МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ  
БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)  
Кафедра программного обеспечения информационных систем и технологий

**КУРСОВОЙ ПРОЕКТ**по дисциплине: “Компьютерные системы и сети”  
  
на тему: «***Сетевое ПО “Алгоритм шифрования Плейфера”»***

Выполнил**:** ст. гр. 10701320 Ковалёв Н. Д.

Приняла**:** ст**.** преподаватель Белова С. В.

Минск 2022

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных систем и технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту**

**по дисциплине «Компьютерные системы и сети»**

Тема: «***Сетевое ПО “Алгоритм шифрования Плейфера”»***

**Исполнитель**: Ковалёв Н. Д. (подпись)

**Ст. гр. 10701320**

**Руководитель**: Белова С. В. (подпись)

Минск 2022

**ОГЛАВЕНИЕ**

[ВВЕДЕНИЕ 6](#_Toc104414390)

[1. Описание и анализ предметной области 7](#_Toc104414391)

[2. Постановка задачи 8](#_Toc104414392)

[3. Проектирование программного обеспечения 9](#_Toc104414393)

[3.1 Диаграмма вариантов использования 9](#_Toc104414394)

[3.2 Архитектура программного обеспечения 9](#_Toc104414395)

[3.3 Описание протокола взаимодействия клиента и сервера. Структура передаваемых сообщений 11](#_Toc104414396)

[3.4 Проектирование интерфейса пользователя 11](#_Toc104414397)

[3.5 Модель данных 13](#_Toc104414398)

[3.6 Выбор средств разработки 13](#_Toc104414399)

[4. Реализация программного обеспечения 14](#_Toc104414400)

[4.1 Реализация серверной части 14](#_Toc104414401)

[4.2 Описание контрольного примера 17](#_Toc104414402)

[5. Руководство пользователя 19](#_Toc104414403)

[ЗАКЛЮЧЕНИЕ 21](#_Toc104414404)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 22](#_Toc104414405)

[ПРИЛОЖЕНИЕ A 23](#_Toc104414406)

[Графическая часть 23](#_Toc104414407)

[Графическая часть 24](#_Toc104414408)

[Графическая часть 25](#_Toc104414409)

[ПРИЛОЖЕНИЕ Б 26](#_Toc104414410)

[Листинг серверной части 26](#_Toc104414411)

[Листинг алгоритма шифрования 32](#_Toc104414412)

# ВВЕДЕНИЕ

Интернет – всемирная система объединённых компьютерных сетей для хранения и передачи информации. Такое определение дает нам Википедия – самая большая интернет-энциклопедия в мире.

Шифр Плейфера – ручная симметричная техника шифрования, в которой впервые использована замена биграмм. Изобретена в 1854 году Чарльзом Уистоном, но названа именем Лорда Лайона Плейфера, который внедрил данный шифр в государственные службы Великобритании. Шифр предусматривает шифрование пар символов (биграмм) вместо одиночных символов, как в шифре подстановки и в более сложных системах шифрования Виженера. Таким образом, шифр Плейфера более устойчив к взлому по сравнению с шифром простой замены, так как затрудняется частотный анализ. Он может быть проведен, но не для 26 возможных символов (латинский алфавит), а для 26x26=676 возможных биграмм. Анализ частоты биграмм возможен, но является значительно более трудным и требует намного больше объёма зашифрованного текста.

Цель курсового проекта – закрепление и углубление знаний, полученных при изучении курса «Компьютерные системы и сети» посредством разработки сетевого ПО “Алгоритм шифрования Плейфера”.

# 1. Описание и анализ предметной области

Компьютерная система – это совокупность аппаратных средств, управляемых программным обеспечением (операционной системой) как единый модуль. Компьютерная система может также предоставлять общие услуги, такие как управление доступом, взаимодействие процессоров и графический интерфейс пользователя.

Компьютерная сеть — это совокупность объектов, имеющих определенные общие признаки и определенным образом связанных между собой. Эта связь может быть непосредственной или опосредованной. Объекты объединяются в сети для экономии ресурсов. Компьютеры также могут образовывать сети. Компьютерная сеть — это совокупность компьютеров, объединенных каналами связи и обеспеченных коммуникационным оборудованием и программным обеспечением для совместного использования данных и оборудования.

Шифр Плейфера - ручная [симметричная](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D1%88%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) техника [шифрования](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), в которой впервые использована [замена биграмм](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D1%8B%D0%B9_%D1%88%D0%B8%D1%84%D1%80).

Для того чтобы зашифровать сообщение, необходимо разбить его на биграммы (группы из двух символов), например «Hello World» становится «HE LL OW OR LD», и отыскать эти биграммы в таблице. Два символа биграммы соответствуют углам прямоугольника в ключевой матрице. Определяем положения углов этого прямоугольника относительно друг друга. Затем, руководствуясь следующими 4 правилами, зашифровываем пары символов исходного текста:

1. Если два символа биграммы совпадают (или если остался один символ), добавляем после первого символа «Х», зашифровываем новую пару символов и продолжаем. В некоторых вариантах шифра Плейфера вместо «Х» используется «Q».
2. Если символы биграммы исходного текста встречаются в одной строке, то эти символы замещаются на символы, расположенные в ближайших столбцах справа от соответствующих символов. Если символ является последним в строке, то он заменяется на первый символ этой же строки.
3. Если символы биграммы исходного текста встречаются в одном столбце, то они преобразуются в символы того же столбца, находящиеся непосредственно под ними. Если символ является нижним в столбце, то он заменяется на первый символ этого же столбца.
4. Если символы биграммы исходного текста находятся в разных столбцах и разных строках, то они заменяются на символы, находящиеся в тех же строках, но соответствующие другим углам прямоугольника.

Для расшифровки необходимо использовать инверсию этих четырёх правил, откидывая символы «Х» (или «Q»), если они не несут смысла в исходном сообщении.

# 

# 2. Постановка задачи

В рамках данного курсового проекта необходимо реализовать сетевое приложение, в котором происходит шифрование и дешифрование сообщений с помощью шифра Плейфера.

Разрабатываемое приложение должно иметь архитектуру клиент-сервер, графический интерфейс для сервера и пользователя, а также многопользовательский режим.

Требования к реализации работы:

- операционная система – MS Windows 10;

- среда выполнения – Visual Studio 2022;

- язык программирования – С#;

- графический интерфейс;

- просмотр необходимой информации;

- просмотр руководства пользователя;

- просмотр информации о разработчиках;

- обратную связь с пользователем, например, вывод сообщений об ошибках, подключении пользователей и т. д.

Архитектурой программного обеспечения будет являться клиент-сервер, взаимодействие клиента и сервера будет организовано с помощью технологии протокола TCP, сервер будет поддерживать многопользовательский режим.

