Казанский федеральный университет

**ы**

**Садыков Ильдар Наилевич**

**09-812**

**Отчет по C#**

**rsa, md5 шифратор дешифратор**

Казань

2021 - 2022

**Содержание**

[Функционал Приложения 3](#_Toc90229197)

[Алгоритмы 3](#_Toc90229198)

[Алгоритм RSA 3](#_Toc90229199)

[Алгоритм MD5 5](#_Toc90229200)

[Описание программы 7](#_Toc90229201)

[Листинг 22](#_Toc90229202)

# Функционал Приложения

Приложение реализует следующие возможности:

1. Шифрация/Дешифрация текста алгоритмом RSA
2. Шифрация/Дешифрация текста алгоритмом MD5
3. Шифрация/Дешифрация файла алгоритмом RSA
4. Шифрация/Дешифрация файла алгоритмом MD5

# Алгоритмы

Прежде чем изучать алгоритмы, очень важно понять, что такое криптография. Криптография - это практика и изучение методов, которые используются для преобразования простого текста в неразборчивый текст, понятный только получателю. По соображениям безопасности очень важно зашифровать данные, а также отправить и сохранить их в определенном формате, чтобы они не могли быть пропущены и, следовательно, неправильно использованы. Существует тмного алгоритмов, разработанных для безопасной передачи и хранения данных, которые имеют свои плюсы и минусы. Эти алгоритмы не только защищают данные от кражи, но и обеспечивают аутентификацию пользователя.

# Алгоритм RSA

**RSA** - [криптографический алгоритм с открытым ключом](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%BC_%D0%BA%D0%BB%D1%8E%D1%87%D0%BE%D0%BC), основывающийся на [вычислительной сложности](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C) [задачи факторизации](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D1%86%D0%B5%D0%BB%D1%8B%D1%85_%D1%87%D0%B8%D1%81%D0%B5%D0%BB) больших [целых чисел](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BB%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE).

Алгоритм RSA включает в себя четыре этапа: генерация ключей, передача ключей, шифрование и расшифрование.

В основу криптографической системы с открытым ключом RSA положена сложность [задачи факторизации](https://neerc.ifmo.ru/wiki/index.php?title=%D0%A0%D0%B0%D0%B7%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BD%D0%B0_%D0%BC%D0%BD%D0%BE%D0%B6%D0%B8%D1%82%D0%B5%D0%BB%D0%B8_(%D1%84%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F)) произведения двух больших простых чисел. Для шифрования используется операция возведения в степень по модулю большого числа. Для дешифрования (обратной операции) за разумное время необходимо уметь вычислять [функцию Эйлера](https://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%D0%AD%D0%B9%D0%BB%D0%B5%D1%80%D0%B0) от данного большого числа, для чего необходимо знать разложение числа на простые множители. В криптографической системе с открытым ключом каждый участник располагает как открытым ключом (*public key*), так и закрытым ключом (*private key*). В криптографической системе RSA каждый ключ состоит из пары целых чисел. Каждый участник создаёт свой открытый и закрытый ключ самостоятельно. Закрытый ключ каждый из них держит в секрете, а открытые ключи можно сообщать кому угодно или даже публиковать их. Открытый и закрытый ключи каждого участника обмена сообщениями в криптосистеме RSA образуют «согласованную пару» в том смысле, что они являются взаимно обратными. То есть для любых допустимых пар открытого и закрытого ключей (p, s) существуют соответствующие функции шифрования Ep(x) и расшифрования Ds(x) такие, что для любого сообщения m ∈ M, где M - множество допустимых сообщений, m = Ds(Ep(m)).

Создание открытого и секретного ключей

RSA - ключи генерируются следующим образом:

1. Выбираются два различных [случайных простых числа](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9F%D1%80%D0%BE%D1%81%D1%82%D1%8B%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%B0) p и q заданного размера (например, 1024 бита каждое).
2. Вычисляется их произведение n = p ⋅ q, которое называется *модулем*.
3. Вычисляется значение [функции Эйлера](https://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%D0%AD%D0%B9%D0%BB%D0%B5%D1%80%D0%B0) от числа n: φ(n)=(p−1)⋅(q−1).
4. Выбирается целое число ee (1 < e < φ(n)), [взаимно простое](https://neerc.ifmo.ru/wiki/index.php?title=%D0%92%D0%B7%D0%B0%D0%B8%D0%BC%D0%BD%D0%BE_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%8B%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%B0) со значением функции φ(n). Обычно в качестве ee берут простые числа, содержащие небольшое количество единичных бит в двоичной записи.

* Число ee называется открытой экспонентой (*public exponent*)
* Время, необходимое для шифрования с использованием [быстрого возведения в степень](https://neerc.ifmo.ru/wiki/index.php?title=%D0%91%D1%8B%D1%81%D1%82%D1%80%D0%BE%D0%B5_%D0%B2%D0%BE%D0%B7%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B2_%D1%81%D1%82%D0%B5%D0%BF%D0%B5%D0%BD%D1%8C), пропорционально числу единичных бит в ee.
* Слишком малые значения ee, например 33, потенциально могут ослабить безопасность схемы RSA.

1. Вычисляется число d, [мультипликативно](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9C%D1%83%D0%BB%D1%8C%D1%82%D0%B8%D0%BF%D0%BB%D0%B8%D0%BA%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8,_%D1%81%D0%B2%D0%B5%D1%80%D1%82%D0%BA%D0%B0_%D0%94%D0%B8%D1%80%D0%B8%D1%85%D0%BB%D0%B5) обратное к числу ee по модулю φ(n), то есть число, удовлетворяющее сравнению:

d ⋅ e ≡ (modφ(n)).

Примечание

Сравнеие двух целых чисел по модулю натурального числа m - математическая операция, позволяющая ответить на вопрос о том, дают ли два выбранных целых числа при делении на mm один и тот же остаток. Любое целое число при делении на mm дает один из m возможных остатков: число от 0 до m − 1.

* Число d называется секретной экспонентой. Обычно, оно вычисляется при помощи [расширенного алгоритма Евклида](https://neerc.ifmo.ru/wiki/index.php?title=%D0%9D%D0%B0%D0%B8%D0%B1%D0%BE%D0%BB%D1%8C%D1%88%D0%B8%D0%B9_%D0%BE%D0%B1%D1%89%D0%B8%D0%B9_%D0%B4%D0%B5%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C).

1. Пара {e, n} публикуется в качестве открытого ключа RSA (*public key*).
2. Пара {d, n} играет роль закрытого ключа RSA (*private key*) и держится в секрете.

# Алгоритм MD5

На вход алгоритма поступает входной поток данных, хеш которого необходимо найти. Длина сообщения измеряется в битах и может быть любой (в том числе нулевой). Запишем длину сообщения в *L*. Это число целое и неотрицательное. Кратность каким-либо числам необязательна. После поступления данных идёт процесс подготовки потока к вычислениям.

