Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
Институт Вычислительной математики и информационных технологий

ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ

(Технологическая практика)

Обучающийся Десятов Александр Геннадьевич _ гр.09-641_ _____
(ФИО студента) (Группа) (Подпись)

Руководитель практики
от кафедры __доцент КСАИТ Андрианова А.А_._____

Оценка за практику _____ _____
(Подпись)

Дата сдачи отчета _____

Казань – 2020

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Во время учебной (технологической) практики изучены основные термины, понятия и технологии криптовалют. В ходе работы были получены сведения о роли криптографии в криптовалюте, были изучены транзакции существующих криптовалют, происходило ознакомление с блокчейном и схемами хранения на его основе. Разработанная система имеет распределенное хранение, она принимает решение с помощью консенсуса. Как и положено криптовалютной системе, она использует технологии майнинга.[1]

Проект выполнялся на объектно-ориентированном языке программирования C#, на данном языке код получается удобным для чтения, а также разработка имеет высокую скорость. Работа велась в команде из четырех человек. Созданная криптовалюта AKRAcoin получила название с помощью первых букв имен членов команды.

Проект выполнялся с помощью сервиса для совместной разработки GitHub. Каждый член команды, создавая модуль, выполнял свою задачу в своей ветке. На плечах автора данного отчета лежала разработка клиентского приложения для организации одноранговой сети.

После того, как все члены команды выполнили свои задачи, произошла интеграция разработанных модулей и появилась работоспособная система. В дальнейшем проводилось ее тестирование и исправление мелких недочетов.

ОСНОВНАЯ ЧАСТЬ

Сначала было произведено проектирование структуры пользовательского графического интерфейса. Для разработки была выбрана технология Windows Form для платформы .Net Framework 4.7.2.

В первую очередь была создана пользовательская форма с двумя вкладками: вход и регистрация. Внешний вид формы подходит для устройств с различным разрешением и размером экрана, размер элементов формы прямо пропорционален размеру самой формы. [2] Вкладка регистрации имеет текстовое поле для логина и кнопку самой регистрации. Функционал вкладки регистрации состоит из проверки введенного пользователем логина на уникальность, из оповещения пользователя о некорректности или занятости придуманного им логина (Рисунок 1).

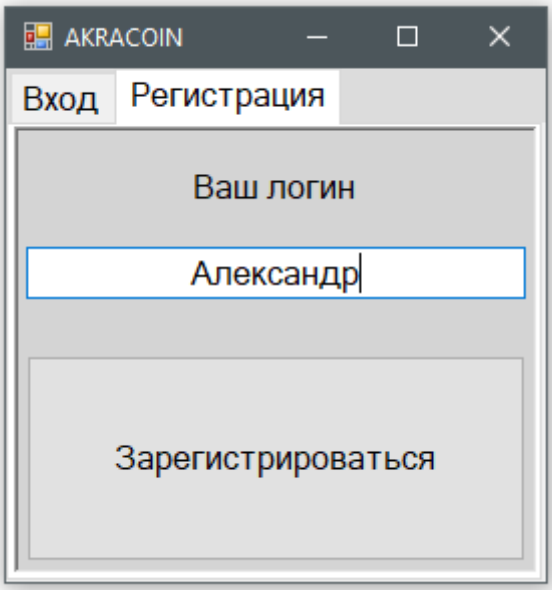

Рисунок 1. Вкладка регистрации

Вкладка входа также имеет поле для ввода логина и кнопку уже для входа. В случае, если пользователь неверно введет свой логин, он получит сообщение об этом. Стоит отметить, что для удобства пользователя, введенный логин при регистрации автоматически переходит в текстовое поле во кладке входа (Рисунок 2).