# 3. Проектирование программного обеспечения

## 3.1 Диаграмма вариантов использования

Архитектура программного обеспечения – описание структуры программной системы, включающее программные компоненты, их свойства и отношения между ними.

Программное обеспечение с хорошо продуманной архитектурой будет выполнять поставленные задачи, одновременно обеспечивая максимально высокую производительность, безопасность, надежность и многие другие факторы, такие, как лёгкость в поддержке, масштабируемости и т.д..

Сетевое приложение представляет собой распределенную программу, т. е. программу, которая состоит из нескольких взаимодействующих частей, каждая из которых выполняется на отдельном компьютере сети (рис. 3.1.1).

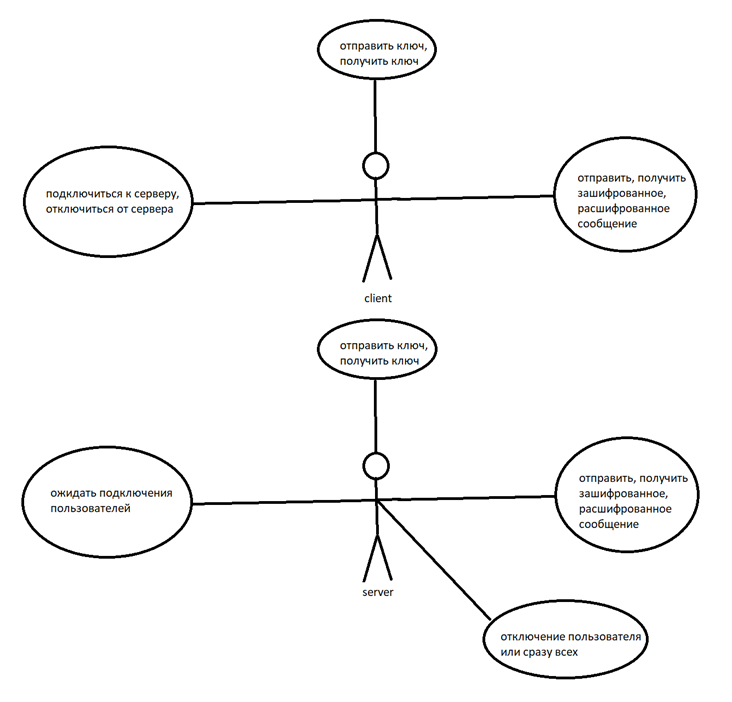


Рисунок 3.1.1 – Диаграмма вариантов использования приложения

## 

## 3.2 Архитектура программного обеспечения

Архитектура программного обеспечения – это структура программы или вычислительной системы, которая включает программные компоненты, видимые снаружи свойства этих компонентов, а также отношения между ними.

Программное обеспечение с хорошо продуманной архитектурой будет выполнять поставленные задачи, одновременно обеспечивая максимально высокую производительность, безопасность, надежность и многие другие факторы.

Сетевое приложение представляет собой распределенную программу, т. е. программу, которая состоит из нескольких взаимодействующих частей, каждая из которых выполняется на отдельном компьютере сети.

Взаимодействие частей распределённого приложения представлено на рисунке 3.2.1.

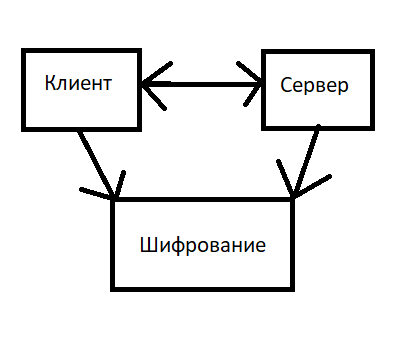


Рисунок 3.2.1— Взаимодействие частей распределённого приложения

Распределенная программа использует технологию клиент-сервер.

Технология "клиент-сервер" описывает взаимодействие между двумя компьютерами, согласно которому клиент запрашивает у сервера некоторые услуги, а сервер обслуживает запрос.

Чат с шифром Плейфера, разработанный в рамках текущего курсового проекта, реализует многопоточную модель взаимодействия.

В многопоточном сервере приложение работает в основном потоке, а для каждого подключившегося клиента создается новый поток. Поток существует все время взаимодействия с клиентом. При отключении клиента поток уничтожается.

В случае небольшого сервера и нескольких клиентов такой подход работает хорошо, к тому же его легко реализовать. К сожалению, многопоточный сервер плохо масштабируется. Главный его недостаток — большое число создаваемых и уничтожаемых потоков.

В данном проекте реализуется серверное приложение, которое будет обслуживать клиентов, которые подключаются к этому приложению по многопоточной модели, то есть для нового клиента новый поток.

Курсовая работа состоит из следующих модулей:

* Server;
* Client;

На рисунке 3.2.2 показано взаимодействие главных модулей.

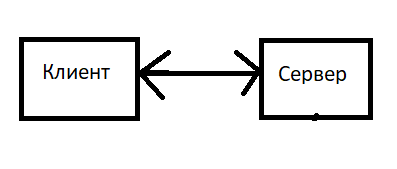


Рисунок 3.2.2 – Взаимодействие главных модулей

В данном проекте реализуется серверное приложение, которое будет обслуживать клиентов, которые подключаются к этому приложению по многопоточной модели, то есть для нового клиента новый поток.

При подключении клиента к серверу начинается обмен данными между ними на основе технологии протокола TCP.

## 3.3 Описание протокола взаимодействия клиента и сервера. Структура передаваемых сообщений

**TCP (протокол управления передачей)**— надежный, он обеспечивает передачу информации, проверяя дошла ли она, насколько полным является объем полученной информации и т.д. TCP дает возможность двум хостам производить обмен пакетами через установку соединения. Он предоставляет услугу для приложений, повторно запрашивает потерянную информацию, устраняет дублирующие пакеты, регулируя загруженность сети. TCP гарантирует получение и сборку информации у адресата в правильном порядке.

Поскольку необходимо разработать сетевое ПО «Алгоритм шифрования Плейфера», то структура передаваемого сообщения клиента будет выглядеть следующим образом:

-массив байт заданного размера;

## 3.4 Проектирование интерфейса пользователя

Интерфейс пользователя разрабатывался в среде Visual Studio 2022. Это интегрированная среда разработки для многих языков программирования, таких, как С#, C++, Javascript и др. Предоставляет средства для анализа кода, графический отладчик, инструмент для запуска юнит-тестов и поддерживает веб-разработку.

При открытии приложения появляется основная форма, в которой осуществляется подключение к серверу (рис. 3.3.1).

