ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО

ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

**«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ

И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Специальность: информационная безопасность

**Отчет по работе**

«Создание клиент-серверного приложения с реализацией криптографических методов шифрования и аутентификации»

**Работу выполнил:**

Рахимулин Ринат, гр. 09-208

**Преподаватель:**

Мубараков Булат Газинурович

|  |
| --- |
|  |

**Казань – 2016**

Содержание

1. [Введение………………………...………………………………………………………2](#_Toc446065913)
2. [Структура программы…………...……………………………………………………..3](#_Toc446065914)

2.1 Регистрация пользователя………..……....………………………………………...4

2.2 Вход - логин пользователя……………...………………………………………….4

2.3 Отправка сообщений……...………………………………………………………..4

1. [Примеры из кода программы………………………………………………………….5](#_Toc446065915)

3.1 Обработка сервером входящих соединений………………………………….......5

3.2 Выработка общего ключа Диффи-Хеллмана………………………………….…6

3.3 Выработка пар ключей RSA……………………………....………………………8

1. [Заключение…………………………………….……………………………………...10](#_Toc446065916)
2. **Введение**

Целью выполнения данной работы было написание клиент-серверного приложения, моделирующего-представляющего основные (самые распространенные) криптографические методы шифрования и аутентификации.

В качестве базового примеры было реализовано клиент-серверное приложение, моделирующее работу чата (реализация клиента и сервера, обмен между ними сообщениями на основе стека протоколов TCP/IP). Клиент и сервер представляют собой раздельные приложения, устанавливаемые на различные ЭВМ, но подключенные к одной сети(либо к локальной сети, либо хостовые машины сети Интернет(для сервера)). Программа написана на языке C#, с использованием фреймворка .NET 4.5, соединение – на базе асинхронных потоковых сокетов. Хотя, это просто декоративные слова, за которыми скрывается обычный отдельный поток и обычный сокет, работающий в этом потоке, пока существует соединение. Сокеты принимают и отправляют байты, в них уже встроены протоколы верификации ошибок при нестабильных соединениях.

Базовая модель чата включает в себя регистрацию клиентов, их вход в сеть (ввод логина-пароля, полученных в процессе регистрации) и отправку сообщений на сервер с чтением общей информации по беседе с сервера.

Были реализованы следующие алгоритмы: протокол обмена ключами Диффи-Хеллмана (не стойки к атаке «человек-по-середине», но позволяющий легко выработать общий ключ шифрования в процессе обмена сообщениями, а не до непосредственного соединения), протокол потокового шифрования RC4, который шифрует и дешифрует сообщения одним и тем же ключом, выработанным протоколом Диффи-Хеллмана. Так же реализована ЭЦП на базе криптографического алгоритма RSA и хеширование сообщений с помощью алгоритма MD5, что будет рассмотрено далее.

1. **Структура программы**

Как программист я не буду подробно разбирать каждый кусочек кода программы, т.к. дело это не благодарное и зависит от практической реализации. Постараюсь максимально абстрагироваться от названий файлов-функция и описать абстрактную модель клиент-серверного приложения ( в данном случае - чата).

Клиент и сервер представляют собой различный приложения, предназначенные для установки на компьютерах, подсоединенных к одной сети. Соединение осуществляется методом прослушивания адреса подключенного клиента и транслирующего сервера к одной паре значений адрес: IP-адрес + прослушиваемый порт. Любой действие пользователя(да и программы тоже) запускается в отдельном потоке, дабы не создавать ощущения «зависшего приложения». Так же это сделано для того, чтобы сервер на «зависал», обрабатывая сообщения от одного клиента, в то время как к нему в сокет «стучится» другой клиент с требованием установления соединения. Цель – асинхронная стабильная работа приложения

Что же делает клиент, когда подключается к серверу?

**2.1 Регистрация**

Если клиент ещё не регистрировался на сервере – ему нужно регистрироваться! Для этого он вводит логин-пароль, которые отправляются на сервер и проверяются сервером(а есть ли такой пользователь вообще?). Если есть – сервер генерирует случайное число и отправляет его клиенту. Клиент складывает свой пароль с этим числом и с помощью MD-5 хеш-функции получается хэш сообщения, который отправляет серверу. Сервер получает это сообщение и генерирует свой хэш на основе пароля пользователя и случайного числа. Если хэши совпадают – это наш клиент, разрешим ему соединение для регистрации и переходим к процессу генерации общего ключа Диффи-Хеллмана. Алгоритм Диффи-Хеллмана очень прост и хорошо описан в Википедии.