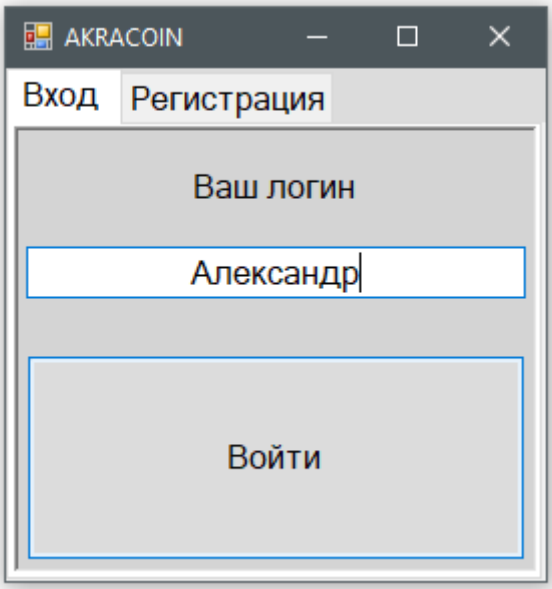

Рисунок 2. Вкладка входа

Генерация открытого и закрытого ключей происходит асинхронно с работой пользователя, чтобы к моменту нажатия на кнопку регистрации ключи уже были сгенерированы. После успешного входа пользователь переходит в рабочую форму (Рисунок 3).

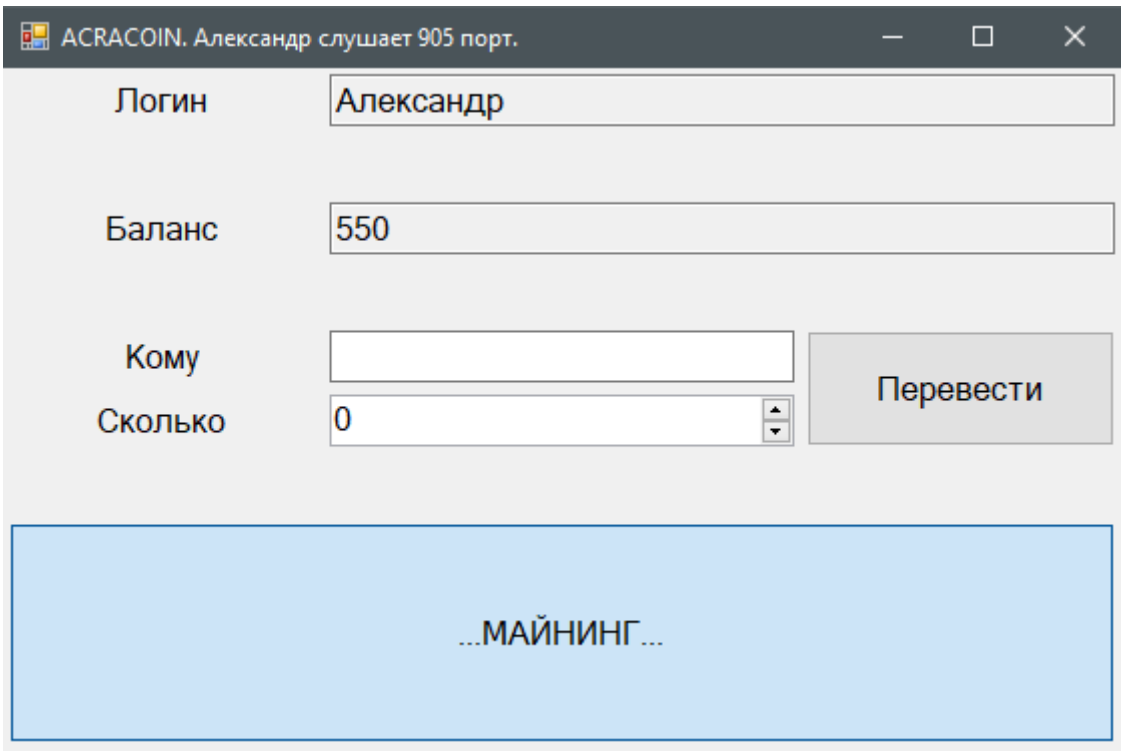

Рисунок 3. Рабочая форма

Рабочая форма пользователя также подходит под различные экраны. Ее размер пользователь может подстроить под себя, не уменьшая при этом ее функционал. Рабочая форма содержит следующую информацию: логин пользователя, его баланс, занятый порт на устройстве для получения сообщений. На форме имеется группа элементов для перевода средств другому пользователю: поле для ввода логина получателя, поле для суммы перевода и кнопка для перевода. А также на форме есть кнопка, позволяющая пользователю начать и закончить майнить.

Следует заметить, что пользователю вовсе не обязательно знать его ключи, поэтому он не наделяется ненужной для него информацией. Однако внутри программы логин и открытый ключ пользователей однозначно соответствуют друг другу.

При включении рабочей формы программа ищет свободный порт и занимает его для получения сокетов от других пользователей.