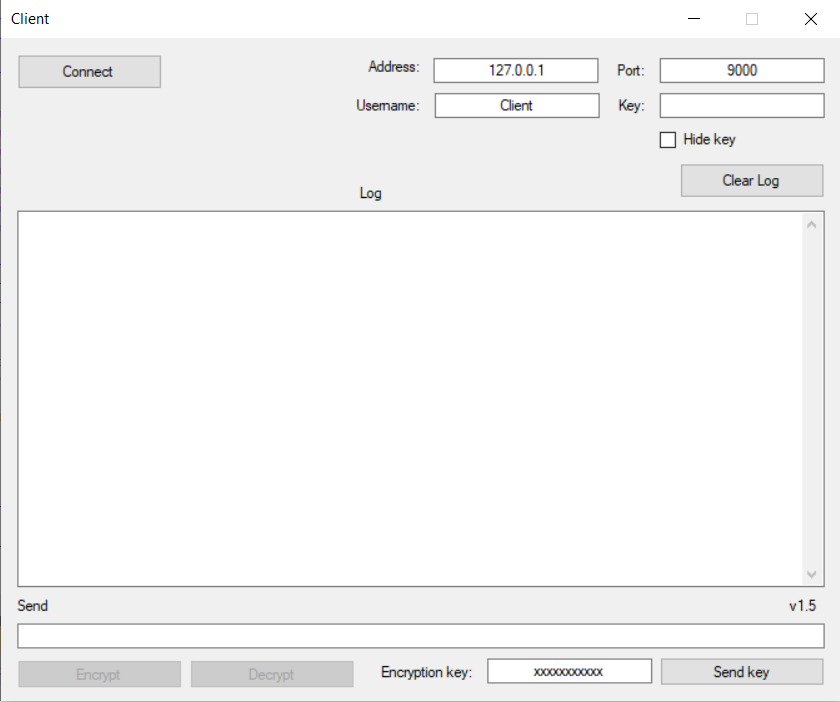


Рисунок 3.3.1 – Основной графический интерфейс до подключения к серверу

После подключения к чату имеем возможность переписываться с другими пользователями (рис. 3.3.2).

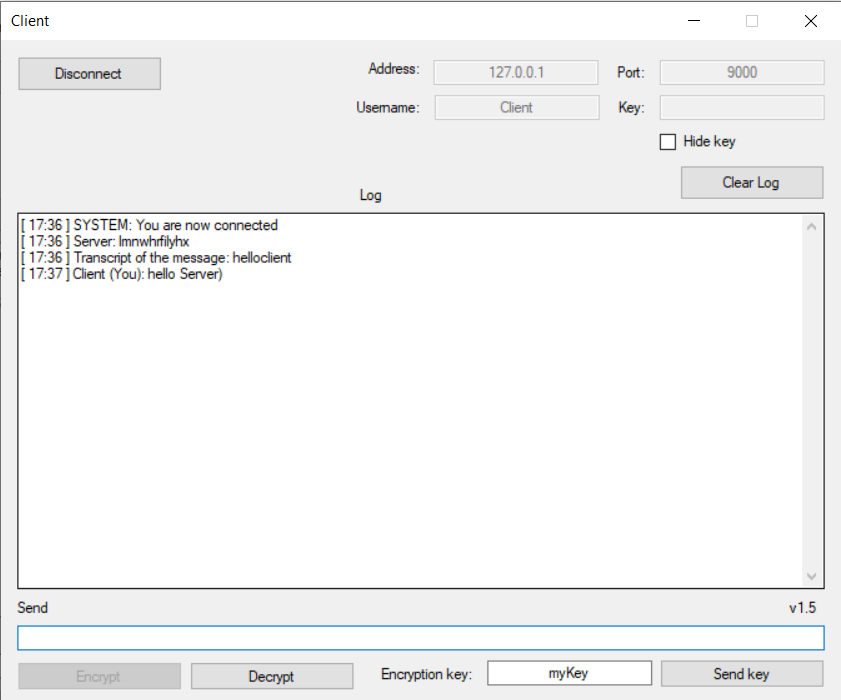


Рисунок 3.3.2 – Основной графический интерфейс после подключения к серверу

## 3.5 Модель данных

Модель данных — это абстрактное, самодостаточное, логическое определение объектов, операторов и прочих элементов, в совокупности составляющих абстрактную машину доступа к данным, с которой взаимодействует пользователь. Эти объекты позволяют моделировать структуру данных, а операторы — поведение данных.

В курсовой работе модель данных представлена структурой MyClient (рис.3.5.1).

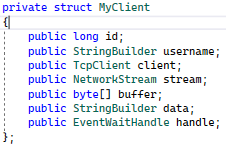


Рисунок 3.5.1 – Структура MyClient

## 3.6 Выбор средств разработки

Приложение будет разрабатываться под ОС Windows на языке программирования C# в среде разработке Visual Studio 2022. В приложение будет использоваться .Net Framework для создания Windows ориентированного ПО и технологии сокетов для обеспечения реализации клиент-серверной архитектуры приложения. Кроме того, для удобного взаимодействия частей программы будет использоваться объектно-ориентированная парадигма программирования.

# 4. Реализация программного обеспечения

## 4.1 Реализация серверной части

Поскольку в ходе курсового проекта необходимо было сделать клиент-серверное приложение, здесь будут описаны основные классы и функции, связанные с серверной частью.

Метод private void run() – используется дли инициализации начальных данных, таких как ip-адрес, номер порта, имя пользователя. А также запускает слушателя с помощью команды listener.start(). Листинг представлен на рисунке 4.1.1.

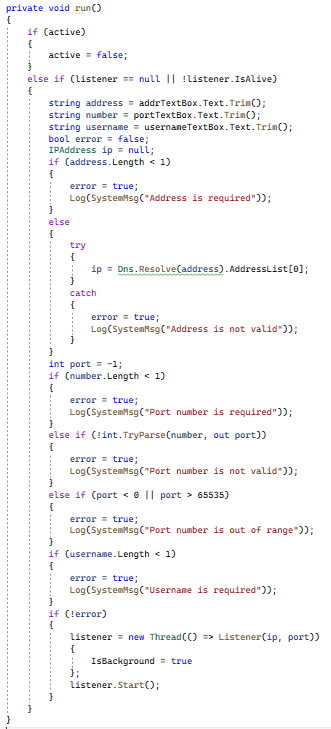


Рисунок 4.1.1 – Листинг функции run

Метод private void Listener(IPAddress ip, int port) – ожидает подключений. Если есть ожидающее подключение, инициализирует структуру MyClient данными. Листинг представлен на рисунке 4.1.2.



Рисунок 4.1.2 – Листинг функции Listener

Метод private void Read(IAsyncResult result) – считывает сообщение и логирует его в чат. Листинг представлен на рисунке 4.1.3.

