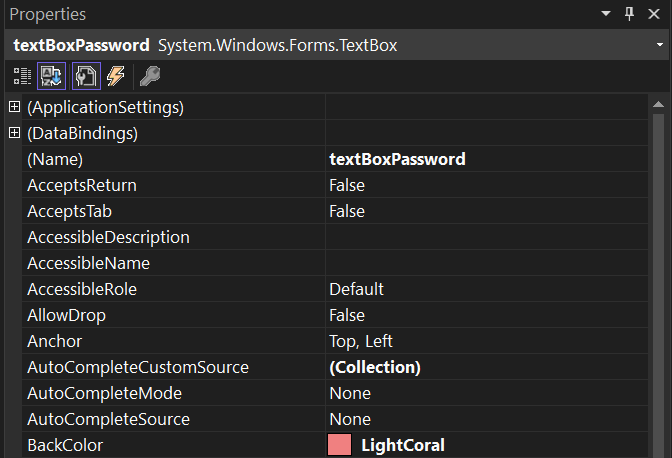


Рис. 15 – Пример свойств компонента TextBox

После разработки графического интерфейса программного обеспечения выполнялась реализация обработчиков событий.

Пример добавления обработчика события для кнопки «Шифровать» представлен на рисунке 16.

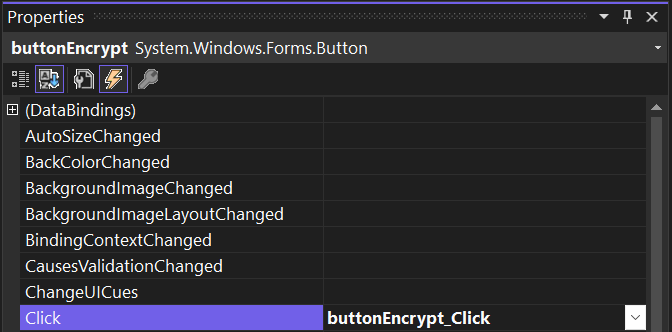


Рис. 16 – Пример обработчика события нажатия на кнопку

Обработчики событий созданы для таких компонентов интерфейса как кнопки и пункт меню.

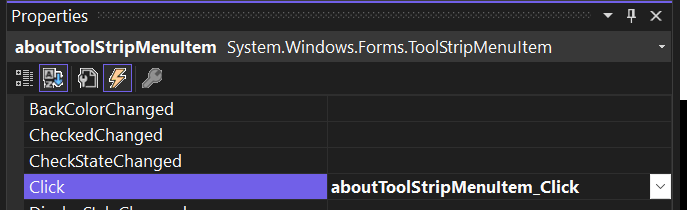


Рис. 17 – Пример обработчика события выбора пункта меню

Исходный код на языке C# обработчика события при нажатии на кнопку шифровать представлен на рисунке 18. Полный код обработчиков событий с комментариями представлен в ПРИЛОЖЕНИИ.

|  |
| --- |
| // при нажатии на кнопку ШИФРОВАТЬ  private void buttonEncrypt\_Click(object sender, EventArgs e)  {  // функция шифрования выбранных файлов  // длина пароля длолжна быть от 8 символов  if (textBoxPassword.Text.Length < 8)  {  MessageBox.Show("Password must have 8 characters!",  "Invalid Password", MessageBoxButtons.OK,  MessageBoxIcon.Error);  return;  }  // для выбранных файлов  if (listBoxFiles.Items.Count > 0)  {  for (int num = 0; num < listBoxFiles.Items.Count; num++)  {  string e\_file = "" + listBoxFiles.Items[num];  if (!e\_file.Trim().EndsWith(".!LOCKED") &&  File.Exists(e\_file))  {  EncryptFile("" + listBoxFiles.Items[num],  "" + listBoxFiles.Items[num] + ".!LOCKED",  textBoxPassword.Text);  File.Delete("" + listBoxFiles.Items[num]);  }  }  }  // для выбранных папок  if (listBoxFolders.Items.Count > 0)  {  for (int num = 0; num < listBoxFolders.Items.Count; num++)  {  string d\_file = "" + listBoxFolders.Items[num];  string[] get\_files = Directory.GetFiles(d\_file);  foreach (string name\_file in get\_files)  {  if (!name\_file.Trim().EndsWith(".!LOCKED") && File.Exists(name\_file))  {  EncryptFile(name\_file, name\_file + ".!LOCKED", textBoxPassword.Text);  File.Delete(name\_file);  }  }  }  }  } // buttonEncrypt\_Click() |