Обмен цепочками между пользователями происходит посредством сокетов протокола UDP. [3] Протокол выбран вследствие того, что в реальный момент времени не все пользователи настроены на получение сокетов.

Размер информации, который передается, может превышать размер буфера получателя, в связи с этим сначала получателю отправляется количество пакетов, которые придут в дальнейшем, и размер последнего пакета, а только потом по очереди присылаются пакеты.

Сообщения, которые передают пользователи друг другу, бывают трех типов.

Первый тип сообщения – первая цепочка нового зарегистрированного пользователя. Данное сообщение получают все пользователи, которые находятся в системе, и добавляют пришедшую цепочку к цепочкам, которые они хранили до этого.

Второй тип сообщения отправляется от пользователя, который решил перевести средства кому-либо. Его получают только те пользователи, у которых нажата кнопка майнинга. У них десериализуется полученная информация, и они начинают майнить в отдельном потоке, чтобы программа не теряла свою работоспособность. После необходимой обработки информации программы майнеров отправляют третий тип сообщения, состоящий из новых получившихся цепочек.

Третий тип сообщения получают все пользователи, находящиеся в системе, в том числе тот, кто его послал. Программы получателей обрабатывают полученные цепочки, сравнивают их с хранящимися у них цепочками и в итоге оставляют результирующие цепочки у себя. Майнер получает вознаграждение, перевод АКРАкоинов оказывается успешным.

После обновления цепочек у пользователей вызывается функция обновления баланса.

Разработанная система стала полностью работоспособной после интеграции dll-библиотек, созданных членами команды. Функции из их модулей используются практически на каждом шаге.

ЗАКЛЮЧЕНИЕ

В ходе создания системы криптовалюты были выполнены следующие задачи:

1) Изучен основной, а затем и углубленный теоретический материал о технологиях криптовалют.
2) Спроектирована структура новой системы криптовалюты.
3) Распределены обязанности внутри команды.
4) Создан репозиторий на сервисе GitHub для совместной работы.
5) Созданы ветки каждого члена команды.
6) Выполнены персональные обязанности каждого члена группы.
7) Произведена интеграция модулей всех членов группы.
8) Создана новая система криптовалюты.
9) Проведено тестирование системы.
10) Получен опыт командной разработки при создании проекта.

За время практики осваивались следующие компетенции:

Код	Расшифровка компетенции	Расшифровка освоения
ОПК-4	способность понимать значение информации в развитии современного общества, применять информационные технологии для поиска и обработки информации	- знание значения криптовалют в современном информационном обществе; - знание архитектуры реализации систем криптовалют.
ОПК-7	способность определять информационные ресурсы, подлежащие защите, угрозы безопасности информации и возможные пути их реализации на основе анализа структуры и содержания информационных процессов и особенностей	- знание основных угроз системам криптовалют и основных способов их устранения.

	функционирования объекта защиты	
ПК-1	способность выполнять работы по установке, настройке и обслуживанию программных, программно-аппаратных (в том числе криптографических) и технических средств защиты информации	- знание принципов функционирования подсистем реализации криптовалют, включая компоненты, которые обеспечивают защиту информации
ПК-2	способность применять программные средства системного, прикладного и специального назначения, инструментальные средства, языки и системы программирования для решения профессиональных задач	- умение программным образом реализовывать различные подсистемы для функционирования подсистемы криптовалют
ПК-3	способность администрировать подсистемы информационной безопасности объекта защиты	- умение производить настройку всех компонентов системы криптовалют, включая компоненты криптографии

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) Курс «Технологии криптовалют» // портал «intuit». – URL: https://www.intuit.ru/studies/courses/3643/885/info [Дата обращения: 11.10.19]

2) «TableLayoutPanel Класс» // портал «docs.microsoft». – URL: https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.tablelayoutpanel?view=netframework-4.8 [Дата обращения: 22.11.19]

3) «Сокеты» // портал «professorweb». – URL: https://professorweb.ru/my/csharp/web/level3/3_1.php [Дата обращения: 13.12.19]