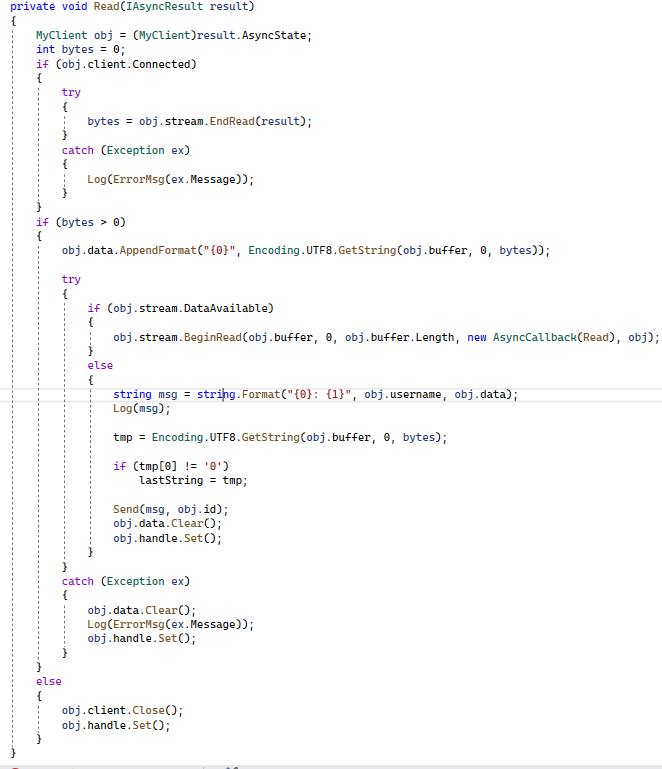


Рисунок 4.1.3 – Листинг функции Read

Метод private void Connection(MyClient obj) – в текущем потоке следит за подключением клиента. Добавляет пользователя в таблицу. Пока подключен к серверу - ждет сообщений. Листинг представлен на рисунке 4.1.4.

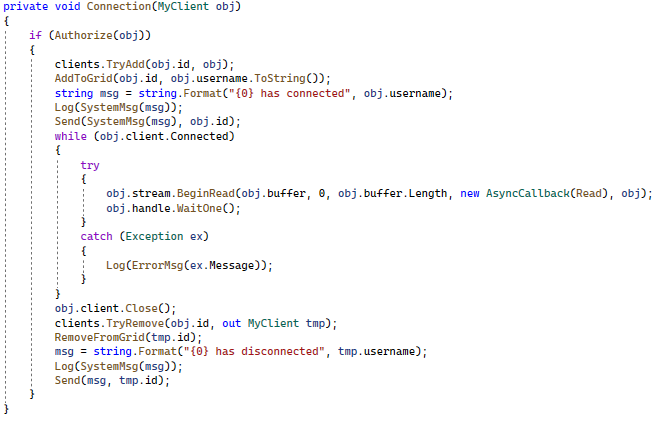


Рисунок 4.1.4 – Листинг функции Connection

Листинг оставшейся части программы находится в приложении Б.

## 4.2 Описание контрольного примера

Для начала работы запускаем сервер (рис. 4.2.1).



Рисунок 4.2.1 – Запуск сервера программы

Сервер ожидает подключений. Далее открываем окно клиента (рис. 4.2.2).

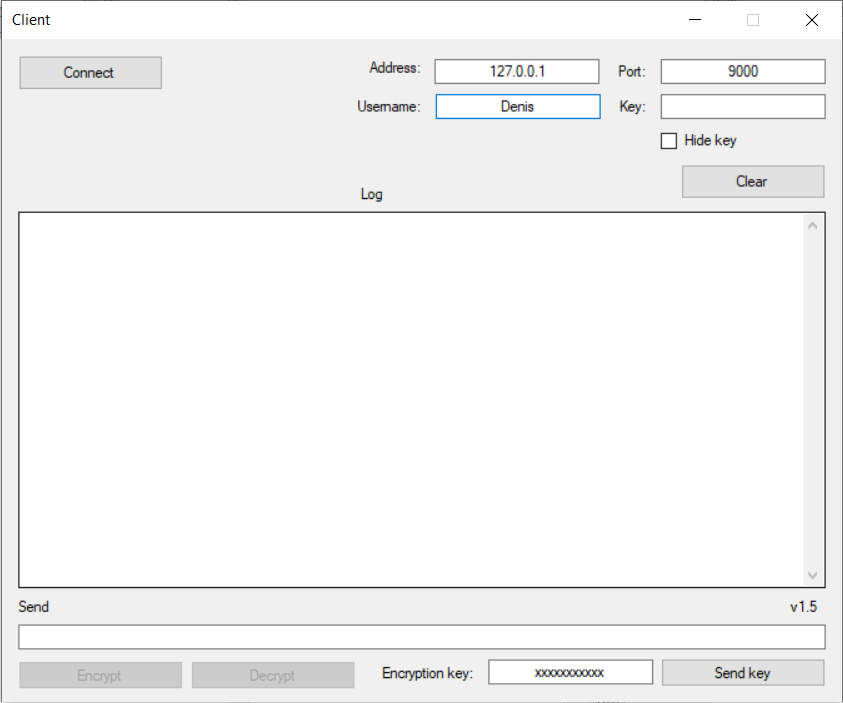


Рисунок 4.2.2 – Окно клиента

Нажимаем на кнопку Сonnect и подключаемся к серверу. Далее вводим сообщение, зашифровываем его (рис. 4.2.3).