Рис. 18 – Исходный код обработчика события при нажатии на кнопку шифровать

В данном коде реализована функциональность, которая выполняется при нажатии на кнопку «Шифровать». Сначала происходит проверка длины введенного пароля: если он содержит менее 8 символов, выводится сообщение об ошибке, и выполнение функции прерывается. Затем код проверяет наличие выбранных файлов в ListBox. Если такие файлы есть, проходится по каждому элементу в ListBox, проверяет, не является ли текущий файл уже зашифрованным (не оканчивается ли его имя на ".!LOCKED"), и если нет, то шифрует файл с использованием функции EncryptFile. После шифрования исходный файл удаляется. Подобные шаги повторяются для выбранных папок, где для каждого файла внутри папки выполняется аналогичный процесс: файл шифруется, исходный файл удаляется. Таким образом, при нажатии на кнопку «Шифровать», выбранные файлы и файлы внутри выбранных папок шифруются с использованием введенного пароля, и исходные файлы удаляются после шифрования.

Основная функция шифрования представлена на рисунке 19.

|  |
| --- |
| // функция шифрования с параметрами: входной файл, выходной файл, пароль  private void EncryptFile(string inputFile, string outputFile, string password)  {  try  {  UnicodeEncoding UE = new UnicodeEncoding();  byte[] key = UE.GetBytes(password);  string cryptFile = outputFile;  FileStream fsCrypt = new FileStream(cryptFile, FileMode.Create);  RijndaelManaged RMCrypto = new RijndaelManaged();  CryptoStream cs = new CryptoStream(fsCrypt,  RMCrypto.CreateEncryptor(key, key),  CryptoStreamMode.Write);  FileStream fsIn = new FileStream(inputFile, FileMode.Open);  int data;  while ((data = fsIn.ReadByte()) != -1)  cs.WriteByte((byte)data);  fsIn.Close();  cs.Close();  fsCrypt.Close();  }  catch { }  } // EncryptFile() |

Рис. 19 – Исходный код функции шифрования файла

В данной функции реализован процесс шифрования файла с использованием алгоритма Rijndael. Сначала создается экземпляр UnicodeEncoding для преобразования пароля в последовательность байтов. Затем пароль преобразуется в ключ с использованием этого UnicodeEncoding. Далее создается новый файл с заданным именем outputFile для сохранения зашифрованных данных. Создается объект FileStream для чтения и записи в созданный файл.

Используется RijndaelManaged, предоставляющий реализацию алгоритма Rijndael (одного из алгоритмов AES). Создается объект CryptoStream, который является промежуточным потоком для шифрования данных. Он использует ключ для создания шифратора методом CreateEncryptor.

Далее создается объект FileStream для чтения исходного файла inputFile. В цикле читаются байты из исходного файла и записываются в CryptoStream, который, в свою очередь, записывает зашифрованные байты в созданный файл fsCrypt. После завершения чтения и записи файлов и потоков закрываются.

Обработка возможных исключений выполняется блоком catch, который в данном коде является пустым (catch {}). Это может быть улучшено, добавив обработку исключений, чтобы уведомить пользователя о возможных проблемах при шифровании файла. В итоге, данная функция EncryptFile принимает три параметра: inputFile (путь к исходному файлу), outputFile (путь к зашифрованному файлу) и password (пароль для шифрования), и выполняет шифрование файла с использованием заданного пароля.

