УДК 004

Разработка WinForms приложения для шифрования файлов с помощью алгоритмов RSA и Rijndael

Эрдман Александр Алексеевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Научный руководитель:
Глаголев Владимир Александрович
Приамурский государственный университет имени Шолом-Алейхема
К.г.н., доцент, доцент кафедры информационных систем, математики и правовой информатики

Аннотация
В статье описан процесс создания оконного приложения с возможностью шифрования файлов. Программа написана на языке программирования C# с использованием библиотеки WinForms. Результатом исследования будет являться программа-шифровщик файлов.
Ключевые слова: C#, WinForms, шифрование

Development of a WinForms application for file encryption using RSA and Rijndael algorithms

Erdman Alexander Alekseevich
Sholom-Aleichem Priamursky State University
Student

Scientific supervisor:
Glagolev Vladimir Aleksandrovich
Sholom-Aleichem Priamursky State University
Ph.D, Associate Professor, Associate Professor of the Department of Information Systems, Mathematics and Legal Informatics

Abstract
The article describes the process of creating a windowed application with the ability to encrypt files. The program is written in the C# programming language using the WinForms library. The result of the study will be a file encryptor program.
Keywords: C#, WinForms, encryption

1 Введение

1.1 Актуальность

В эпоху информационных технологий, когда цифровые данные стали неотъемлемой частью нашей повседневной жизни, вопрос безопасности и

конфиденциальности информации занимает особенно важное место. Широкое распространение интернета, электронной почты, онлайн-банкинга и электронной медицинской документации делает неотложной задачу защиты чувствительных данных от несанкционированного доступа. В этом контексте шифрование данных выступает как ключевой механизм обеспечения безопасности в цифровой эре. История шифрования насчитывает века развития, начиная с простейших методов замены символов и переходя к сложным математическим алгоритмам. В древности шифрование применялось для передачи военной информации и обеспечения секретности политических посланий. С течением времени технологии шифрования стали эволюционировать, подстраиваясь под вызовы современных вычислительных мощностей и методов атак. От классических шифров, таких как шифр Цезаря, до современных квантовых алгоритмов, история шифрования является захватывающим путеводителем по развитию человеческого стремления обеспечить секретность и целостность своей информации.

В данной статье будет рассмотрен процесс создания программы, которая используют такие популярных методы шифрования как RSA и Rijndael.

1.2 Обзор исследований

И.О. Дадашов и Д.В. Гужва в своей научной работе изучили проблему работы алгоритма шифрования RSA с применением разработанной программой для шифрования информации [1]. М.В. Бабичев и А.С. Усачева в своём труде реализовали известные виды атак на систему шифрования RSA, представлены примеры с присутствием уязвимостей для данных атак и даны рекомендации в отношении устранения конкретных уязвимостей, для обеспечения стойкости криптосистемы [2]. Е.В. Ставер описал работу алгоритма шифрования RSA [3]. В.Е. Брыксин, Н.В. Грищенко и О.В. Ильин привели описание сетевого приложения для обмена данными между удаленными хостами на основе сокетного соединения с использованием алгоритма шифрования Rijndael [4]. А.В. Дмитришин и В.А. Лужецкий в контексте исследования структуры работы блоковых симметричных шифров предложили модель управляемого режима сцепления блоков зашифрованного текста [5].

1.3 Цель исследования

Целью исследования является создание и описание приложения на базе библиотеки WinForms с реализацией алгоритмов шифрования RSA и Rijndael.

2 Материалы и методы

Для реализации приложения используется язык программирования C#, библиотека WinForms и криптографический модуль Cryptography. В качестве IDE используется Visual Studio 2019.

3 Результаты и обсуждения

Создание приложения начинается с оформления визуальной составляющей. Создаётся окно, которому уславливаются соответствующие размеры. Далее на макете размещаются следующие элементы: два textbox, в которые будут вводится путь к файлу и пароль для генерации секретного ключа; Button, которые при нажатии на них будут запускать различные функции; CheckBox, который будет отвечать за добавление приватных параметров в ключ; в RichTextBox будет выводиться информация о файле, который будет зашифрован.