Ниже приведены 5 шагов алгоритма:

**Шаг 1. Выравнивание потока**

Сначала к концу потока дописывают единичный бит. Затем добавляют некоторое число нулевых бит такое, чтобы новая длина потока {\displaystyle L'}стала [сравнима](https://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E_%D0%BD%D0%B0%D1%82%D1%83%D1%80%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE_%D1%87%D0%B8%D1%81%D0%BB%D0%B0) с 448 по модулю 512{\displaystyle L'=512\times N+448}. Выравнивание происходит в любом случае, даже если длина исходного потока уже сравнима с 448.

**Шаг 2. Добавление длины сообщения**

В конец сообщения дописывают 64-битное представление длины данных (количество бит в сообщении) до выравнивания. Сначала записывают младшие 4 байта, затем старшие. Если длина превосходит {\displaystyle 2^{64}-1} 2^64 - 1, то дописывают только младшие биты (эквивалентно взятию по модулю {\displaystyle 2^{64}} 2^64). После этого длина потока станет кратной 512. Вычисления будут основываться на представлении этого потока данных в виде массива слов по 512 бит.

**Шаг 3. Инициализация буфера**

Для вычислений инициализируются четыре переменные размером по 32 бита, начальные значения которых задаются шестнадцатеричными числами (порядок байтов [little-endian](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA_%D0%B1%D0%B0%D0%B9%D1%82%D0%BE%D0%B2#%D0%9F%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA_%D0%BE%D1%82_%D0%BC%D0%BB%D0%B0%D0%B4%D1%88%D0%B5%D0%B3%D0%BE_%D0%BA_%D1%81%D1%82%D0%B0%D1%80%D1%88%D0%B5%D0%BC%D1%83)):

А = 01 23 45 67; // 67452301h

В = 89 AB CD EF; // EFCDAB89h

С = FE DC BA 98; // 98BADCFEh

D = 76 54 32 10. // 10325476h

В этих переменных будут храниться результаты промежуточных вычислений. Начальное состояние ABCD называется инициализирующим вектором.

**Шаг 4. Вычисление в цикле**

Определим функции и константы, которые понадобятся нам для вычислений.

Этап 1

/\* [abcd k s i] a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). \*/

[ABCD 0 7 1][DABC 1 12 2][CDAB 2 17 3][BCDA 3 22 4]

[ABCD 4 7 5][DABC 5 12 6][CDAB 6 17 7][BCDA 7 22 8]

[ABCD 8 7 9][DABC 9 12 10][CDAB 10 17 11][BCDA 11 22 12]

[ABCD 12 7 13][DABC 13 12 14][CDAB 14 17 15][BCDA 15 22 16]

Этап 2

/\* [abcd k s i] a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). \*/

[ABCD 1 5 17][DABC 6 9 18][CDAB 11 14 19][BCDA 0 20 20]

[ABCD 5 5 21][DABC 10 9 22][CDAB 15 14 23][BCDA 4 20 24]

[ABCD 9 5 25][DABC 14 9 26][CDAB 3 14 27][BCDA 8 20 28]

[ABCD 13 5 29][DABC 2 9 30][CDAB 7 14 31][BCDA 12 20 32]

Этап 3

/\* [abcd k s i] a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). \*/

[ABCD 5 4 33][DABC 8 11 34][CDAB 11 16 35][BCDA 14 23 36]

[ABCD 1 4 37][DABC 4 11 38][CDAB 7 16 39][BCDA 10 23 40]

[ABCD 13 4 41][DABC 0 11 42][CDAB 3 16 43][BCDA 6 23 44]

[ABCD 9 4 45][DABC 12 11 46][CDAB 15 16 47][BCDA 2 23 48]

Этап 4

/\* [abcd k s i] a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). \*/

[ABCD 0 6 49][DABC 7 10 50][CDAB 14 15 51][BCDA 5 21 52]

[ABCD 12 6 53][DABC 3 10 54][CDAB 10 15 55][BCDA 1 21 56]

[ABCD 8 6 57][DABC 15 10 58][CDAB 6 15 59][BCDA 13 21 60]

[ABCD 4 6 61][DABC 11 10 62][CDAB 2 15 63][BCDA 9 21 64]

Суммируем с результатом предыдущего цикла:

A = AA + A

B = BB + B

C = CC + C

D = DD + D

После окончания цикла необходимо проверить, есть ли ещё блоки для вычислений. Если да, то переходим к следующему элементу массива (*n* + 1) и повторяем цикл.

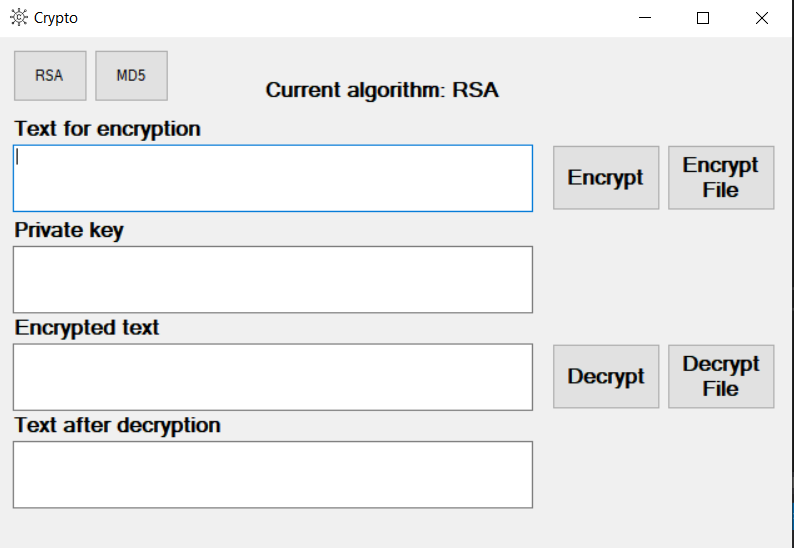
**Шаг 5. Результат вычислений**

Результат вычислений находится в буфере ABCD, это и есть хеш. Если выводить побайтово, начиная с младшего байта A и заканчивая старшим байтом D, то мы получим MD5-хеш. 1, 0, 15, 34, 17, 18…

Используя алгоритмы RSA и MD5 реал

# Описание программы

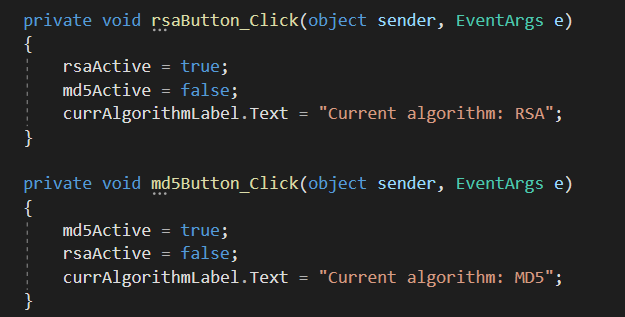
Приложение состоит из основной формы, которая выглядит следующим образом:



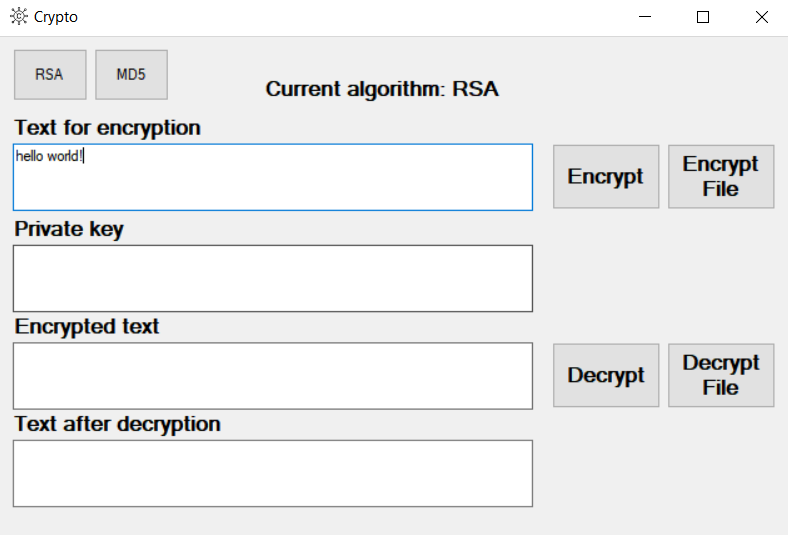
При нажатии на кнопку RSA текущим алгоритмом шифрации/дешифрации выбирается RSA, при нажатии кнопки MD5 алгоритм MD5, также в верху формы видно текущий используемый алгоритм с помощью класса Label.



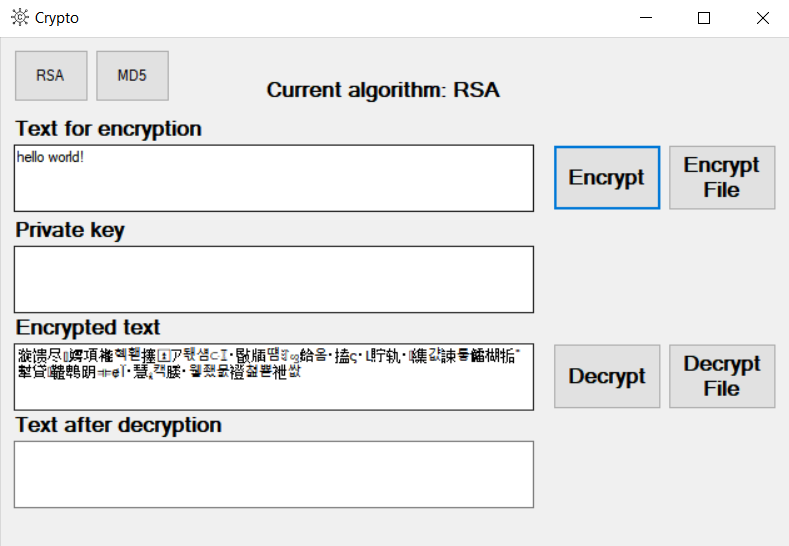
Код обрабатывающий нажатие кнопок:



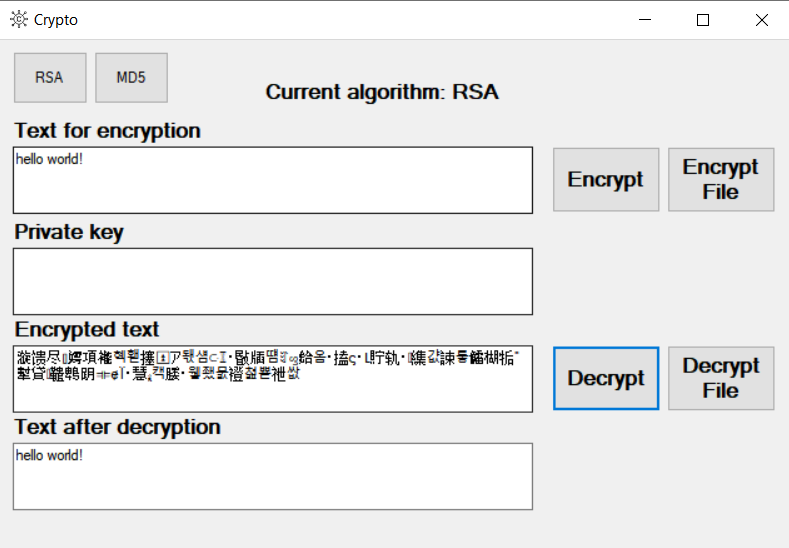
Зашифруем текст с помощью алгоритма RSA:



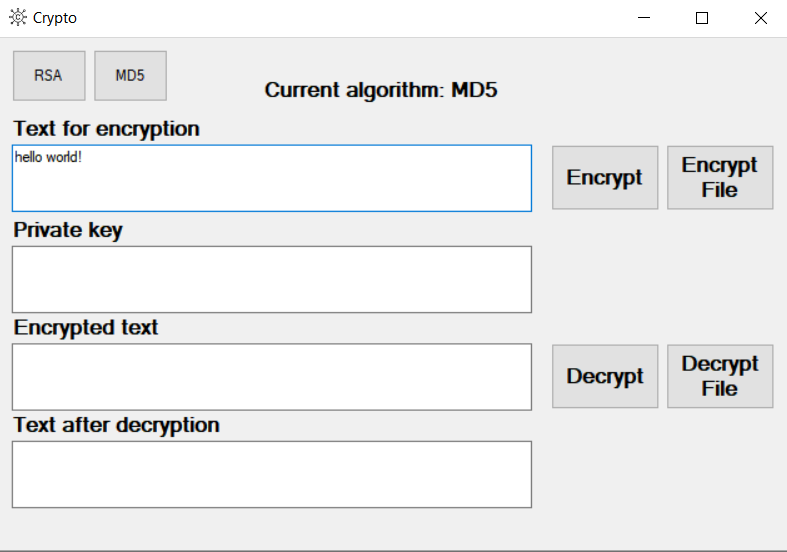
Нажимаем кнопку Encrypt и получаем зашифрованный текст:



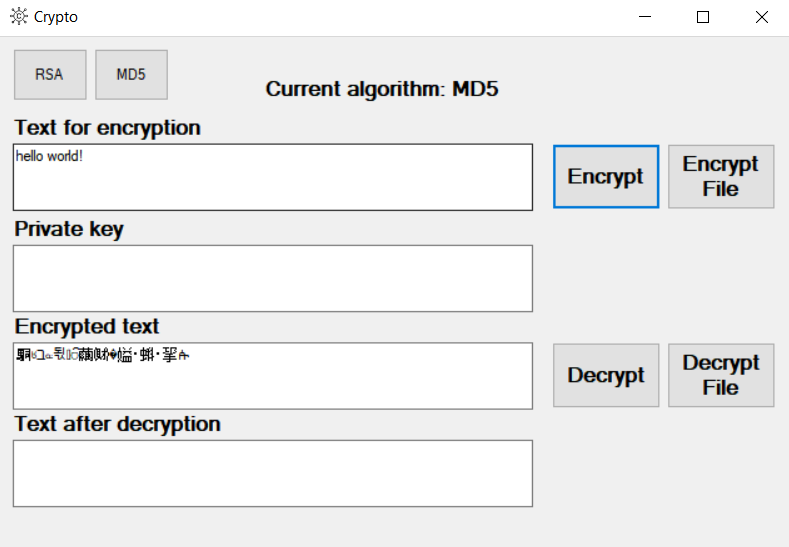
Расшифруем текст нажав кнопку Decrypt:



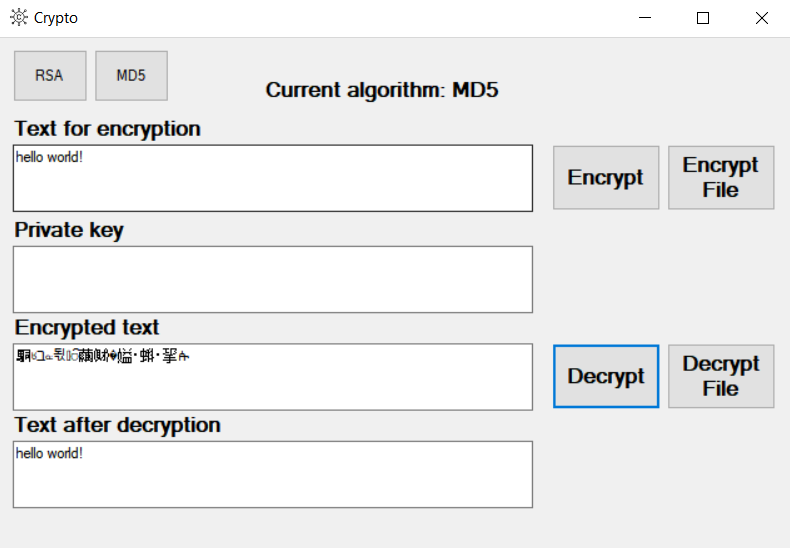
Зашифруем текст с помощью алгоритма MD5:



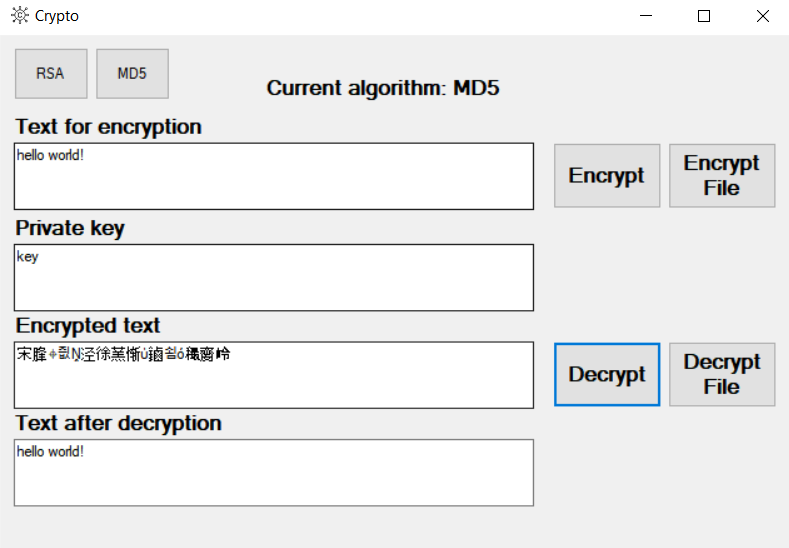
Нажимаем кнопку Encrypt и получаем зашифрованный текст:



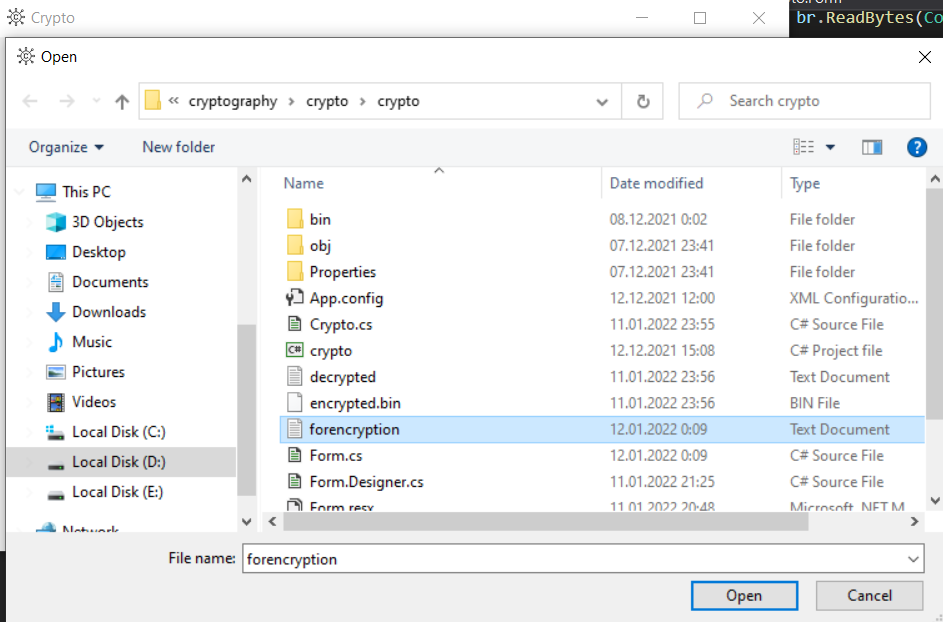
Расшифруем текст нажав кнопку Decrypt:



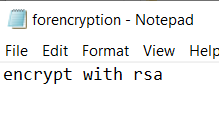
Также для MD5 можно указать приватный ключ, по умолчанию он пустая строка, с приватным ключом получается абсолютно другой хэш



Зашифруем файл forencryption.txt с помощью алгоритма RSA, нажимаем кнопку Encrypt File, появляется окно с выбором файла реализованное с помощью класса OpenDialogFile