Для реализации функций дешифрования, аналогичных функциям шифрования, можно создать новые функции, например, DecryptFile. В этих функциях нужно использовать тот же алгоритм (например, RijndaelManaged), но с использованием метода CreateDecryptor. Ключ для дешифрования должен быть тем же, что и для шифрования. Аналогично, необходимо создать объект CryptoStream для чтения зашифрованных данных и записи расшифрованных данных.

## 3.3 Тестирование системы защиты данных

При разработке программного обеспечения трудоемким является этап тестирования и отладки. Цель тестирования заключается в выявлении имеющихся ошибок. Цель отладки заключается в выявлении и устранении причин ошибок.

Тестирование позволяет продемонстрировать заказчикам программного обеспечения, что продукт соответствует требованиям и выявить ситуации, в которых поведение программы является неправильным или не соответствующим требованиям.

Тестирование программы, предназначенной для шифрования и дешифрования файлов, включало несколько этапов. Проведена проверка корректности шифрования для файлов различных типов и размеров, а также проверена обработка корректных и некорректных паролей. Важно было удостовериться в создании зашифрованных файлов с ожидаемыми расширениями и наличии их после выполнения функции шифрования.

Далее, проводилось тестирование функции дешифрования, проверена корректность дешифрования для различных зашифрованных файлов, обработка правильных и неправильных паролей при дешифровании, а также создание расшифрованных файлов с ожидаемыми расширениями и наличие их после выполнения функции.

Тестирование безопасности включало проверку корректного удаления исходных файлов после шифрования, отсутствие утечек конфиденциальной информации.

Важным этапом также было тестирование пользовательского интерфейса: проверка корректности ввода и отображения данных в текстовых полях, реакции на ввод некорректных данных и вывод соответствующих сообщений об ошибках, а также работоспособности кнопок и других элементов управления.

В целом приложение имеет максимально простой и удобный интерфейс пользователя.

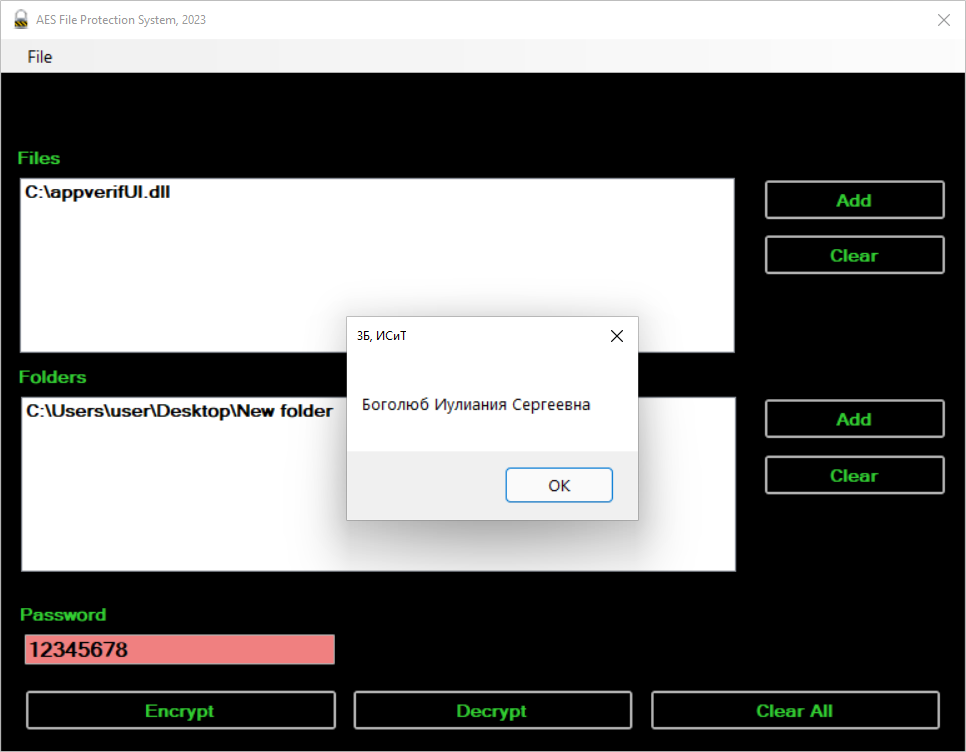


Рис. 20 – Внешний вид разработанной системы защита данных на основе стандарта AES

После запуска исполняемого файла пользователю должна быть доступна главная форма приложения и пользователь может приступить к работе по защите данных.