Перед реализацией главной программы приложения создаётся класс Program.cs, который является точкой входа в приложение (рис. 1).

```
static class Program
{
    [STAThread]
    Ссылок: 0
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new Encryptor());
    }
}
```

Рисунок 1. Класс Program

В данном классе указывается атрибут [STAThread], который указывает на то, что приложение будет выполняться в однопоточном режиме (Single-Threaded Apartment, STA). Данный атрибут предотвращает проблемы при подключении компонентов приложением. После объявляется метод Main, который содержит методы: для установки стандартного стиля оформления для элементов пользовательского интерфейса Windows Forms; метод для установки рендеринга текста, который принимает значение false, что позволяет использовать более качественный метод рендеринга текста; последний метод запускает главное окно приложения, которое представлено объектом Encryptor. Encryptor представляет форму или главное окно приложения и отображает его на экране. После вызова Application.Run(), приложение начнет свою работу, и пользовательский интерфейс будет активен для взаимодействия.

Следующим шагом является реализация ранее упомянутого класса Encryptor. Данный класс будет объявлен в классе Cryptor.cs.

После объявления класса проводится импорт модуля для шифрования данных - System.Security.Cryptography. Следующим шагом является создание объекта CspParameters, который содержит параметры для криптографического поставщика служб (CSP - Cryptographic Service Provider). CSP – это архитектурное решение, предоставляющее криптографические услуги, такие как шифрование и хэширование. Следующий объект (RSACryptoServiceProvider) предоставляет функциональность по работе с

алгоритмом шифрования RSA. Далее происходит создание конструктора Encryptor, который в дальнейшем будет использоваться для создания объектов и свойств в программе (рис. 2).

```
CspParameters cspPar = new CspParameters();

RSACryptoServiceProvider rsa;

Ссылок: 1
public Encryptor()
{
    InitializeComponent();
}
```

Рисунок 2. Создание объектов и конструктора

Перед последующей реализацией программы создаётся и настраивается графическая часть приложения (рис. 3).

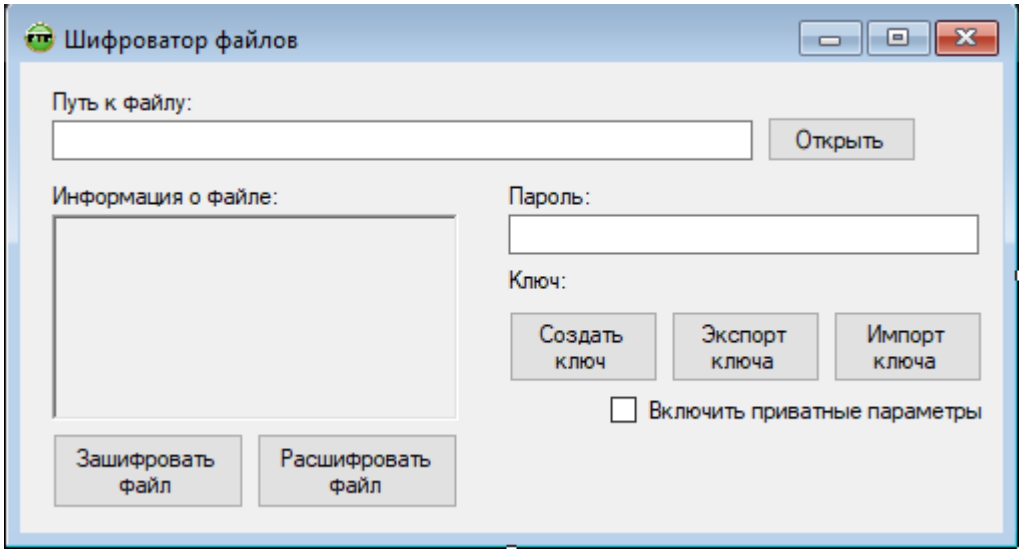


Рисунок 3. Внешний вид приложения

Метод UpdateInfo позволяет обновлять информацию о файле на основе введенного пользователем имени файла (рис. 4).

В начале текст получается из поля BoxFilename. Затем осуществляется проверка на пустое значение. В случае, если значение было е пустым, то есть имя файла введено, то создаётся объект FileInfo для указанного файла. Осуществляется проверка файла на существование в системе. После проверки начинается создание строк Info, которая содержит в себе информацию о файле: его имя, расширение, размер и дату создания. Для реализации данного действия используется конструкция if else.