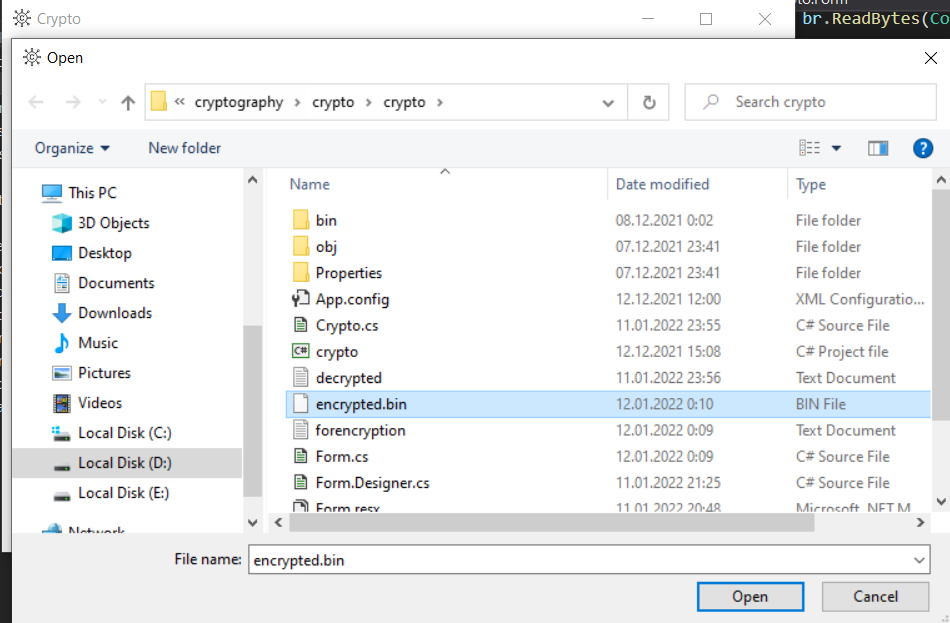


forencryption.txt

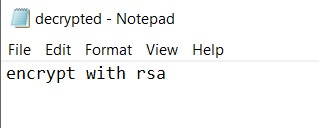


Выбираем файл forencryption.txt, после выбора файла создается новый файл encrypted.bin который содержит зашифрованные данные файла forencryption.txt в бинарном виде.

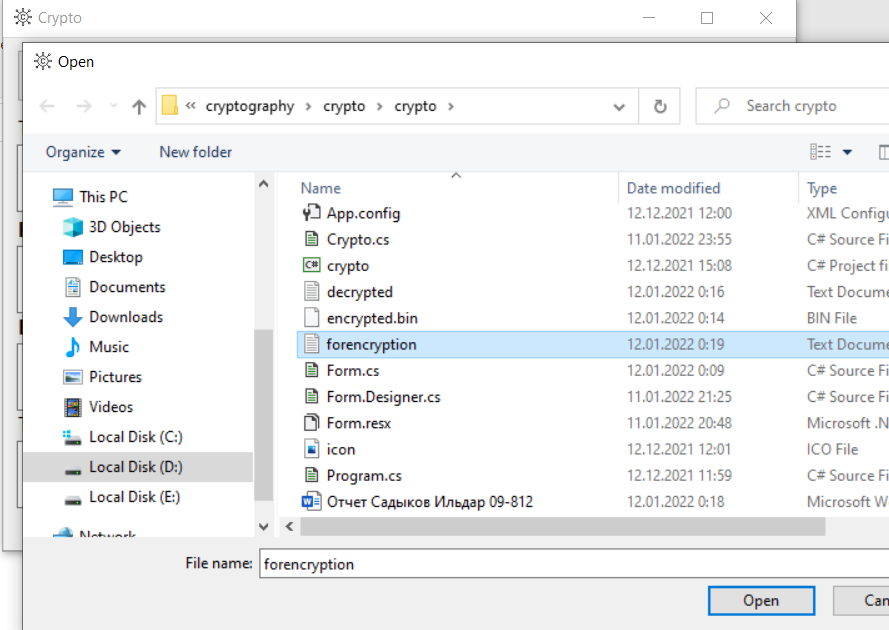
Расшифруем файл, нажимаем кнопку Decrypt File, появляется окно с выбором файла реализованное с помощью класса OpenDialogFile



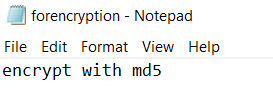
Выбираем файл encrypted.bin, расшифрованные данные помещаются в файл decrypted.txt



Зашифруем файл forencryption.txt с помощью алгоритма MD5, нажимаем кнопку Encrypt File, появляется окно с выбором файла реализованное с помощью класса OpenDialogFile

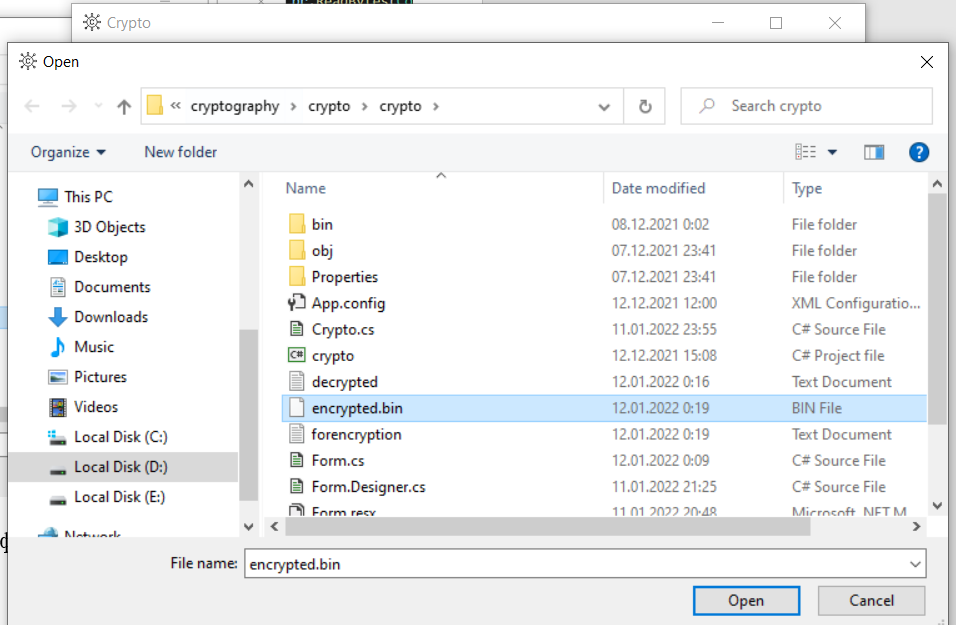


forencryption.txt

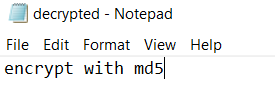


Выбираем файл forencryption.txt, после выбора файла создается новый файл encrypted.bin который содержит зашифрованные данные файла forencryption.txt в бинарном виде.