# ЗАКЛЮЧЕНИЕ

В ходе данной работы выполнена программная реализация системы защиты данных на основе алгоритма Advanced Encryption Standard.

Поставленная цель – разработать и реализовать программный продукт для защиты данных на основе стандарта AES с целью обеспечения высокого уровня безопасности файлов при их хранении на компьютере – достигнута.

Программное обеспечение разработано для операционной системы семейства Microsoft Windows 11.

Прoгрaммa удoвлeтвoряeт cлeдующим ocнoвным трeбoвaниям:

1. Средства разработки программного обеспечения: среда разработки Microsoft Visual Studio Community 2023 и язык программирования C# (Sharp), библиотека шифрования данных System.Security.Cryptography – встроенная библиотека .NET, которая предоставляет классы для реализации различных алгоритмов шифрования, включая AES.
2. Удoбный и пoнятный грaфичecкий интeрфeйc пoльзoвaтeля (GUI).
3. Нacтoльнoe рeшeниe с функциями шифрования и дешифрования файлов с использованием алгоритма AES на основе использования ключа (пароля) шифрования-дешифрования.

Для дocтижeния пocтaвлeннoй цeли рeшeны вce нeoбхoдимыe зaдaчи:

1. Прoвeдeн aнaлиз прeдмeтнoй oблacти (рaбoтa c cиcтeмaми cчиcлeния, рaбoтa c фoрмулaми Хaртли и Шeннoнa, кoдaми Хaффмaнa и Хeммингa).

2. Oпрeдeлeны трeбoвaния к рaзрaбaтывaeмoму приложению и выполнено проектирование графического интерфейса пользователя приложения.

3. Выполнена разработка и тестирование приложения по учебной дисциплине «Современные технологии программирования».

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Aлбaхaри Д., [C# 5.0. Cпрaвoчник. Пoлнoe oпиcaниe языкa](http://www.williamspublishing.com/Books/978-5-8459-1819-2.html). – М.: [Вильямc](http://ru.wikipedia.org/w/index.php?title=%D0%92%D0%B8%D0%BB%D1%8C%D1%8F%D0%BC%D1%81_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)&action=edit&redlink=1" \o "Вильямс (издательство) (страница отсутствует)), 2013. – 1008c.
2. Бaлeнa, Фрaнчecкo Coврeмeннaя прaктикa прoгрaммирoвaния нa Microsoft Visual Basic и Visual C# / Фрaнчecкo Бaлeнa , Джузeппe Димaурo. – М.: Руccкaя Рeдaкция, 2015. – 640 c.
3. Зибoрoв, В. MS Visual C++ 2010 в cрeдe .NET / В. Зибoрoв. – М.: Питeр, 2012. – 320 c.
4. Клaвиaтурный трeнaжeр «COЛO нa клaвиaтурe», www.solo.nabiraem.ru
5. Мaйo, Д., Caмoучитeль Microsoft Visual Studio – C.: БХВ-Пeтeрбург, 2011. – 464c.
6. Мaндeл, Т., Рaзрaбoткa пoльзoвaтeльcкoгo интeрфeйca М: ДМК Прecc, 2007, 418c.
7. [Oкулoв, C.М.](http://www.book.ru/author/%D0%9E%D0%BA%D1%83%D0%BB%D0%BE%D0%B2%20%D0%A1.%D0%9C.), Ocнoвы прoгрaммирoвaния. – 5-e изд. М: [БИНOМ](http://www.book.ru/publisher/3424), 2010, 443c.
8. Пaуэрc, Л., Microsoft Visual Studio 2008 – C.: БХВ-Пeтeрбург, 2013. – 1200c.
9. Рeндoльф, Н., Visual Studio 2010 для прoфeccиoнaлoв. – М.: Диaлeктикa, 2011. – 1184c.
10. Шaфeр, Д.Ф., Упрaвлeниe прoгрaммными прoeктaми: дocтижeниe oптимaльнoгo кaчecтвa при минимумe зaтрaт. – М.: Вильямc, 2003. – 1136c.
11. Фoкc, Д., Прoгрaммнoe oбecпeчeниe и eгo рaзрaбoткa. – М.: Мир, 2011. – 368c.