В случае, когда файла не существует или имя файла не было введено, то текстовое поле BoxInfo очищается.

```
private void UpdateInfo()
{
    String Filename = BoxFilename.Text;
    if (!Filename.Equals(""))
    {
        FileInfo oFileInfo = new FileInfo(Filename);

        if (oFileInfo.Exists)
        {
            String Info = "";
            Info += "Имя: " + oFileInfo.Name + "\n";
            Info += "Расширение: " + oFileInfo.Extension + "\n";

            long Size = oFileInfo.Length;
            if(Size < 1024)
            {
                Info += "Размер: " + Size + " Bytes\n";
            }
            else if(Size >= 1024 && Size < 1048576)
            {
                Info += "Размер: " + (Size / 1024 + 1) + " KB\n";
            }
            else if (Size >= 1048576 && Size < 1073741824)
            {
                Info += "Размер: " + (Size / 1048576 + 1) + " MB\n";
            }
            else if (Size >= 1073741824 && Size < 1099511627776)
            {
                Info += "Размер: " + (Size / 1073741824 + 1) + " GB\n";
            }
            else
            {
                Info += "Размер: " + (Size / 1099511627776 + 1) + " TB\n";
            }

            Info += "Дата создания: " + oFileInfo.CreationTime + "\n";

            BoxInfo.Text = Info;
        }

        else
        {
            BoxInfo.Text = "";
        }
    }
    else
    {
        BoxInfo.Text = "";
    }
}
```

Рисунок 4. Метод получения и обновление информации о файле

После получения информации о файле следует реализация функции для шифрования данных – EncryptFile (рис. 5).

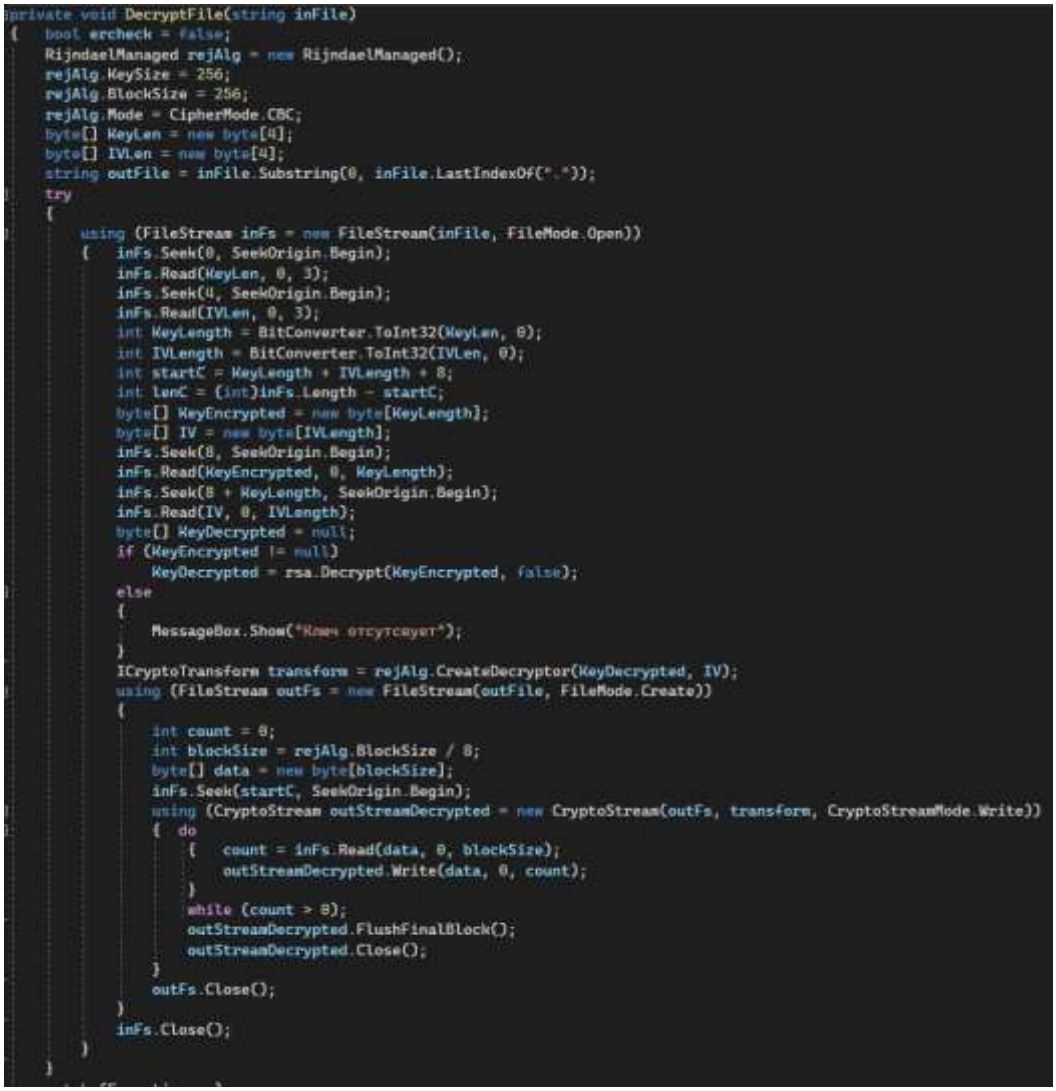
В начале данного метода создаётся таймер, с помощью которого можно измерить и увидеть время, затраченное на шифрование. Данное действие является дополнительным. Следующим является создание объекта RijndaelManaged для доступа к управляемой версии алгоритма Rijndael. После задаются размеры ключа и размер блока в битах. Затем происходит установка режима блочного шифра - Сцепление блоков шифротекста. В дополнении к этому создаётся объект шифратора с генерированным ключом и вектором инициализации, для которого создаётся массив с длинами. Для того, чтобы преобразовать длины в байты и запись в массивы используются методы