ПРИЛОЖЕНИЕ

Код для формы входа и регистрации.

```
using System;
using System.ComponentModel;
using System.IO;
using System.Net.Sockets;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

using Cryptography;

using StorageLib;

namespace @interface
{
  public partial class Form_Login_Registration : System.Windows.Forms.Form
  {
    Socket socket_sending;

    string publicKey_new = "";
    string secretKey_new = "";
    string mod_new = "";
    // В этом файле логины и ключи
    string path_by_login_public_secret_mod = Environment.CurrentDirectory +
@"\login_public_secret_mod.txt";
    // В этом файле логины, ip, порты
    string path_by_login_ip_port = Environment.CurrentDirectory +
@"\login_ip_port.txt";
    // Файлы пользователя
    string path_to_user_files = Environment.CurrentDirectory; // Когда будет
создан логин, будет более точный путь

    // Длина буфера приема сообщений
    int buffer_length_for_getting = 1024;

    public Form_Login_Registration()
    {
      InitializeComponent();
    }

    private void Form_Login_Registration_Shown(object sender, EventArgs e)
    {
      socket_sending = new Socket(AddressFamily.InterNetwork,
SocketType.Dgram, ProtocolType.Udp);

      // Запуск потока для формирования ключей
      bW_KeysGeneration.RunWorkerAsync();

      // Поставить курсор в поле ввода логина
      tB_Login_inLog.Select();
    }
    // Поток генерации ключей
    private void bW_KeysGeneration_DoWork(object sender, DoWorkEventArgs e)
    {
      try
      {
        while (true)
        {
          // Формирование ключей
          var keys = new Crypto().GenerateKeys();
```

```
            // Открытый ключ
            publicKey_new = keys[0];
            // Закрытый ключ
            secretKey_new = keys[1];
            // Mod
            mod_new = keys[2];

            // Проверка на уникальность
            bool exist_public = false;
            using (FileStream fr = new
FileStream(path_by_login_public_secret_mod, FileMode.OpenOrCreate,
FileAccess.Read))
            {
              using (StreamReader sr = new StreamReader(fr,
Encoding.Default))
              {
                string line;
                while ((line = sr.ReadLine()) != null)
                {
                  var login_public_secret_mod = line.Split();
                  if (login_public_secret_mod[1] == publicKey_new)
                  {
                    exist_public = true;
                    break; // То есть заново сформировать
                  }
                }
              }
            }
            if(!exist_public)
              break;
          }
        }
        catch(Exception ex)
        {
          MessageBox.Show(ex.ToString() + "\n" + ex.Message);
        }
      }
      // Переключение вкладки
      private void tabControl_Login_Registration_SelectedIndexChanged(object
sender, EventArgs e)
      {
        TabControl tc = (TabControl)sender;
        // Поставить курсор в поле для ввода в конце текста
        if (tc.SelectedTab.Text == "Регистрация")
        {
          tB_Login_inReg.Select();
        }
        else if (tc.SelectedTab.Text == "Вход")
        {
          tB_Login_inLog.Select();
        }
      }
      private void btn_Registration_Click(object sender, EventArgs e)
      {
        string login = tB_Login_inReg.Text;
        if (login.Equals(""))
        {
          MessageBox.Show("Пожалуйста, введите логин :)");
          return;
        }

        if (login.IndexOf(" ") != -1)
        {
```

```
            MessageBox.Show("Логин не должен содержать пробел :(");
            return;
        }

        if (publicKey_new.Equals("") || secretKey_new.Equals("") ||
mod_new.Equals(""))
        {
            MessageBox.Show("Еще не сформированы ключи :(\nПожалуйста,
подождите!");
            return;
        }

        // Проверка на уникальность логина и открытого ключа
        using (FileStream fr = new FileStream(path_by_login_public_secret_mod,
FileMode.OpenOrCreate, FileAccess.Read))
        {
            using (StreamReader sr = new StreamReader(fr, Encoding.Default))
            {
                string line;
                while ((line = sr.ReadLine()) != null)
                {
                    var login_public_secret_mod = line.Split();