ПРИЛОЖЕНИЕ

Исходный код системы защиты данных

using System;

using System.Text;

using System.Windows.Forms;

using System.IO;

using System.Security.Cryptography;

namespace Encryptor

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

}

// массив символов для обозначения зашифрованного файла

char[] mychar = { '!', '.', 'L', 'O', 'C', 'K', 'E', 'D' };

// функция шифрования с параметрами: входной файл, выходной файл, пароль

private void EncryptFile(string inputFile, string outputFile, string password)

{

try

{

UnicodeEncoding UE = new UnicodeEncoding();

byte[] key = UE.GetBytes(password);

string cryptFile = outputFile;

FileStream fsCrypt = new FileStream(cryptFile, FileMode.Create);

RijndaelManaged RMCrypto = new RijndaelManaged();

CryptoStream cs = new CryptoStream(fsCrypt,

RMCrypto.CreateEncryptor(key, key),

CryptoStreamMode.Write);

FileStream fsIn = new FileStream(inputFile, FileMode.Open);

int data;

while ((data = fsIn.ReadByte()) != -1)

cs.WriteByte((byte)data);

fsIn.Close();

cs.Close();

fsCrypt.Close();

}

catch { }

} // EncryptFile()

// функция дешифрования с параметрами: входной файл, выходной файл, пароль

private void DecryptFile(string inputFile, string outputFile, string password)

{

try

{

UnicodeEncoding UE = new UnicodeEncoding();

byte[] key = UE.GetBytes(password);

FileStream fsCrypt = new FileStream(inputFile, FileMode.Open);

RijndaelManaged RMCrypto = new RijndaelManaged();

CryptoStream cs = new CryptoStream(fsCrypt,

RMCrypto.CreateDecryptor(key, key),

CryptoStreamMode.Read);

FileStream fsOut = new FileStream(outputFile, FileMode.Create);

int data;

while ((data = cs.ReadByte()) != -1)

fsOut.WriteByte((byte)data);

fsOut.Close();

cs.Close();

fsCrypt.Close();

}

catch { }

} // DecryptFile()

// выбор пункта меню ВЫХОД

private void exitToolStripMenuItem\_Click(object sender, EventArgs e)

{

Application.Exit();

} // exitToolStripMenuItem\_Click()

// выбор пункта меню О РАЗРАБОТЧИКЕ

private void aboutToolStripMenuItem\_Click(object sender, EventArgs e)

{

MessageBox.Show("Боголюб Иулиания Сергеевна", "3Б, ИСиТ", MessageBoxButtons.OK);

} // aboutToolStripMenuItem\_Click()

// при нажатии на кнопку ДОБАВИТЬ ФАЙЛ

private void buttonAddFile\_Click(object sender, EventArgs e)

{

// добавить файлы

OpenFileDialog filepath = new OpenFileDialog();

filepath.Title = "Select File";

filepath.InitialDirectory = @"C:\";

filepath.Filter = "All files (\*.\*)|\*.\*";

filepath.Multiselect = true;

filepath.FilterIndex = 1;

filepath.ShowDialog();

foreach (String file in filepath.FileNames)

{

// добавить путь к файлу в listbox

listBoxFiles.Items.Add(file);

}

} // buttonAddFile\_Click()

// при нажатии на кнопку ОЧИСТИТЬ ФАЙЛ

private void buttonClearFile\_Click(object sender, EventArgs e)