keyEncrypted.Length и rejAlg.IV.Length. С помощью оператора добавления к строковой переменной части происходит формирование расширение зашифрованного файла - enc. В дальнейшем с помощью оператора обработки исключений try происходит запись в выходной файл следующие данные: длины ключа, вектора инициализации, а также зашифрованного ключа Rijndael, непосредственно вектора инициализации и зашифрованной части слова. Данные действия осуществляются с помощью метода Write. CryptoStream позволяет зашифровать само содержимое файла с помощью алгоритма Rijndael. Оставшаяся часть кода обрабатывает блоки данных. ПО завершению программы происходит остановка таймера и всплывает окно с предупреждением, что файл успешно был зашифрован.

```
private void EncryptFile(string inFile)
{
    System.Diagnostics.Stopwatch stopwatch = new System.Diagnostics.Stopwatch();
    stopwatch.Start();
    bool ercheck = false;
    RijndaelManaged rejAlg = new RijndaelManaged();
    rejAlg.KeySize = 256;
    rejAlg.BlockSize = 256;
    rejAlg.Mode = CipherMode.CBC;
    byte[] keyEncrypted = rsa.Encrypt(rejAlg.Key, false);
    byte[] KeyLen = new byte[4];
    byte[] IVLen = new byte[4];
    int lKey = keyEncrypted.Length;
    KeyLen = BitConverter.GetBytes(lKey);
    int lIV = rejAlg.IV.Length;
    IVLen = BitConverter.GetBytes(lIV);
    string outFile = inFile + ".enc";
    try
    {
        using (FileStream outFs = new FileStream(outFile, FileMode.Create))
        {
            outFs.Write(KeyLen, 0, 4);
            outFs.Write(IVLen, 0, 4);
            outFs.Write(keyEncrypted, 0, lKey);
            outFs.Write(rejAlg.IV, 0, lIV);
            using (CryptoStream outStreamEncrypted = new CryptoStream(outFs, transform, CryptoStreamMode.Write))
            {
                int count = 0;
                int blockSize = rejAlg.BlockSize / 8;
                byte[] data = new byte[blockSize];

                using (FileStream inFs = new FileStream(inFile, FileMode.Open))
                {
                    do
                    {
                        count = inFs.Read(data, 0, blockSize);
                        outStreamEncrypted.Write(data, 0, count);
                    }
                    while (count > 0);
                    inFs.Close();
                }
                outStreamEncrypted.FlushFinalBlock();
                outStreamEncrypted.Close();
            }
            outFs.Close();
        }
    }
    catch(Exception ex)
    {
        MessageBox.Show(ex.Message);
        ercheck = true;
    }

    stopwatch.Stop();
    if (!ercheck)
    {

        MessageBox.Show("Файл был зашифрован. Время шифрования:\n" + stopwatch.Elapsed);
    }
}
```