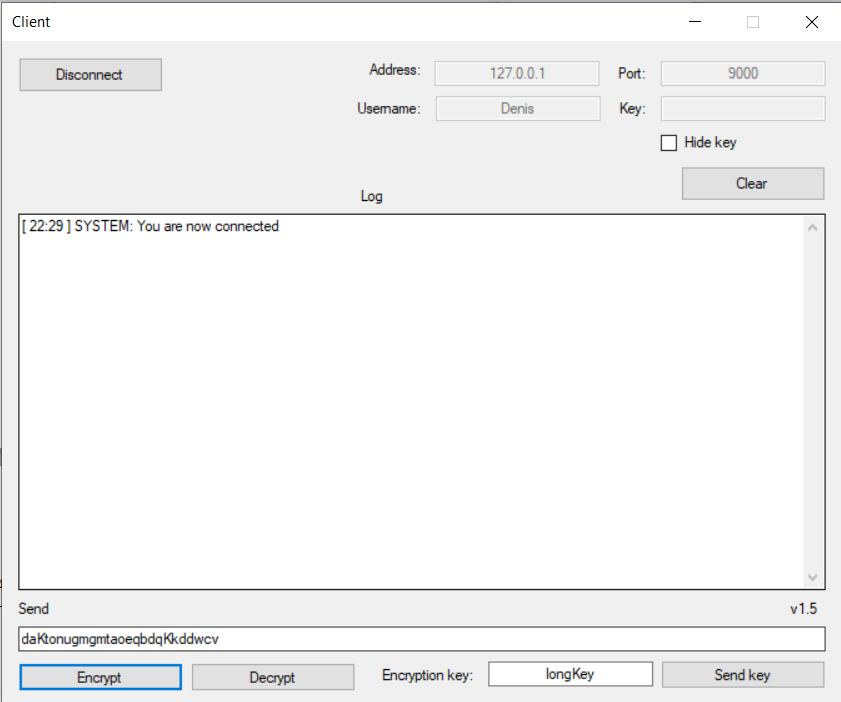


Рисунок 4.2.3 – Окно клиента

Отправляем сообщение и ключ, теперь с окна сервера можно его расшифровать (рис. 4.2.4).

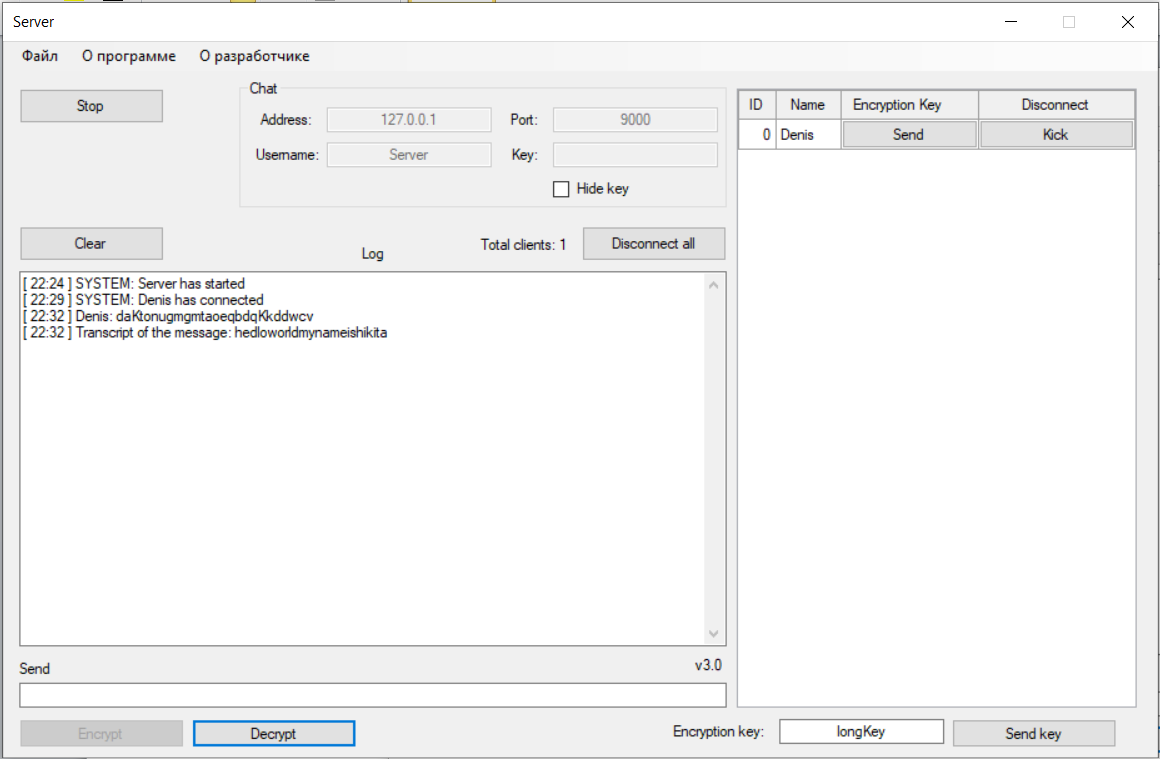


Рисунок 4.2.4 – Окно сервера

Сервер получил ключ и расшифровал сообщение.

Сервер также может шифровать сообщения и отправлять их. Также может отправлять ключ не всем, а определенному клиенту.

# 5. Руководство пользователя

Интерфейс приложения со вспомогательными надписями представлен на рисунке 5.1.

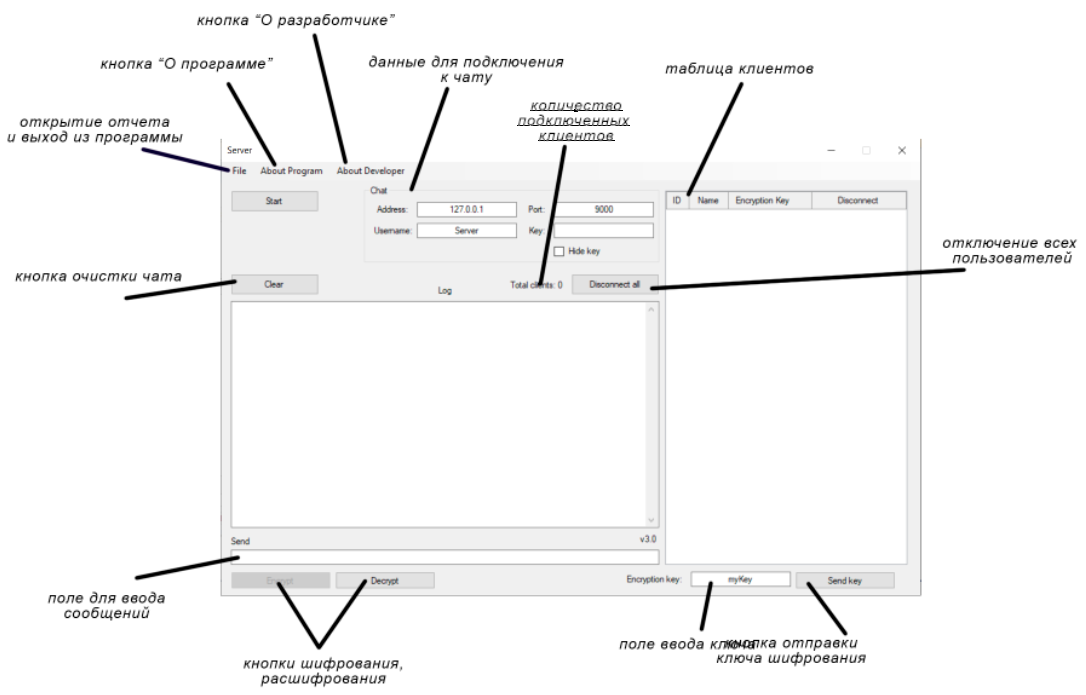


Рисунок 5.1 – Интерфейс приложения с аннотациями

Последовательность взаимодействия с приложением:

1. Запуск сервера и ожидание подключения клиентов;

2. Ввод сообщения и шифрование его с помощью ключа;

3. Отправка зашифрованного сообщения и ключа;

4. Расшифровка сообщения, если ключ передан.