                    if (login_public_secret_mod[0] == login)
                    {
                        MessageBox.Show("Такой логин уже занят :(");
                        return;
                    }
                    if(login_public_secret_mod[1] == publicKey_new)
                    {
                        publicKey_new = "";
                        secretKey_new = "";
                        mod_new = "";
                        // Запуск потока для формирования ключей
                        bW_KeysGeneration.RunWorkerAsync();

                        MessageBox.Show("Открытый ключ заняли :(\nНажмите еще раз на
кнопку");
                        return;
                    }
                }
            }
        }

        // Отправка в центр сертификации (логин, открытый ключ, закрытый ключ,
mod)
        using (StreamWriter sw = new
StreamWriter(path_by_login_public_secret_mod, true, Encoding.Default))
        {
            sw.WriteLine(login + " " + publicKey_new + " " + secretKey_new + " "
+ mod_new);
        }

        // Проверяем есть ли уже папка этого пользователя и создаем, если нет
        path_to_user_files = Environment.CurrentDirectory + "\\" + login;
        Directory.CreateDirectory(path_to_user_files);

        // Создание генезиса
        Chain ch = new Chain();
        byte[] first_chain = ch.CreateChain(publicKey_new, 0);

        common helper = new common();

        // Запись цепочки в файл
```

```
        helper.WriteChainToFile(path_to_user_files, first_chain);

        // Запуск потока отправки цепочки всем, кроме себя (тип сообщения 1)
        byte TypeMessage = 1;
        Task.Run(() => helper.SendSocket(socket_sending,
                    helper.getEndPoints(path_by_login_ip_port), // Сам пока ни
на каком порту не ждет
                    TypeMessage,
                    first_chain,
                    buffer_length_for_getting));

        MessageBox.Show("Вы успешно зарегистрированы!\nПожалуйста, запомните
свой логин: " + login);
        // Переход на вкладку Входа
        tabControl_Login_Registration.SelectedTab = tabPage_Login;
// В поле текста для входа сразу вводится логин от регистрации, чтобы
пользователю не вводить второй раз
        tB_Login_inLog.Text = login;

        // На тот случай, если кто-то еще захочет зарегистрироваться
        tB_Login_inReg.Text = "";
        publicKey_new = "";
        secretKey_new = "";
        mod_new = "";
        // Новая генерация ключа
        bW_KeysGeneration.RunWorkerAsync();
    }

    private void btn_Login_Click(object sender, EventArgs e)
    {
        string login = tB_Login_inLog.Text;
        // Есть ли такой пользователь
        bool exist = false;
        using (FileStream fr = new FileStream(path_by_login_public_secret_mod,
FileMode.OpenOrCreate))
        {
            using (StreamReader sr = new StreamReader(fr, Encoding.Default))
            {
                string line;
                while ((line = sr.ReadLine()) != null)
                {
                    var login_public_secret_mod = line.Split();
                    if (login_public_secret_mod[0] == login)
                    {
                        exist = true;
                        break;
                    }
                }
            }
        }
        if (exist)
        {
            //// Открытие рабочей формы (Нужно передать логин)
            Form_Work form_Work = new Form_Work(login, socket_sending);
            // После закрытия рабочей формы закрыть основную
            form_Work.FormClosed += (s, ev) => Close();
            // Показать рабочую форму
            form_Work.Show();
            Hide();
        }
        else
        {
            MessageBox.Show("Пользователь с логином " + login + " не найден!");
```

```
            // Ввод этого логина в поле текста для регистрации
            tB_Login_inReg.Text = login;
        }
      }

      private void Form_Login_Registration_FormClosing(object sender,
FormClosingEventArgs e)
      {
        // Выключение потока, если он запущен
        if (bW_KeysGeneration.IsBusy)
        {
          bW_KeysGeneration.WorkerSupportsCancellation = true;
          bW_KeysGeneration.CancelAsync();
        }
      }
    }
}
```