Рисунок 5. Метод шифрования EncryptFile

После функции шифрования нужно реализовать функцию дешифрования файла (рис. 6).



```
private void DecryptFile(string inFile)
{   bool ercheck = false;
    RijndaelManaged rejAlg = new RijndaelManaged();
    rejAlg.KeySize = 256;
    rejAlg.BlockSize = 256;
    rejAlg.Mode = CipherMode.CBC;
    byte[] KeyLen = new byte[4];
    byte[] IVLen = new byte[4];
    string outFile = inFile.Substring(0, inFile.LastIndexOf("."));
    try
    {
        using (FileStream inFs = new FileStream(inFile, FileMode.Open))
        {   inFs.Seek(0, SeekOrigin.Begin);
            inFs.Read(KeyLen, 0, 3);
            inFs.Seek(4, SeekOrigin.Begin);
            inFs.Read(IVLen, 0, 3);
            int KeyLength = BitConverter.ToInt32(KeyLen, 0);
            int IVLength = BitConverter.ToInt32(IVLen, 0);
            int startC = KeyLength + IVLength + 8;
            int lenC = (int)inFs.Length - startC;
            byte[] KeyEncrypted = new byte[KeyLength];
            byte[] IV = new byte[IVLength];
            inFs.Seek(8, SeekOrigin.Begin);
            inFs.Read(KeyEncrypted, 0, KeyLength);
            inFs.Seek(8 + KeyLength, SeekOrigin.Begin);
            inFs.Read(IV, 0, IVLength);
            byte[] KeyDecrypted = null;
            if (KeyEncrypted != null)
                KeyDecrypted = rsa.Decrypt(KeyEncrypted, false);
            else
            {
                MessageBox.Show("Ключ отсутсвует");
            }
            ICryptoTransform transform = rejAlg.CreateDecryptor(KeyDecrypted, IV);
            using (FileStream outFs = new FileStream(outFile, FileMode.Create))
            {
                int count = 0;
                int blockSize = rejAlg.BlockSize / 8;
                byte[] data = new byte[blockSize];
                inFs.Seek(startC, SeekOrigin.Begin);
                using (CryptoStream outStreamDecrypted = new CryptoStream(outFs, transform, CryptoStreamMode.Write))
                {  do
                    {   count = inFs.Read(data, 0, blockSize);
                        outStreamDecrypted.Write(data, 0, count);
                    }
                    while (count > 0);
                    outStreamDecrypted.FlushFinalBlock();
                    outStreamDecrypted.Close();
                }
                outFs.Close();
            }
            inFs.Close();
        }
    }
```

Рисунок 6. Метод дешифрования DecryptFile

Как и в предыдущем методе задаётся переменная bool ercheck, которая является флагом отслеживания ошибок во время выполнения. Первая часть функции аналогична функции шифрования. В обработке исключений try в начале открывается зашифрованный файл. Далее происходит чтение первых 4 байтов, которые являются длинной ключа. Также считываются последующие 4 байта, которые являются длиной вектора инициализации. Полученные данные записываются в буфер обмена. Для перевода длин из байтов в целочисленные значения используется метод BitConverter. Затем следует определение начала зашифрованного текста и его длину. Для работы дешифратора также создаются два массива для зашифрованного ключа и вектора инициализации, после чего происходит их получение и запись в буфер. Для расшифрованного ключа объявляется отдельный массив keyDecrypted.

Далее используется директива using, внутри которой происходит расшифровка данных из входного потока (inFs) с использованием алгоритма шифрования (rejAlg). Создается объект дешифратора (transform) с заданным

ключом и вектором инициализации. Расшифрованные данные записываются в выходной файл (outFile). Для эффективной обработки больших файлов и экономии памяти процесс дешифрации осуществляется блоками определенного размера. После завершения дешифрации и записи данных в выходной поток происходит очистка буфера и закрытие потоков.

Основные функции программы были реализованы. Далее программируются элементы интерфейса: кнопка «открыть», текстовое поле, где выводится информация о файле (рис. 7).

```
private void ButtonBrowse_Click(object sender, EventArgs e)
{
    DialogResult dialog = OpenFileDialog.ShowDialog();

    if (dialog == DialogResult.OK)
    {
        BoxFilename.Text = OpenFileDialog.FileName;
    }
}

private void BoxFilename_TextChanged(object sender, EventArgs e)
{
    UpdateInfo();
}
```

Рисунок 7. Реализация кнопок «Открыть»,

Код, изображённый на рисунке 7, относится к графическому интерфейсу приложения и обрабатывает события взаимодействия с файлом. При нажатии кнопки "Открыть файл" (ButtonBrowse_Click) открывается диалоговое окно для выбора файла, и если пользователь выбирает файл и подтверждает выбор, то путь к выбранному файлу сохраняется в текстовом поле BoxFilename. Второй обработчик (BoxFilename_TextChanged) вызывает функцию UpdateInfo() при изменении содержимого текстового поля с именем файла. Таким образом, пользователь может выбрать файл и автоматически обновить информацию о нем, вызывая UpdateInfo().