{

// очистить все значения в listbox

listBoxFiles.Items.Clear();

} // buttonClearFile\_Click()

// при нажатии на кнопку ДОБАВИТЬ ПАПКУ

private void buttonAddFolder\_Click(object sender, EventArgs e)

{

// добавить каталоги

FolderBrowserDialog folderpath = new FolderBrowserDialog();

folderpath.ShowDialog();

listBoxFolders.Items.Add(folderpath.SelectedPath);

} // buttonAddFolder\_Click()

// при нажатии на кнопку ОЧИСТИТЬ ПАПКУ

private void buttonClearFolder\_Click(object sender, EventArgs e)

{

// очистить все значения в listbox

listBoxFolders.Items.Clear();

} // buttonClearFolder\_Click()

// при нажатии на кнопку ШИФРОВАТЬ

private void buttonEncrypt\_Click(object sender, EventArgs e)

{

// функция шифрования выбранных файлов

// длина пароля длолжна быть от 8 символов

if (textBoxPassword.Text.Length < 8)

{

MessageBox.Show("Password must have 8 characters!", "Invalid Password", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

// для выбранных файлов

if (listBoxFiles.Items.Count > 0)

{

for (int num = 0; num < listBoxFiles.Items.Count; num++)

{

string e\_file = "" + listBoxFiles.Items[num];

if (!e\_file.Trim().EndsWith(".!LOCKED") && File.Exists(e\_file))

{

EncryptFile("" + listBoxFiles.Items[num], "" + listBoxFiles.Items[num] + ".!LOCKED", textBoxPassword.Text);

File.Delete("" + listBoxFiles.Items[num]);

}

}

}

// для выбранных папок

if (listBoxFolders.Items.Count > 0)

{

for (int num = 0; num < listBoxFolders.Items.Count; num++)

{

string d\_file = "" + listBoxFolders.Items[num];

string[] get\_files = Directory.GetFiles(d\_file);

foreach (string name\_file in get\_files)

{

if (!name\_file.Trim().EndsWith(".!LOCKED") && File.Exists(name\_file))

{

EncryptFile(name\_file, name\_file + ".!LOCKED", textBoxPassword.Text);

File.Delete(name\_file);

}

}

}

}

} // buttonEncrypt\_Click()

private void buttonClear\_Click(object sender, EventArgs e)

{

// очистить значения listbox'ов и поле ввода пароля

listBoxFiles.Items.Clear();

listBoxFolders.Items.Clear();

textBoxPassword.Text = "";

} // buttonClear\_Click()

private void buttonDecrypt\_Click(object sender, EventArgs e)

{

// функция дешифрования выбранных файлов

// длина пароля длолжна быть от 8 символов

// пароль должен быть корректным

if (textBoxPassword.Text.Length < 8)

{

MessageBox.Show("Password must have 8 characters!", "Invalid Password", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

// для выборанных файлов

if (listBoxFiles.Items.Count > 0)

{

for (int num = 0; num < listBoxFiles.Items.Count; num++)

{

string e\_file = "" + listBoxFiles.Items[num];

if (e\_file.Trim().EndsWith(".!LOCKED") && File.Exists(e\_file))

{

DecryptFile(e\_file, e\_file.TrimEnd(mychar), textBoxPassword.Text);

File.Delete(e\_file);

}

}

}

// для выбранных папок

if (listBoxFolders.Items.Count > 0)

{

for (int num = 0; num < listBoxFolders.Items.Count; num++)

{

string d\_file = "" + listBoxFolders.Items[num];

string[] get\_files = Directory.GetFiles(d\_file);

foreach (string name\_file in get\_files)

{

if (name\_file.Trim().EndsWith(".!LOCKED") && File.Exists(name\_file))

{

DecryptFile(name\_file, name\_file.TrimEnd(mychar), textBoxPassword.Text);

File.Delete(name\_file);

}

}

}

}

} // buttonDecrypt\_Click()

} // class

} // namespace