Код для рабочей формы.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Numerics;
using System.Runtime.Serialization.Formatters.Binary;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using System.Windows.Forms;
using MiningTask;
using StorageLib;
namespace @interface
{
  public partial class Form_Work : Form
  {
    Socket socket_getting;
    int port_getting = 0;
    string ip_getting = "";
    Socket socket_sending;
    string login;
    string publicKey = "";
    string secretKey = "";
    string mod = "";
    // В этом файле логины и ключи
    string path_by_login_public_secret_mod = Environment.CurrentDirectory +
@"\login_public_secret_mod.txt";
    // В этом файле логины, ip, порты
    string path_by_login_ip_port = Environment.CurrentDirectory +
@"\login_ip_port.txt";
    // Файлы пользователя
    string path_to_user_files; // будет более точный путь с логином
    // Длина буфера приема сообщений (в двух формах задается значение)
    int buffer_length_for_getting = 1024;
    public Form_Work(string login_f, Socket socket_sending_f)
    {
      InitializeComponent();
      socket_sending = socket_sending_f;
      login = login_f;
      path_to_user_files = Environment.CurrentDirectory + "\\" + login;
      // Получение всех ключей и запись в переменную
```

```
        using (FileStream fr = new FileStream(path_by_login_public_secret_mod,
FileMode.OpenOrCreate, FileAccess.Read))
        {
          using (StreamReader sr = new StreamReader(fr, Encoding.Default))
          {
            string line;
            while ((line = sr.ReadLine()) != null)
            {
              var login_public_secret_mod = line.Split();
              if (login_public_secret_mod[0] == login)
              {
                publicKey = login_public_secret_mod[1];
                secretKey = login_public_secret_mod[2];
                mod = login_public_secret_mod[3];
                break;
              }
            }
          }
        }
      }

      private void Form_Work_Load(object sender, EventArgs e)
      {
        socket_getting = new Socket(AddressFamily.InterNetwork,
SocketType.Dgram, ProtocolType.Udp);
        //socket_sending = new Socket(AddressFamily.InterNetwork,
SocketType.Dgram, ProtocolType.Udp); // Создан в окне входа-регистрации

        tB_login.Text = login;
        updateBalance();

        Task listeningTask = new Task(GetSocket);
        // Запуск потока прослушивания
        listeningTask.Start();

        // Курсор в текстовое поле для ввода пользователю, которому следует
отправить
        tB_ToUser.Select();
      }
      // Обновление баланса
      private void updateBalance()
      {
        TMoney money = new TMoney();
        // Все цепочки
        List<byte[]> all_chains = GetAllChains();
        BigInteger balance = money.SumMoney(all_chains, publicKey);
        tB_balance.Text = balance.ToString();
      }
      // Получить все цепочки из файлов
      private List<byte[]> GetAllChains()
      {
        List<byte[]> all_chains = new List<byte[]>();
        // по всем бинарным файлам внутри папки
        FileInfo[] files = new DirectoryInfo(path_to_user_files).GetFiles();
        foreach (var file in files)
        {
          byte[] bytes_f = File.ReadAllBytes(file.FullName);
          all_chains.Add(bytes_f);
        }
        return all_chains;
      }
      private void Form_Work_FormClosing(object sender, FormClosingEventArgs e)
      {
```

```
        if (socket_getting != null)
        {
          socket_getting.Shutdown(SocketShutdown.Both);
          socket_getting.Close();
          socket_getting = null;
        }
        if (socket_sending != null)
        {
          socket_sending.Shutdown(SocketShutdown.Both);
          socket_sending.Close();
          socket_sending = null;
        }
        if(bW_Mining.IsBusy)
        {
          bW_Mining.WorkerSupportsCancellation = true;
          bW_Mining.CancelAsync();
        }
      }
      // поток для приема подключений
      private void GetSocket()
      {
        #region Подготовка к принятию сокета
        // Стартовая позиция для поиска свободного порта
        port_getting = 905;
        // столько портов проверится, начиная с port_getting, до первого
найденного свободного
        int attempts = 100;
        // Поиск свободного порта
        while ((attempts--) > 0)
        {
          try
          {
            // Получение ip-адреса.
              ip_getting = "127.0.0.1";
            // IPEndPoint localIP = new IPEndPoint(IPAddress.Any,
port_getting);

            IPEndPoint iPendPoint = new
IPEndPoint(IPAddress.Parse(ip_getting), port_getting);
            socket_getting.Bind(iPendPoint);
            Text += ". " + login + " слушает " + port_getting + " порт.";
            // Обновление login_ip_port
            using (FileStream fr = new FileStream(path_by_login_ip_port,
FileMode.OpenOrCreate))
            {
              // Все строчки файла
              string all_login_ip_port = string.Empty;
              using (StreamReader sr = new StreamReader(fr,
Encoding.Default))
              {
                all_login_ip_port = sr.ReadToEnd();
              }

              // Новая строка login_ip_port
              string newLine_login_ip_port = login + " " + ip_getting + " " +
port_getting;
              // Поиск старой строки
              Match match = Regex.Match(all_login_ip_port, login + ".*\n");
        // Если логин есть в файле, то старая строчка убирается
              if(!match.Value.Equals(""))
                all_login_ip_port = all_login_ip_port.Replace(match.Value,
"");
        // Заново записывается файл целиком и новая строка в конце
```