Далее следует реализация обработчиков событий для кнопок "Шифрование" (ButtonEncrypt_Click) и "Расшифрование" (ButtonDecrypt_Click) (рис. 8). В каждом из них сначала извлекается путь к файлу из текстового поля BoxFilename, затем создается объект FileInfo для проверки существования файла. После этого происходит проверка наличия объекта RSA-ключа. В случае отсутствия файла выводится сообщение об ошибке "File not found", если ключ не установлен - "Key not set". Для расшифровки также проверяется наличие приватного ключа в объекте RSA и соответствие расширения файла "enc". При выполнении всех проверок вызывается соответствующая функция - EncryptFile или DecryptFile.

```
private void ButtonEncrypt_Click(object sender, EventArgs e)
{
    String Filename = BoxFilename.Text;
    FileInfo oFileInfo = new FileInfo(Filename);

    if (!oFileInfo.Exists)
    {
        MessageBox.Show("Файл не найден;");
    }
    else if (rsa == null)
    {
        MessageBox.Show("Ключ не установлен!");
    }
    else
    {
        EncryptFile(oFileInfo.FullName);
    }
}

private void ButtonDecrypt_Click(object sender, EventArgs e)
{
    String Filename = BoxFilename.Text;
    FileInfo oFileInfo = new FileInfo(Filename);

    if (!oFileInfo.Exists)
    {
        MessageBox.Show("Файл не найден;");
    }
    else if (rsa == null)
    {
        MessageBox.Show("Key not set!");
    }

    else if (rsa.PublicOnly)
    {
        MessageBox.Show("Не удается расшифровать с помощью открытого ключа");
    }
    else if (oFileInfo.Extension != ".enc")
    {
        MessageBox.Show("Необходимо выбрать файл .enc для расшифровки");
    }
    else
    {
        DecryptFile(oFileInfo.FullName);
    }
}
```

Рисунок 8. Обработчик нажатия кнопок «Зашифровать файл» и «Расшифровать файл»

В конце прописываются кнопки, отвечающие за ключи – создание ключа, его экспорт и импорт (рис. 9 и 10). При нажатии кнопки "Создать ключ" (ButtonCreateKey_Click) сначала проверяется, что введенный пароль состоит из минимум 4 символов. Затем создается объект RSA с использованием заданного пароля и сохраняется в контейнере. При успешном создании ключа выводится информация о нем. Кнопка "Экспорт" (ButtonExport_Click) обрабатывает сохранение ключа в выбранную папку. Если ключ существует, открывается диалоговое окно выбора папки, после чего происходит запись ключа в файл. Возможно добавление приватных параметров в ключ, в зависимости от установленной опции. Если файл существует, он перезаписывается.