Расшифруем файл, нажимаем кнопку Decrypt File, появляется окно с выбором файла реализованное с помощью класса OpenDialogFile

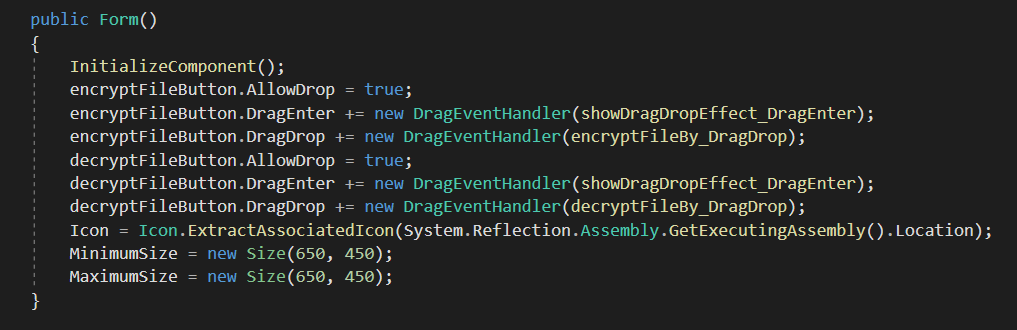


Выбираем файл encrypted.bin, расшифрованные данные помещаются в файл decrypted.txt

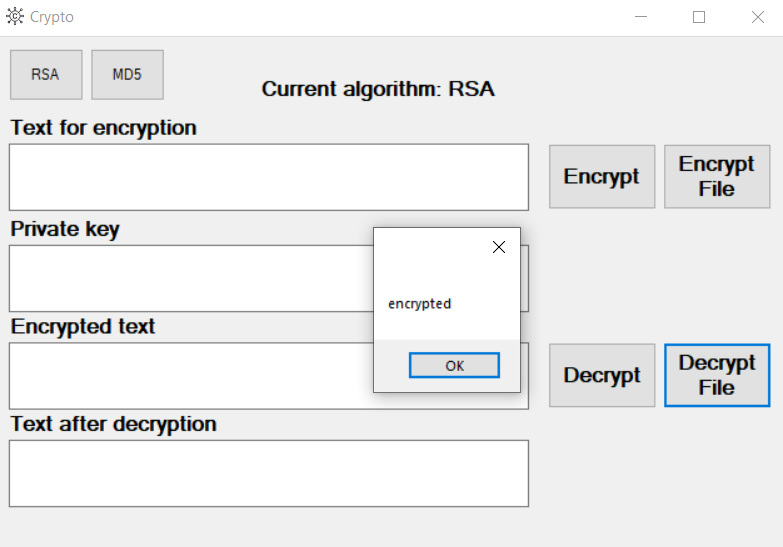


Также был реализован выбор файлов с помощью drag and drop, вызывается тот же функционал но только при переносе файла в пространство кнопки.

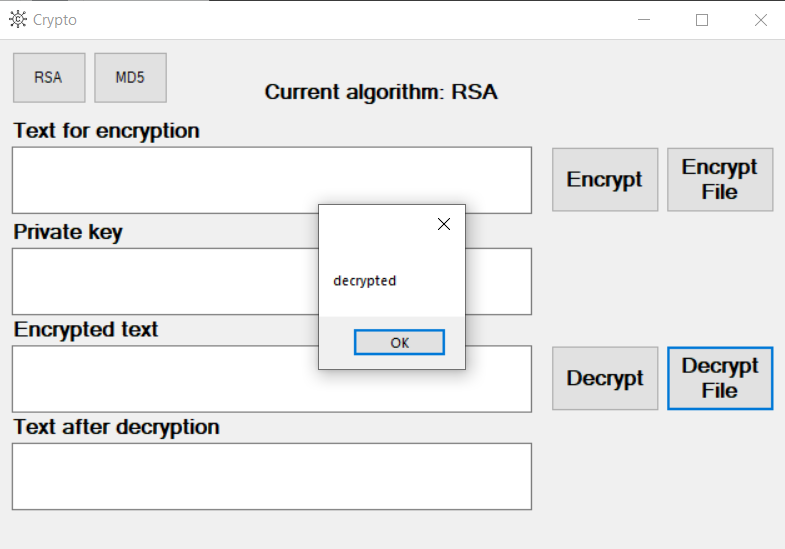
В конструкторе к кнопкам добавляются хэндлеры которые сработают при сбросе файла на эти кнопки.



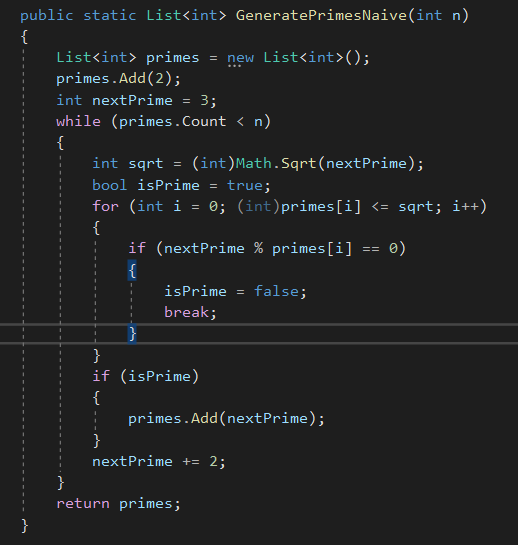
После переноса файла для шифрации программа говорит что файл зашифрован.



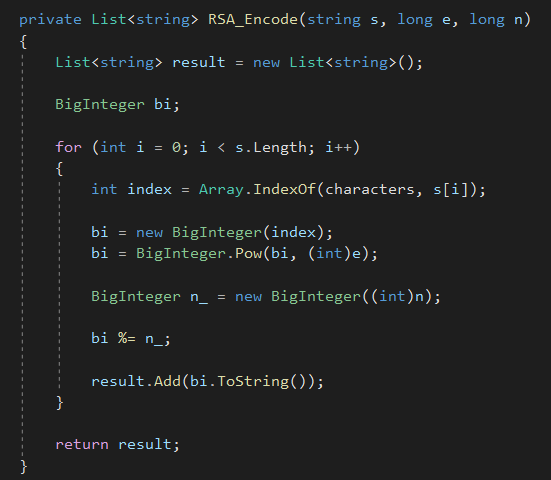
После переноса файла для дешифрации программа говорит что файл расшифрован.



Алгоритм RSA требует два рандомных простых числа, для этого при старте программы инициализирую массив простых чисел с помощью алгоритма решето Эратосфена и позже для шифрации выбираю рандомно элементы этого массива.

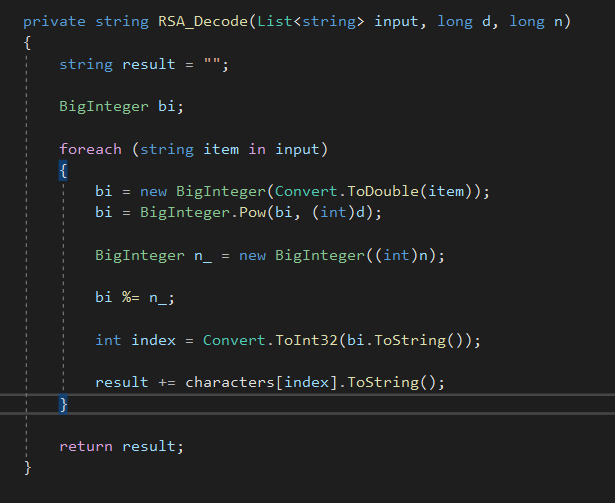


Метод, выполняющий шифрование строки алгоритмом RSA

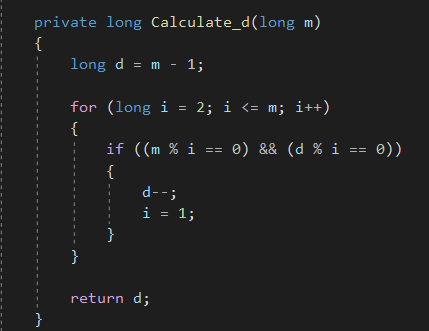


При возведении числа в степень в данном случае получаются очень большие числа, которые не помещаются ни в один из стандартных типов. Поэтому для их хранения используется экземпляр класса BigInteger. Этот класс позволяет хранить целые числа произвольной (любой) длины и выполнять математические операции с ними.