```
            using (StreamWriter sw = new
StreamWriter(path_by_login_ip_port, false, Encoding.Default))
            {
              sw.Write(all_login_ip_port);
              sw.WriteLine(newLine_login_ip_port);
            }
          }
          break;
        }
        catch (SocketException)
        {// порт занят
          port_getting++;
          continue;
        }
        catch (Exception ex)
        {// другое исключение
          rTB_temp2.Text += "\n" + ex.Message;
          return;
        }
      }
      #endregion
      // В этом цикле получение сокетов
      while (true)
      {
        // буфер для получаемых данных
        byte[] buffer = new byte[buffer_length_for_getting];
        //адрес, с которого пришли данные
        EndPoint remoteIp = new IPEndPoint(IPAddress.Any, 0);

        // Получение числа пакетов сообщения
        socket_getting.ReceiveFrom(buffer, ref remoteIp);
        int count_packets = BitConverter.ToInt32(buffer, 0);

        // Получение длины последнего пакета сообщения
        socket_getting.ReceiveFrom(buffer, ref remoteIp);
        int length_last = BitConverter.ToInt32(buffer, 0);

        // Все полученные байты в одной переменной
        List<byte> data = new List<byte>();
        for(int i=0; i<count_packets-1; i++)
        {
          socket_getting.ReceiveFrom(buffer, ref remoteIp);
          data.AddRange(buffer);
        }
        // Получение последнего пакета
        socket_getting.ReceiveFrom(buffer, ref remoteIp);
        data.AddRange(buffer.ToList().GetRange(0, length_last));

        // Обрабатываем полностью полученное сообщение
        byte typeMessage = data[0];
        data.RemoveAt(0); // Убрать из данных тип сообщения

        switch (typeMessage)
        {
          case 1:
            rTB_temp1.Text += "\nПришло сообщение (тип " + typeMessage +
")байт: " + data.Count;
            // Пришла одна цепочка (при регистрации всем отправилось)
            // Запись одной цепочки в файл
            new common().WriteChainToFile(path_to_user_files,
data.ToArray());
            break;
          case 2:
```

```
            // Если майнит
            if (cB_mining.Checked)
            {
              rTB_temp2.Text += "\nПришло сообщение (тип " + typeMessage +
")байт: " + data.Count;
           // После того, как кто-то перевел деньги
           // Десериализация byte[] -> List<List<byte[]>>
              List<List<byte[]>> ch_and_trans =
deSerial_list_list_bytes(data.ToArray());

       // В отдельном потоке, так как будет задержка:
       bW_Mining.RunWorkerAsync(ch_and_trans);
            }
            break;
          case 3:
            rTB_temp3.Text += "\nПришло сообщение (тип " + typeMessage +
")байт: " + data.Count;
            // От майнеров получают все
            // Десериализация в
            List<byte[]> new_chains = deSerial_list_bytes(data.ToArray());

            // Взять все свои цепочки
            List<byte[]> myChains = GetAllChains();

            Chain chain = new Chain();
            foreach(var ch in new_chains)
            {
              // Получение обновленной старой цепочки
              myChains = chain.WhIslonger(myChains, ch);
            }
        // Старые файлы-цепочки удалить вместе с папкой
          Directory.Delete(path_to_user_files, true);
            // Записать каждую цепочку в свои файлы
            common helper = new common();
            foreach(var ch in myChains)
              helper.WriteChainToFile(path_to_user_files, ch.ToArray());

            break;
          default:
            break;
        }
        updateBalance();
      }
    }
    // Функция майнинга в потоке
    private void bW_Mining_DoWork(object sender, DoWorkEventArgs e)
    {
      List<List<byte[]>> ch_and_tr = (List < List<byte[]> >) e.Argument;
      CoinsMining coinsMining = new CoinsMining();
      // Mining(Все цепочки того, кто принял; Пришедший десериализованный,
ОткрКлюч майнера))
      // Вернет новые цепочки
      List<byte[]> list_new_chains = coinsMining.Mining(GetAllChains(),
ch_and_tr, publicKey);
      // Сериализация
      byte[] data_for_send = Serial_list_bytes(list_new_chains);
      // Отправить всем, в том числе себе (тип сообщения 3)
      byte TypeMessage = 3;
      // Запуск потока отправки
      common helper = new common();
      rTB_temp3.Text += "\nОтправляю тип(" + TypeMessage + ") байт: " +
data_for_send.Length;
      Task.Run(() => helper.SendSocket(socket_sending,
```