Кроме того, можно воспользоваться кнопками “File”, “About Program ” и “About Developer” для открытия отчета, выхода из программы, получения кратких сведений о программе и разработчике.

При нажатии кнопки “About Program” выскакивает окно, представленное на рисунке 5.2, с информацией о версии, предназначении и разработчиках.

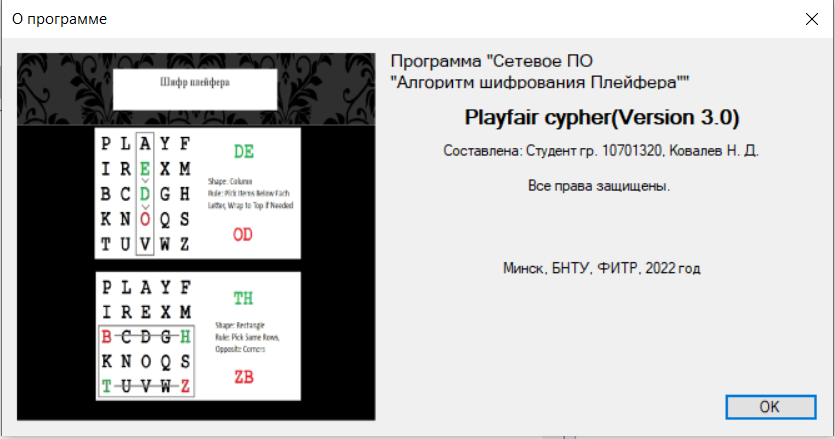


Рисунок 5.2 – Окно “About Program”

Если нажать на кнопку “About Developer”, то откроется окно с полной информацией о разработчике. Скриншот представлен на рисунке 5.3.

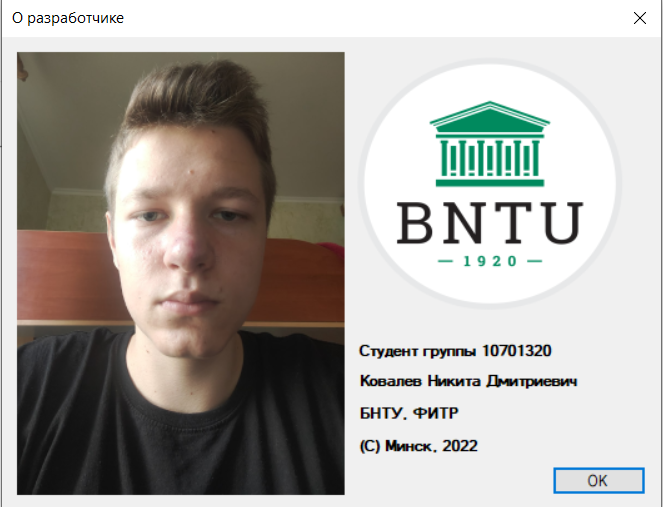


Рисунок 5.3 – Скриншот окна “About Developer”

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта:

* было разработано сетевое приложение с использованием шифра Плейфера;
* были углублены знания по дисциплине “Компьютерные системы и сети”;
* приобретен опыт разработки оконного программного обеспечения, выполняющего роль клиента;
* изучены возможности создания справочной системы высокой степени сложности и различных форматов.

Программа имеет достаточно ясный и понятый дружелюбный пользовательский интерфейс, обеспечивающий удобство в работе и получение необходимой пользователю информации с наименьшими затратами времени.

Таким образом, разработанное в ходе выполнения курсового проекта приложение является законченным программным продуктом.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Кумар В., Кровчик Э. и др. .NET. Сетевое программирование / Пер. с англ. –М.: «Лори», 2007

2. Библиотека MSDN [Электронный ресурс], Библиотека технической документации – Режим доступа: https://www.msdn.microsoft.com/library, свободный – Загл. с экрана. – Яз. рус., англ.

3. Справочная информация об ARP протоколе [Электронный ресурс], – Режим доступа: http://www.netpatch.ru/arch/misc-files/stievens/tcp-ip/glava4.html, свободный – Загл. с экрана. – Яз. рус.

4. Э. Таненбаум. Компьютерные сети изд. «Питер» 2012. – 960с.

# ПРИЛОЖЕНИЕ A

## Графическая часть

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | |
|  |  |  |  |  | КП—10701320 29–ДО–2022 | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | | Ковалёв |  |  | Диаграмма вариантов использования приложения | Лит | | | Лист | Листов |
| Руковод. | | Белова |  |  |  | Д |  | **1** | **3** |
| Консульт. | | Белова |  |  | 1 40 01 01  БНТУ, г. Минск | | | | |
| Н.контр. | | Белова |  |  |
| Зав.каф. | |  |  |  |

## Графическая часть

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | |
|  |  |  |  |  | КП—10701320 29–ДО–2022 | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | | Ковалёв |  |  | Архитектура приложения | Лит | | | Лист | Листов |
| Руковод. | | Белова |  |  |  | Д |  | **2** | **3** |
| Консульт. | | Белова |  |  | 1 40 01 01  БНТУ, г. Минск | | | | |
| Н.контр. | | Белова |  |  |
| Зав.каф. | |  |  |  |

## Графическая часть

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | |
|  |  |  |  |  | КП—10701320 29–ДО–2022 | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | | Ковалёв |  |  | Интерфейс сервера | Лит | | | Лист | Листов |
| Руковод. | | Белова |  |  |  | Д |  | **3** | **3** |
| Консульт. | | Белова |  |  | 1 40 01 01  БНТУ, г. Минск | | | | |
| Н.контр. | | Белова |  |  |
| Зав.каф. | |  |  |  |

# ПРИЛОЖЕНИЕ Б

## Листинг серверной части

**Файл Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Client

{

static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Client());

}

}

}

**Файл MainForm.cs**

using System;

using System.Collections.Concurrent;

using System.Collections.Generic;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Web.Script.Serialization;

using System.Windows.Forms;

using playfairСipher;

using Word = Microsoft.Office.Interop.Word;

namespace Server

{

public partial class MainForm : Form

{

private bool active = false;

private Thread listener = null;

private long id = 0;

private struct MyClient

{

public long id;

public StringBuilder username;

public TcpClient client;

public NetworkStream stream;

public byte[] buffer;

public StringBuilder data;

public EventWaitHandle handle;