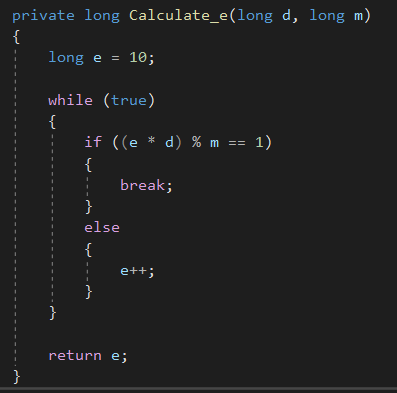
Метод, выполняющий расшифрование строки алгоритмом RSA



Вычисление параметра d (d должно быть взаимно простым с m).



Метод, вычисляющий значение параметра e.



# Листинг

**Код основной формы:**

using System;

using System.Drawing;

using System.IO;

using System.Text;

using System.Windows.Forms;

namespace crypto

{

public partial class Form : System.Windows.Forms.Form

{

private void showDragDropEffect\_DragEnter(object sender, DragEventArgs e)

{

e.Effect = DragDropEffects.All;

}

void encryptFileBy\_DragDrop(object sender, DragEventArgs e)

{

string[] files = e.Data.GetData(DataFormats.FileDrop) as string[];

if (files != null)

{

try

{

string filename = files[0];

string fileText = System.IO.File.ReadAllText(filename);

byte[] plainText = ByteConverter.GetBytes(fileText);

if (rsaActive)

{

encryptedFileText = crypto.RSAencryption(plainText, crypto.RSA.ExportParameters(false), false);

}

else if (md5Active)

{

char[] key = txtPrivateKey.Text.ToCharArray();

encryptedFileText = crypto.MD5encryption(plainText, key);

}

FileStream fs = new FileStream(encryptedFilename, FileMode.Create);

BinaryWriter bw = new BinaryWriter(fs);

bw.Write(encryptedFileText, 0, encryptedFileText.Length);

fs.Flush();

fs.Close();

}

catch { }

}

MessageBox.Show("encrypted");

}

void decryptFileBy\_DragDrop(object sender, DragEventArgs e)

{

string[] files = e.Data.GetData(DataFormats.FileDrop) as string[];

if (files != null)

{

try

{

string filename = files[0];

FileStream fs = new FileStream(filename, FileMode.Open);

BinaryReader br = new BinaryReader(fs);

byte[] encryptedFromFile = br.ReadBytes(Convert.ToInt32(fs.Length));

fs.Flush();

fs.Close();

if (rsaActive)

{

byte[] decrypted = crypto.RSAdecryption(encryptedFromFile, crypto.RSA.ExportParameters(true), false);

File.WriteAllText(decryptedFilename, System.Text.Encoding.UTF8.GetString(decrypted));

}

else if (md5Active)

{

char[] key = txtPrivateKey.Text.ToCharArray();

byte[] decrypted = crypto.MD5decryption(encryptedFromFile, key);

File.WriteAllText(decryptedFilename, System.Text.Encoding.UTF8.GetString(decrypted));

}

}

catch { }

}

MessageBox.Show("decrypted");

}

public Form()

{

InitializeComponent();

encryptFileButton.AllowDrop = true;

encryptFileButton.DragEnter += new DragEventHandler(showDragDropEffect\_DragEnter);

encryptFileButton.DragDrop += new DragEventHandler(encryptFileBy\_DragDrop);

decryptFileButton.AllowDrop = true;

decryptFileButton.DragEnter += new DragEventHandler(showDragDropEffect\_DragEnter);

decryptFileButton.DragDrop += new DragEventHandler(decryptFileBy\_DragDrop);

Icon = Icon.ExtractAssociatedIcon(System.Reflection.Assembly.GetExecutingAssembly().Location);

MinimumSize = new Size(650, 450);

MaximumSize = new Size(650, 450);

}

Crypto crypto = new Crypto();

UnicodeEncoding ByteConverter = new UnicodeEncoding();

byte[] encryptedText = { 0 };

byte[] encryptedFileText = { 0 };

bool rsaActive = true;

bool md5Active = false;

const string encryptedFilename = "D:\\projects\\VS19 projects\\cryptography\\crypto\\crypto\\encrypted.bin";

const string decryptedFilename = "D:\\projects\\VS19 projects\\cryptography\\crypto\\crypto\\decrypted.txt";

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

{

rsaActive = true;

md5Active = false;

currAlgorithmLabel.Text = "Current algorithm: RSA";

}

private void md5Button\_Click(object sender, EventArgs e)

{

md5Active = true;

rsaActive = false;

currAlgorithmLabel.Text = "Current algorithm: MD5";

}

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

{

try

{

byte[] plainText = ByteConverter.GetBytes(txtForEncryption.Text);

if (rsaActive)

{

encryptedText = crypto.RSAencryption(plainText, crypto.RSA.ExportParameters(false), false);

}

else if (md5Active)

{

char[] key = txtPrivateKey.Text.ToCharArray();

encryptedText = crypto.MD5encryption(plainText, key);

}

txtEncrypted.Text = ByteConverter.GetString(encryptedText);

}

catch { }

}

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

{

try

{

byte[] decryptedText = { 0 };

if (rsaActive)

{

decryptedText = crypto.RSAdecryption(encryptedText, crypto.RSA.ExportParameters(true), false);

}

else if (md5Active)

{

char[] key = txtPrivateKey.Text.ToCharArray();

decryptedText = crypto.MD5decryption(encryptedText, key);

}

txtDecrypted.Text = ByteConverter.GetString(decryptedText);

}

catch { }

}

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

{

try

{

if (openFileDialogEncrypt.ShowDialog() == DialogResult.Cancel)

{

return;

}

string filename = openFileDialogEncrypt.FileName;

string fileText = System.IO.File.ReadAllText(filename);

byte[] plainText = ByteConverter.GetBytes(fileText);

if (rsaActive)

{

encryptedFileText = crypto.RSAencryption(plainText, crypto.RSA.ExportParameters(false), false);

}

else if (md5Active)

{

char[] key = txtPrivateKey.Text.ToCharArray();

encryptedFileText = crypto.MD5encryption(plainText, key);