```
        helper.getEndPoints(path_by_login_ip_port),
                    TypeMessage,
                    data_for_send,
                    buffer_length_for_getting));
      }
      // Сериализация list<byte[]>
      private byte[] Serial_list_bytes(List<byte[]> list_new_chains)
      {
        MemoryStream ms = new MemoryStream();
        BinaryFormatter formatter = new BinaryFormatter();
        formatter.Serialize(ms, list_new_chains);
        return ms.ToArray();
      }
      private List<byte[]> deSerial_list_bytes(byte[] data)
      {
        MemoryStream ms = new MemoryStream(data);
        BinaryFormatter formatter = new BinaryFormatter();
        return (List<byte[]>)formatter.Deserialize(ms);
      }
      // Сериализация list<list<byte[]>>
      private byte[] Serial_list_list_bytes(List<List<byte[]>>
list_chains_and_trans)
      {
        MemoryStream ms = new MemoryStream();
        BinaryFormatter formatter = new BinaryFormatter();
        formatter.Serialize(ms, list_chains_and_trans);
        return ms.ToArray();
      }
      private List<List<byte[]>> deSerial_list_list_bytes(byte[] data)
      {
        MemoryStream ms = new MemoryStream(data);
        BinaryFormatter formatter = new BinaryFormatter();
        return (List<List<byte[]>>)formatter.Deserialize(ms);
      }

      // Кнопка "Перевести"
      private void btn_transfer_Click(object sender, EventArgs e)
      {
        string login_ToUser = tB_ToUser.Text;
        // ОК получателя
        string publicKey_ToUser = "";
        using (FileStream fr = new FileStream(path_by_login_public_secret_mod,
FileMode.OpenOrCreate, FileAccess.Read))
        {
          using (StreamReader sr = new StreamReader(fr, Encoding.Default))
          {
            string line;
            while ((line = sr.ReadLine()) != null)
            {
              var login_public_secret_mod = line.Split();
              if (login_public_secret_mod[0] == login_ToUser)
              {
                publicKey_ToUser = login_public_secret_mod[1];
                break;
              }
            }
          }
        }
        if(publicKey_ToUser.Equals(""))
        {
          // Не найден пользователь с таким логином
          MessageBox.Show(login_ToUser + " не существует :(");
          return;
```

```
        }
        try{
          // Все цепочки отправителя
          List<byte[]> allMyChains = GetAllChains(); ;

          TMoney tmoney = new TMoney();
          List<List<byte[]>> list_chains_and_trans =
tmoney.SendMoney(publicKey, publicKey_ToUser, new
BigInteger(nUD_summSize.Value), mod, secretKey, allMyChains);

          // Сериализация для отправки
          byte[] bytes_for_send =
Serial_list_list_bytes(list_chains_and_trans);
// Отправка всем, но получат только майнеры, кроме отправителя (сообщения 2)
          byte TypeMessage = 2;
          // Запуск потока отправки
          common helper = new common();
          Task.Run(() => helper.SendSocket(socket_sending,
                      helper.getEndPoints(path_by_login_ip_port, ip_getting,
port_getting),
                      TypeMessage,
                      bytes_for_send,
                  buffer_length_for_getting));
        }
        catch(Exception ex)
        {
  MessageBox.Show(login + ":\n" + ex.ToString() + "\n" + ex.Message);
        }
      }
      // Изменение поля баланса
      private void tB_balance_TextChanged(object sender, EventArgs e)
      {
        // Изменять максимальное значение поля "сколько перевести"
        nUD_summSize.Maximum = Decimal.Parse(tB_balance.Text);
      }
      // Переключение Майнинг-Немайнинг
      private void cB_mining_CheckedChanged(object sender, EventArgs e)
      {
        CheckBox cB = (CheckBox)sender;
        if (cB.Checked)// Если начался майнинг
        {// Изменяется текст кнопки
          cB.Text = "...МАЙНИНГ...";
        }
        else // Если закончился майнинг
        {// Изменяется текст кнопки
          cB.Text = "Майнить";
      }}}}
```