};

private ConcurrentDictionary<long, MyClient> clients = new ConcurrentDictionary<long, MyClient>();

private Task send = null;

private Thread disconnect = null;

private bool exit = false;

public SecurityAlgorithm \_target;

public MainForm()

{

InitializeComponent();

}

/\* Логирование, если параметр не передается, то лог очищается. \*/

private void Log(string msg = "")

{

string[] tmp = msg.Split(':');

if (!exit)

{

logTextBox.Invoke((MethodInvoker)delegate

{

if (msg.Length > 0)

{

string key = tmp[1].Trim();

if ((key[0] == '0'))

{

encryptionKeyTextBox.Text = key.Substring(1);

}

else

{

logTextBox.AppendText(string.Format("[ {0} ] {1}{2}", DateTime.Now.ToString("HH:mm"),

msg, Environment.NewLine));

}

}

else

{

logTextBox.Clear();

}

});

}

}

/\* Возвращает сообщение ошибки. \*/

private string ErrorMsg(string msg)

{

return string.Format("ERROR: {0}", msg);

}

/\* Возвращает системное сообщение. \*/

private string SystemMsg(string msg)

{

return string.Format("SYSTEM: {0}", msg);

}

/\* Активация кнопок. \*/

private void Active(bool status)

{

if (!exit)

{

startButton.Invoke((MethodInvoker)delegate

{

active = status;

if (status)

{

addrTextBox.Enabled = false;

portTextBox.Enabled = false;

usernameTextBox.Enabled = false;

keyTextBox.Enabled = false;

startButton.Text = "Stop";

Log(SystemMsg("Server has started"));

}

else

{

addrTextBox.Enabled = true;

portTextBox.Enabled = true;

usernameTextBox.Enabled = true;

keyTextBox.Enabled = true;

startButton.Text = "Start";

Log(SystemMsg("Server has stopped"));

}

});

}

}

/\* Прослушивание авторизации. \*/

private void ReadAuth(IAsyncResult result)

{

MyClient obj = (MyClient)result.AsyncState;

int bytes = 0;

if (obj.client.Connected)

{

try

{

bytes = obj.stream.EndRead(result);

}

catch (Exception ex)

{

Log(ErrorMsg(ex.Message));

}

}

if (bytes > 0)

{

obj.data.AppendFormat("{0}", Encoding.UTF8.GetString(obj.buffer, 0, bytes));

try

{

if (obj.stream.DataAvailable)

{

obj.stream.BeginRead(obj.buffer, 0, obj.buffer.Length, new AsyncCallback(ReadAuth), obj);

}

else

{

JavaScriptSerializer json = new JavaScriptSerializer(); // feel free to use JSON serializer

Dictionary<string, string> data = json.Deserialize<Dictionary<string, string>>(obj.data.ToString());

if (!data.ContainsKey("username") || data["username"].Length < 1 || !data.ContainsKey("key") || !data["key"].Equals(keyTextBox.Text))

{

obj.client.Close();

}

else

{

obj.username.Append(data["username"].Length > 200 ? data["username"].Substring(0, 200) : data["username"]);

Send("{\"status\": \"authorized\"}", obj);

}

obj.data.Clear();

obj.handle.Set();

}

}

catch (Exception ex)

{

obj.data.Clear();

Log(ErrorMsg(ex.Message));

obj.handle.Set();

}

}

else

{

obj.client.Close();

obj.handle.Set();

}

}

/\* Авторизация пользователя. \*/

private bool Authorize(MyClient obj)

{

bool success = false;

while (obj.client.Connected)

{

try

{

obj.stream.BeginRead(obj.buffer, 0, obj.buffer.Length, new AsyncCallback(ReadAuth), obj);

obj.handle.WaitOne();

if (obj.username.Length > 0)

{

success = true;

break;

}

}

catch (Exception ex)

{

Log(ErrorMsg(ex.Message));

}

}

return success;

}

/\* Старт. \*/

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

{

run();

}

/\* Запись. \*/

private void Write(IAsyncResult result)

{

MyClient obj = (MyClient)result.AsyncState;

if (obj.client.Connected)

{

try

{

obj.stream.EndWrite(result);

}

catch (Exception ex)

{

Log(ErrorMsg(ex.Message));

}

}

}

/\* Отправить сообщение определенному клиенту. \*/

private void BeginWrite(string msg, MyClient obj)

{

byte[] buffer = Encoding.UTF8.GetBytes(msg);

if (obj.client.Connected)

{

try

{

obj.stream.BeginWrite(buffer, 0, buffer.Length, new AsyncCallback(Write), obj);

}

catch (Exception ex)

{

Log(ErrorMsg(ex.Message));

}

}

}

/\* Отправление сообщений. \*/

private void BeginWrite(string msg, long id = -1)

{

byte[] buffer = Encoding.UTF8.GetBytes(msg);

foreach (KeyValuePair<long, MyClient> obj in clients)

{

if (id != obj.Value.id && obj.Value.client.Connected)

{

try

{

obj.Value.stream.BeginWrite(buffer, 0, buffer.Length, new AsyncCallback(Write), obj.Value);

}

catch (Exception ex)

{

Log(ErrorMsg(ex.Message));

}

}

}

}

/\* Отправление сообщений. \*/

private void Send(string msg, MyClient obj)

{

if (send == null || send.IsCompleted)

{

send = Task.Factory.StartNew(() => BeginWrite(msg, obj));

}

else

{

send.ContinueWith(antecendent => BeginWrite(msg, obj));

}

}

/\* Отправление сообщений. \*/

private void Send(string msg, long id = -1)

{

if (send == null || send.IsCompleted)

{

send = Task.Factory.StartNew(() => BeginWrite(msg, id));

}

else

{

send.ContinueWith(antecendent => BeginWrite(msg, id));

}

}

/\* По нажатию клавиши Enter происходит отправка сообщения. \*/

private void SendTextBox\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Enter)

{

e.Handled = true;

e.SuppressKeyPress = true;

if (sendTextBox.Text.Length > 0)

{

string msg = sendTextBox.Text;

sendTextBox.Clear();

Log(string.Format("{0} (You): {1}", usernameTextBox.Text.Trim(), msg));

Send(string.Format("{0}: {1}", usernameTextBox.Text.Trim(), msg));

}

}

}

/\* Отключение пользователя или всех, если id не передается. \*/

private void Disconnect(long id = -1) // disconnect everyone if ID is not supplied or is lesser than zero

{

if (disconnect == null || !disconnect.IsAlive)

{

disconnect = new Thread(() =>

{

if (id >= 0)

{

clients.TryGetValue(id, out MyClient obj);

obj.client.Close();

RemoveFromGrid(obj.id);

}

else

{

foreach (KeyValuePair<long, MyClient> obj in clients)