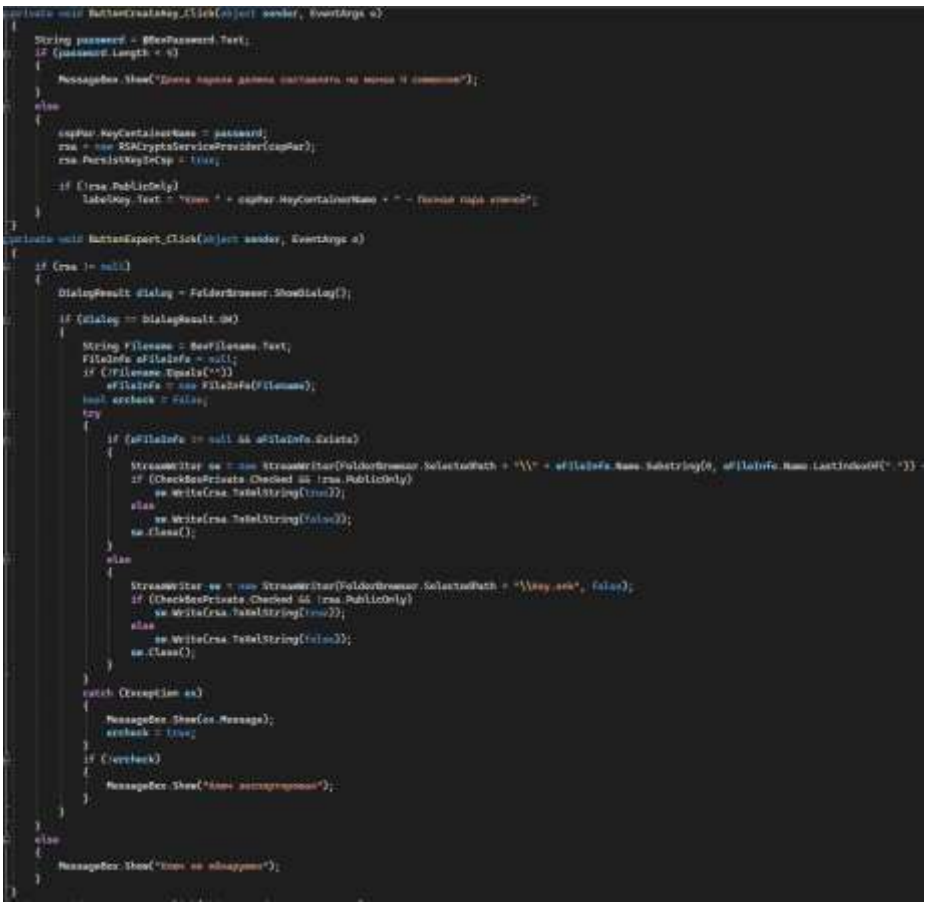


Рисунок 9. Обработчики событий кнопок «Создать ключ» и «Экспорт ключа»

```
private void ButtonImport_Click(object sender, EventArgs e)
{
    DialogResult dialog = OpenKeyFile.ShowDialog();

    if (dialog == DialogResult.OK)
    {
        String Filename = OpenKeyFile.FileName;
        FileInfo oFileInfo = new FileInfo(Filename);

        try
        {
            if (oFileInfo.Extension == ".enk")
            {
                StreamReader sr = new StreamReader(oFileInfo.FullName);
                if (rsa == null)
                {
                    cspPar.KeyContainerName = "Temp";
                    rsa = new RSACryptoServiceProvider(cspPar);
                }
                string keytxt = sr.ReadToEnd();
                rsa.FromXmlString(keytxt);
                rsa.PersistKeyInCsp = true;
                if (rsa.PublicOnly == true)
                    labelKey.Text = "Ключ: Открытый ключ от " + oFileInfo.Name;
                else
                    labelKey.Text = "Ключ: Полная пара ключей от " + oFileInfo.Name;
                sr.Close();
            }
            else
            {
                MessageBox.Show("Должен быть выбран .enk файл");
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
}
```

Рисунок 10. Обработчик события кнопки «Импорт ключа»

Кнопка "Импорт" (ButtonImport_Click) позволяет загрузить ключ из выбранного файла. При выборе файла с расширением "enk", происходит чтение файла, инициализация объекта RSA с ключом из файла, и выводится информация о ключе в интерфейсе. Если выбран некорректный файл, выводится соответствующее сообщение об ошибке.

Приложение реализовано. После запуска приложения можно видеть, что приложение выполняет весь свой запрограммированный функционал (рис. 11).