}

FileStream fs = new FileStream(encryptedFilename, FileMode.Create);

BinaryWriter bw = new BinaryWriter(fs);

bw.Write(encryptedFileText, 0, encryptedFileText.Length);

fs.Flush();

fs.Close();

}

catch { }

}

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

{

try

{

if (openFileDialogEncrypt.ShowDialog() == DialogResult.Cancel)

{

return;

}

string filename = openFileDialogEncrypt.FileName;

FileStream fs = new FileStream(filename, FileMode.Open);

BinaryReader br = new BinaryReader(fs);

byte[] encryptedFromFile = br.ReadBytes(Convert.ToInt32(fs.Length));

fs.Flush();

fs.Close();

if (rsaActive)

{

byte[] decrypted = crypto.RSAdecryption(encryptedFromFile, crypto.RSA.ExportParameters(true), false);

File.WriteAllText(decryptedFilename, System.Text.Encoding.UTF8.GetString(decrypted));

}

else if (md5Active)

{

char[] key = txtPrivateKey.Text.ToCharArray();

byte[] decrypted = crypto.MD5decryption(encryptedFromFile, key);

File.WriteAllText(decryptedFilename, System.Text.Encoding.UTF8.GetString(decrypted));

}

}

catch { }

}

}

}

**Код класса с криптографическими алгоритмами:**

using System;

using System.Text;

using System.Security.Cryptography;

using System.Numerics;

using System.Collections.Generic;

using System.IO;

using System.Windows.Forms;

using System.Diagnostics;

namespace crypto

{

class Crypto

{

private RSACryptoServiceProvider rSA = new RSACryptoServiceProvider();

private MD5CryptoServiceProvider mD5 = new MD5CryptoServiceProvider();

public MD5CryptoServiceProvider MD5 { get => mD5; set => mD5 = value; }

public RSACryptoServiceProvider RSA { get => rSA; set => rSA = value; }

Random rnd = new Random();

List<int> primes = GeneratePrimesNaive(20);

public byte[] RSAencryption(byte[] data, RSAParameters RSAKey, bool DoOAEPPadding)

{

try

{

byte[] encryptedData = { 0 };

using (RSACryptoServiceProvider RSA = new RSACryptoServiceProvider())

{

RSA.ImportParameters(RSAKey);

encryptedData = RSA.Encrypt(data, DoOAEPPadding);

}

return encryptedData;

}

catch (CryptographicException e)

{

Console.WriteLine(e.Message);

return null;

}

}

public byte[] RSAdecryption(byte[] data, RSAParameters RSAKey, bool DoOAEPPadding)

{

try

{

byte[] decryptedData = { 0 };

using (RSACryptoServiceProvider RSA = new RSACryptoServiceProvider())

{

RSA.ImportParameters(RSAKey);

decryptedData = RSA.Decrypt(data, DoOAEPPadding);

}

return decryptedData;

}

catch (CryptographicException e)

{

Console.WriteLine(e.ToString());

return null;

}

}

public byte[] MD5encryption(byte[] data, char[] privateKey)

{

using (var md5 = new MD5CryptoServiceProvider())

{

using (var tdes = new TripleDESCryptoServiceProvider())

{

tdes.Key = md5.ComputeHash(UTF8Encoding.UTF8.GetBytes(privateKey));

tdes.Mode = CipherMode.ECB;

tdes.Padding = PaddingMode.PKCS7;

using (var transform = tdes.CreateEncryptor())

{

return transform.TransformFinalBlock(data, 0, data.Length);

}

}

}

}

public byte[] MD5decryption(byte[] data, char[] privateKey)

{

using (var md5 = new MD5CryptoServiceProvider())

{

using (var tdes = new TripleDESCryptoServiceProvider())

{

tdes.Key = md5.ComputeHash(UTF8Encoding.UTF8.GetBytes(privateKey));

tdes.Mode = CipherMode.ECB;

tdes.Padding = PaddingMode.PKCS7;

using (var transform = tdes.CreateDecryptor())

{

return transform.TransformFinalBlock(data, 0, data.Length);

}

}

}

}

readonly char[] characters = new char[] {

'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',

'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z',

'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',

'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',

'1', '2', '3', '4', '5', '6', '7', '8', '9', '0',

' ', '\n', '\t' };

long d;

long n;

public void RSAEncrypt(string filename)

{

long p = primes[rnd.Next(0, primes.Count - 1)];

long q = primes[rnd.Next(0, primes.Count - 1)];

string s = "";

StreamReader sr = new StreamReader(filename);

while (!sr.EndOfStream)

{

s += sr.ReadLine();

}

sr.Close();

n = p \* q;

long m = (p - 1) \* (q - 1);

d = Calculate\_d(m);

long e\_ = Calculate\_e(d, m);

List<string> result = RSA\_Encode(s, e\_, n);

File.WriteAllLines("D:\\projects\\VS19 projects\\cryptography\\crypto\\crypto\\encrypted.txt", result);

}

public string RSADecrypt(string filename)

{

List<string> input = new List<string>();

StreamReader sr = new StreamReader(filename);

while (!sr.EndOfStream)

{

input.Add(sr.ReadLine());

}

sr.Close();

return RSA\_Decode(input, d, n);

}

private List<string> RSA\_Encode(string s, long e, long n)

{

List<string> result = new List<string>();

BigInteger bi;

for (int i = 0; i < s.Length; i++)

{

int index = Array.IndexOf(characters, s[i]);

bi = new BigInteger(index);

bi = BigInteger.Pow(bi, (int)e);

BigInteger n\_ = new BigInteger((int)n);

bi %= n\_;

result.Add(bi.ToString());

}

return result;

}

private string RSA\_Decode(List<string> input, long d, long n)

{

string result = "";

BigInteger bi;

foreach (string item in input)

{

bi = new BigInteger(Convert.ToDouble(item));

bi = BigInteger.Pow(bi, (int)d);

BigInteger n\_ = new BigInteger((int)n);

bi %= n\_;

int index = Convert.ToInt32(bi.ToString());

result += characters[index].ToString();

}

return result;

}

private long Calculate\_d(long m)

{

long d = m - 1;

for (long i = 2; i <= m; i++)

{

if ((m % i == 0) && (d % i == 0))

{

d--;

i = 1;

}

}

return d;

}

private long Calculate\_e(long d, long m)

{

long e = 10;

while (true)

{

if ((e \* d) % m == 1)

{

break;

}

else

{

e++;

}

}

return e;

}

public static List<int> GeneratePrimesNaive(int n)

{

List<int> primes = new List<int>();

primes.Add(2);

int nextPrime = 3;

while (primes.Count < n)

{

int sqrt = (int)Math.Sqrt(nextPrime);

bool isPrime = true;

for (int i = 0; (int)primes[i] <= sqrt; i++)

{

if (nextPrime % primes[i] == 0)

{

isPrime = false;

break;

}

}

if (isPrime)

{

primes.Add(nextPrime);

}

nextPrime += 2;

}

return primes;

}

}

}