{

obj.Value.client.Close();

RemoveFromGrid(obj.Value.id);

}

}

})

{

IsBackground = true

};

disconnect.Start();

}

}

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

{

Disconnect();

}

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

{

Log();

}

/\* Шифрование. \*/

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

{

//if (isRussianAlphabet(sendTextBox.Text))

\_target = new PlayFairEng(encryptionKeyTextBox.Text);

string actual = \_target.Encrypt(sendTextBox.Text);

sendTextBox.Text = actual;

}

/\* Дешифрование. \*/

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

{

if (lastString.Length > 0 && encryptionKeyTextBox.Text.Length > 0)

{

\_target = new PlayFairEng(encryptionKeyTextBox.Text);

string actual = \_target.Decrypt(lastString);

Log(string.Format("Transcript of the message: {0}", actual));

}

}

}

## Листинг алгоритма шифрования

**Файл SecurityAlgorithm.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace playfairСipher

{

public abstract class SecurityAlgorithm

{

//protected readonly Dictionary<char, int> alphabet;

public SecurityAlgorithm()

{

/\*alphabet = new dictionary<char, int>();

char c = 'a';

alphabet.add(c, 0);

for (int i = 1; i < 26; i++)

{

alphabet.add(++c, i);

}\*/

}

public abstract string Encrypt(string plainText); // шифрование

public abstract string Decrypt(string cipher); // дешифрование

}

}

**Файл PlayFairEng.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace playfairСipher

{

public class PlayFairEng : SecurityAlgorithm

{

protected readonly Dictionary<char, int> alphabet;

string key;

public PlayFairEng(string key)

{

alphabet = new Dictionary<char, int>();

char c = 'a';

alphabet.Add(c, 0);

for (int i = 1; i < 26; i++)

{

alphabet.Add(++c, i);

}

this.key = key;

}

#region Public Methods

public override string Encrypt(string plainText)

{

return Process(plainText, Mode.Encrypt);

}

public override string Decrypt(string cipherText)

{

return Process(cipherText, Mode.Decrypt);

}

#endregion

#region Private Methods

private string Process(string message, Mode mode)

{

// Ключ:Символ

// Значение:Позиция

Dictionary<char, string> characterPositionsInMatrix = new Dictionary<char, string>();

// Ключ:Символ

// Значение:Позиция

Dictionary<string, char> positionCharacterInMatrix = new Dictionary<string, char>();

FillMatrix(key.Distinct().ToArray(), characterPositionsInMatrix, positionCharacterInMatrix);

if (mode == Mode.Encrypt)

{

message = RepairWord(message);

}

string result = "";

for (int i = 0; i < message.Length; i += 2)

{

string substring\_of\_2 = message.Substring(i, 2).ToLower(); // получение символов из текста по парам

// получить строку и столбец каждого символа

string rc1 = characterPositionsInMatrix[substring\_of\_2[0]];

string rc2 = characterPositionsInMatrix[substring\_of\_2[1]];

if (rc1[0] == rc2[0]) // та же колонка, другая строка

{

int newC1 = 0, newC2 = 0;

switch (mode)

{

case Mode.Encrypt: // шифрование

newC1 = (int.Parse(rc1[1].ToString()) + 1) % 5;

newC2 = (int.Parse(rc2[1].ToString()) + 1) % 5;

break;

case Mode.Decrypt: // дешифрование

newC1 = (int.Parse(rc1[1].ToString()) - 1) % 5;

newC2 = (int.Parse(rc2[1].ToString()) - 1) % 5;

break;

}

newC1 = RepairNegative(newC1);

newC2 = RepairNegative(newC2);

result += positionCharacterInMatrix[rc1[0].ToString() + newC1.ToString()];

result += positionCharacterInMatrix[rc2[0].ToString() + newC2.ToString()];

}

else if (rc1[1] == rc2[1]) // та же колонка, другая строка

{

int newR1 = 0, newR2 = 0;

switch (mode)

{

case Mode.Encrypt: // шифрование

newR1 = (int.Parse(rc1[0].ToString()) + 1) % 5;

newR2 = (int.Parse(rc2[0].ToString()) + 1) % 5;

break;

case Mode.Decrypt: // дешифрование

newR1 = (int.Parse(rc1[0].ToString()) - 1) % 5;

newR2 = (int.Parse(rc2[0].ToString()) - 1) % 5;

break;

}

newR1 = RepairNegative(newR1);

newR2 = RepairNegative(newR2);

result += positionCharacterInMatrix[newR1.ToString() + rc1[1].ToString()];

result += positionCharacterInMatrix[newR2.ToString() + rc2[1].ToString()];

}

else // разные строки и столбцы

{

// 1-й символ: строка 1-го + столбец 2-го

// 2-ой символ: строка 2-го + столбец 1-го

result += positionCharacterInMatrix[rc1[0].ToString() + rc2[1].ToString()];

result += positionCharacterInMatrix[rc2[0].ToString() + rc1[1].ToString()];

}

}

// удаляет все x из строки при дешифровании

if (mode == Mode.Decrypt)

{

result = result.Replace("x", "");

}

return result;

}

private string RepairWord(string message)

{

string trimmed = message.Replace(" ", "");

string result = "";

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

{

result += trimmed[i];

if (i < trimmed.Length - 1 && message[i] == message[i + 1]) // проверить, являются ли две последовательные буквы одинаковыми

{

result += 'x';

}

}

if (result.Length % 2 != 0) // проверить, является ли длина четной

{

result += 'x';

}

return result.ToLower();

}

private void FillMatrix(IList<char> key, Dictionary<char, string> characterPositionsInMatrix, Dictionary<string, char> positionCharacterInMatrix)

{

char[,] matrix = new char[5, 5];

int keyPosition = 0, charPosition = 0;

List<char> alphabetPF = alphabet.Keys.ToList();

alphabetPF.Remove('j');

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

{

if (charPosition < key.Count)

{

matrix[i, j] = key[charPosition]; // заполнить матрицу ключом

alphabetPF.Remove(key[charPosition]);

charPosition++;

}

else // клавиша закончена... заполните остальной алфавит

{

matrix[i, j] = alphabetPF[keyPosition];

keyPosition++;

}

string position = i.ToString() + j.ToString();

// хранить позиции символов в словаре, чтобы не искать их каждый раз

characterPositionsInMatrix.Add(matrix[i, j], position);

positionCharacterInMatrix.Add(position, matrix[i, j]);

}

}

}

private int RepairNegative(int number)

{

if (number < 0)

{

number += 5;

}

return number;

}

#endregion

}

}

**Файл Enumerations.cs**

namespace playfairСipher

{

internal enum Mode

{

Encrypt, Decrypt

}

}