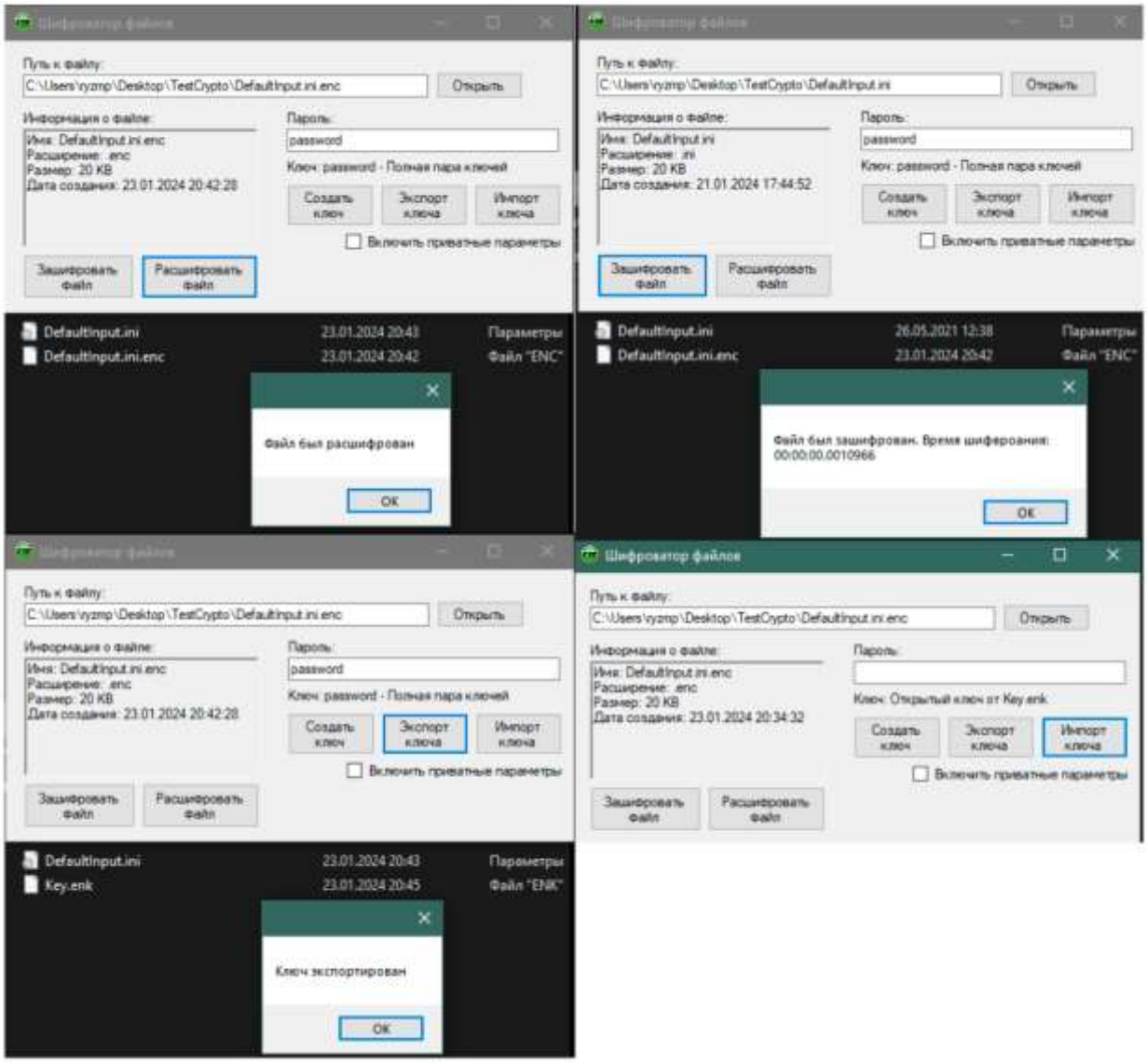


Рисунок 11. Результаты работы программы

Таким образом в процессе исследования было реализовано приложение, которое предоставляет возможность шифрования и дешифрования с помощью алгоритмов RSA и Rijndael. Функционал приложения был описан и работоспособность приложения была проверена на практике.

Библиографический список

1. Дадашов И.О., Гужва Д.В. Определение условий применимости алгоритма шифрования RSA для ключей шифрования длиной менее 256 бит // Теория и практика проектного образования. 2021. № 1 (17). С. 39-42.
2. Бабичева М.В., Усачева А.С. Исследование уязвимостей алгоритма шифрования RSA // Вестник Донецкого национального университета. Серия Г: Технические науки. 2020. № 3. С. 33-37.
3. Ставер Е.В. Алгоритм RSA. Шифрование и дешифрование текстовых сообщений // Научный аспект. 2012. № 3. С. 88-89.
4. Брыксин В.Е., Грищенко Н.В., Ильин О.В. Приложение для сетевой передачи данных с применением симметричного шифрования // В сборнике: Информационные технологии в сфере РСЧС и ГО. Сборник трудов секции № 12 XXXIII Международной научно-практической конференции. Химки, 2023. С. 172-178.
5. Дмитришин А.В., Лужецкий В.А. Режим управляемого сцепления блоков зашифрованного текста // Винницкий национальный технический университет. 2009. № 1 (82). С. 34-36.