* 1. **Вход – логин.**

Когда пользователь зарегистрировался – он сохраняется в памяти сервера. И во время входа – так же отправляет логин-пароль серверу. Сервер проверяет – ага, с нами соединяется клиент с требованием входа в систему для отправки сообщений – и сервер решает сгенерировать базовые данные для осуществления ЭЦП. Но сначала – с помощью алгоритма Диффи-Хеллмана генерируется общий ключ для установленной сессии. Затем, сервер генерирует открытый ключ RSA [e,N] и отправляет их клиенту. Заметим, чтобы общий ключ уже выработан и любой сообщение шифруется-дешифруется как клиентом, так и сервером! И, выработав общие ключи шифрования для установленной сессии, сервер запускает эту сессию в отдельном потоке: пока не прервется связь или пока клиент не решит выйти из сессии.

* 1. **Отправка сообщений.**

Клиент прилогинился, теперь он может отправлять сообщения. И так, пользователь ввел какой-то текст и нажал кнопку «Отправить». Запускается отдельным поток на клиенте, который занимается отправкой сообщения на сервер. Сервер получил сообщение и тут он должен ответить клиенту что-то вроде «Всё хорошо, я получил твоё сообщение». Но вдруг этот сервер не наш сервер, а сервер злоумышленника? В нашем случае сервер генерирует свою подпись – прообраз сообщения – на основе пары приватных ключей RSA [d,N]. И пересылает этот прообраз вместе с сообщением клиенту. Клиент не будь дурак – проверяет это дело: расшифровывает полученный прообраз с помощью открытого ключа (как он это делает – прекрасно описано в Википедии) и сравнивает сообщения: если они совпали – то всё хорошо, он выводит сообщение «ЭЦС сервера: true». Иначе – пользователь будет предупрежден, что что-то не так, «друг оказался не друг – а так…».

**3. Примеры из кода программы**

**3.1 Выработка пар ключей RSA**

Ниже представлен функция, которые отвечают за отправку и получение сообщений как на сервере: сервер получается нового клиента-сокет через функцию Accept(), читает, что хочет сделать клиент(“R”-регистрация, “I” – input, вход) и реализует зависимое от этого своё поведение. Как уже говорилось – любой отдельное действие сервера выделено в отдельный независимый поток, который сам решает – когда и что нужно отправлять/принимать, без спрашивания пользователя.

// функция прослушки юзеров

private void AcceptSocket()

{

Thread th = new Thread(delegate () //создаем новый поток для прослушки

{

while ( running ) // делаем, пока сервер запущен

{

trace("ACCEPTING");

Socket temp = socket.Accept();// Появилось новое соединение

trace("ACCEPTED");

Thread subth = new Thread(delegate () //создаем новый поток для прослушки

{

bool isLoginned = false;

while ( !isLoginned )

{

string login, password, res; // временные переменные

res = ReceiveString(temp); //конвертируем байты в "путь" (регистр или авторизац)

if (res[0] == 'R') //если "путь" будет регистрация, то

{

generateKey(temp);

txtLog.Invoke(new Action(() => {

trace("При регистрации был сгенерирован ключ: " + DHkey.ToString());

}));

login = RC4(ReceiveString(temp), DHkey.ToString()); //конвертируем байты в логин

password = RC4(ReceiveString(temp), DHkey.ToString()); //конвертируем байты в пароль

User user = this.users.Where(x => x.login == login).FirstOrDefault();

if (user != null) // если такой логин уже содержится

{

SendBytes(temp, "0"); //то отправляем клиенту сообщение о том, что такой логин уже есть

}

else

{

users.Add(new User(login, password));

SendBytes(temp, "1");// отправляем клиенту сообщение о том, что регистрация прошла успешно

this.txtUsers.Invoke(new Action(() => { txtUsers.AppendText("Пользователь " + login + " зарегистрировался.\r\n"); }));

}

}

else if (res[0] == 'I') // иначе, если вход, то

{

password = "pswd";

bool exists = true;

login = ReceiveString(temp); //конвертируем байты в логин

User user = this.users.Where(x => x.login == login).FirstOrDefault();

if (user != null)

{

password = user.password;

}

else

{

exists = false;

}

Random rnd = new Random();

BigInteger n = Function.num\_gen(10);

SendBytes(temp, n.ToString());

string hash1 = ReceiveString(temp);

string hash2 = Function.calcHash(password + n);

generateKey(temp);

generateRSA();

// Отправляем открытые ключи

SendBytes(temp, N.ToString());

SendBytes(temp, E.ToString());

trace("EXISTS: " + exists);

if (exists == false || hash1 != hash2)

{

SendBytes(temp, "0");// Ошибка

}

else

{

isLoginned = true;

clients.Add(temp);

txtLog.Invoke(new Action(() => {

trace("При авторизации был сгенерирован ключ: " + DHkey.ToString());

}));

// Отправляем клиенту сообщение о том, что пользователь успешно вошел

SendBytes(temp, "1");

this.txtUsers.Invoke(new Action(() => { txtUsers.AppendText("Пользователь " + login + " вошел в сеть\r\n"); }));

// И создаем поток для принятия от юзера дальнейших сообщений

Thread thh = new Thread(delegate ()

{

try

{

// ждем сообщение от пользователя

string m = RC4(ReceiveString(temp), DHkey.ToString()); ;

while (m != "E")

{

trace(">user----");

//получили от юзера сообщение m

string M;

if (!running)

{

M = "Сервер недоступен!\r\n";

trace(M);

SendBytes(temp, RC4(M, DHkey.ToString()));

}

if ((running) && (m != "E"))

{

M = "Cервер получил сообщение: " + m;

trace(M);

SendBytes(temp, RC4(M, DHkey.ToString()));

BigInteger EDS = new BigInteger(Function.getBytesFromStr(M));// ЭЦП

SendBytes(temp, RC4(BigInteger.ModPow(EDS, D, N).ToString(), DHkey.ToString()));//отправляем ЭЦП

this.txtUsers.Invoke(new Action(() => {

txtUsers.AppendText( login + ":\r\n" + m + "\r\n");

}));

}

else

txtUsers.Invoke(

new Action(() => {

txtUsers.AppendText("Пользователь " + login + " вышел из сети\r\n");

}));

m = RC4(ReceiveString(temp), DHkey.ToString());

}

}

catch

{

txtUsers.Invoke(new Action(() => { txtUsers.AppendText("Пользователь " + login + " вышел из сети\r\n"); }));

}

});

thh.Start(); // запускаем этот поток

threads.Add(thh);

}

}

else

{

trace("LISTENING: pass");

// pass, в ином случае не реагировать на входящие данные

}

}

});

subth.Start(); // запускаем этот поток

threads.Add(subth);

}

});

th.Start(); // запускаем весь предыдущий поток

threads.Add(th); // Добавляем поток в список потоков

}

**3.2 Выработка общего ключа Диффи-Хеллмана**

public void generateKey(Socket temp)

{

trace("Generating key...");

BigInteger n = Function.num\_gen(8);

SendBytes(temp, n.ToString());

BigInteger q = Function.num\_gen(8);

SendBytes(temp, q.ToString());

BigInteger y = Function.num\_gen(7);

BigInteger K = Function.PowMod(q, y, n);

BigInteger M = BigInteger.Parse( ReceiveString(temp));

SendBytes(temp, K.ToString());

DHkey = Function.PowMod(M, y, n);

trace("Key generated!");

}

**3.3 Выработка пар ключей RSA**

public void generateRSA()

{

P = Function.num\_gen(100);

Q = Function.num\_gen(100);

N = P \* Q;

FI = (P - 1) \* (Q - 1);

E = Function.num\_gen(P.ToString().Length / 2 + 1);

D = get\_inverse(FI, E);

}

**4. Заключение**

В ходе выполнения работы все поставленные цели были выполнены: реализация клиент-серверного приложения на базе различных криптографических протоколов. Хотя приложение временами работает нестабильно, но свою функцию оно выполняет полностью: верификация любых действий клиента и сервера, установление безопасного (пусть и примитивного) виртуального канала обмена данными ( когда непосредственно сам обмен сообщениями происходит методом их шифрования, а не прямой передачи без шифрования).  
  
 Для дальнейшей работы требуется только лишь более вдумчивый подход к реализации основных алгоритмов на базе очень больших чисел (на данный момент – каждая реализация насквозь кустарная и написана по документациям из RFC и Википедии), а так же – хороший дизайнер, т.к. требования к дизайну у современных пользователей очень высокие. Иначе никто не будет пользоваться таким приложением, т.к. рынок свободного ПО буквально наводнен подобными кустарными реализациями.

Поставленную цель считают выполненной и приложением – реализованным.