Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт математики и фундаментальной информатики
Базовая кафедра вычислительных и информационных технологий

УТВЕРЖДАЮ
Заведующий кафедрой
_____ / В. В. Шайдуров

«____» _____ 2019 г.

БАКАЛАВРСКАЯ РАБОТА

Направление 02.03.01 Математика и компьютерные науки

Защищенная кроссплатформенная система обмена сообщениями в сети Интернет

Научный руководитель
кандидат технических наук,
доцент _____/ С.В. Исаев

Выпускник _____/ А.М. Сухих

Красноярск 2019

РЕФЕРАТ

Выпускная квалификационная работа по теме «Защищенная кроссплатформенная система обмена сообщениями в сети Интернет» содержит 47 страниц текстового документа, 1 приложение, 20 использованных источников, 28 иллюстраций.

СЕТЕВАЯ БЕЗОПАСНОСТЬ, ШИФРОВАНИЕ ДАННЫХ, КРИПТОГРАФИЯ, СИММЕТРИЧНЫЕ КРИПТОСИСТЕМЫ, АСИММЕТРИЧНЫЕ КРИПТОСИСТЕМЫ, ОБМЕН СООБЩЕНИЯМИ.

Объект исследования – методы и технологии защиты данных.

Цель работы - создание программной системы, пользователи которой могли бы обмениваться сообщениями между собой, не опасаясь компрометации передаваемых и получаемых данных, а также, не завися от используемой операционной системы.

В результате работы реализована система, состоящая из двух приложений, клиентского и серверного, позволяющая своим пользователям безопасно обмениваться любыми данными.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Сложно переоценить значимость компьютерных сетей в современном мире. Самая большая и распространённая сеть, под названием Интернет ежедневно используется миллионами людей по всему миру для решения своих рутинных задач и общения. В интернете люди могут оплатить услуги жилищных компаний, купить подарки себе и друзьям, отдать приказ банку на перевод денег близким. При общении друг с другом люди отправляют в сеть много личной информации, о себе, своих детях, и работе. Естественно, никому не хочется, чтобы его банковские операции или личная информация попали в руки к мошенникам. Именно поэтому сегодня вопрос безопасности компьютерных сетей и защиты информации в них от посторонних глаз стоит особенно остро.

Целью данной работы было создание программной системы, пользователи которой могли бы обмениваться сообщениями между собой, не опасаясь компрометации передаваемых и получаемых данных, а также, не завися от используемой операционной системы, будь то Linux или windows.

На сегодняшний день такие системы уже существуют и широко распространены, как пример можно привести viber, telegram, WhatsApp и другие. Однако многие из них имеют закрытый исходный код, и проверить, что же на самом деле происходит с вашими данными не удастся.

1 Безопасность в сетях

Безопасность сети является довольно обширной темой, однако ее можно условно разделить на несколько основных областей: аутентификация, секретность, целостность и строгое соблюдение обязательств. Аутентификация отвечает за определение личности пользователя, перед предоставлением ему доступа к какой-либо информации. Секретность следит за недоступностью вашей информации неавторизованным пользователям. Контроль целостности позволяет убедиться, что полученное сообщение не было специально изменено или случайно повреждено каким-либо образом в процессе передачи. И, наконец, задача обеспечения строгого соблюдения обязательств заключается в использовании цифровых подписей.

Защита информации в сетях основана на принципах криптографии. Поэтому стоит поговорить о ней подробнее.

1.1 Основные принципы и понятия криптографии

Обеспечение защиты информации в криптографии происходит за счет методов шифрования — обратимого преобразования исходного текста на некоторой функции и специального параметра, называемого ключом, в шифрованный текст. Традиционная криптография образует раздел симметричных криптосистем, в которых шифрование и дешифрование проводится с использованием одного и того же секретного ключа. Помимо этого раздела современная криптография включает в себя асимметричные криптосистемы, системы электронной цифровой подписи, хеш-функции, управление ключами, получение скрытой информации и многое другое.

Сообщения, требующие секретной передачи и называемые открытым текстом, преобразуются на стороне отправителя с помощью специальной функции шифрования и секретного ключа в зашифрованный текст. После этого зашифрованный результат, передается по каналу связи получателю. Предполагается, что за каналом связи наблюдает некий злоумышленник, который видит и копирует зашифрованный текст. Но для злоумышленника расшифровка сообщения представляет сложную, а иногда и вовсе не решаемую на практике вычислительную задачу, потому что в отличие от получателя, которому предназначено сообщение, ему не известен ключ дешифровки. Существует два типа злоумышленников: пассивный – может только прослушивать канал связи, и активный, который помимо прослушивания способен вставлять свои сообщения или изменять исходные сообщения до того, как они достигнут получателя. На рисунке 1 показана модель шифрования-дешифрования и положение злоумышленника в системе передачи данных.

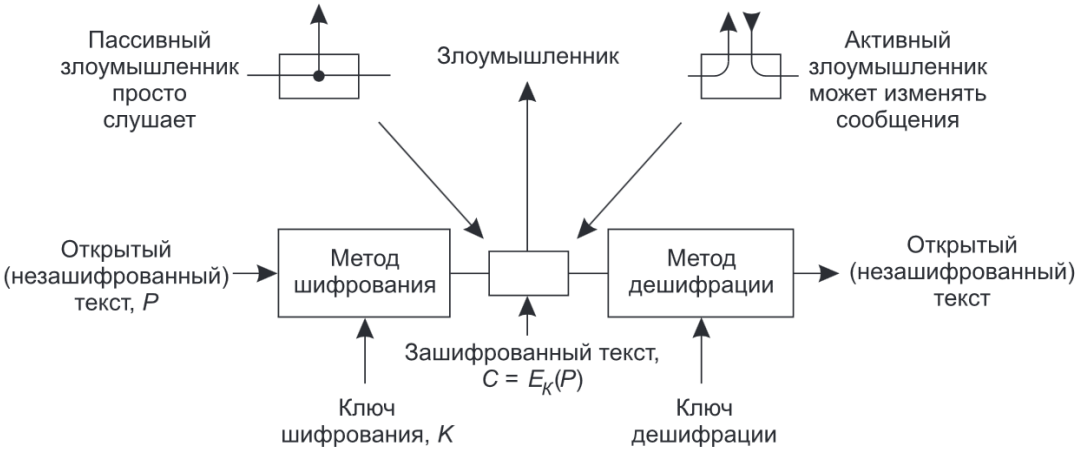

Рисунок 1 - модель шифрования – дешифрования

На практике при компрометации одного метода шифрования необходимо приложить огромные усилия и средства, для разработки другого. Поэтому принято считать, что никакие методы не являются секретными, другими словами считается, что взломщику шифра прекрасно известен метод используемый для шифрования. Идея, заключающаяся в предположении о том, что взломщику известен метод и краеугольным камнем секретности является эксклюзивный ключ, называется принципом Керкгофа, который гласит: Алгоритмы шифрования общедоступны; секретны только ключи.

1.2 Шифрование

Любой современный шифр используемый на практике, представляет собой смесь двух основных методов: метода подстановки и метода перестановки.

Метод подстановки представляет из себя замену каждого символа (или группы символов) исходного текста некоторым другим символом (или группой символов). Стоит отметить, что шифры на основе подстановок сохраняют порядок следования символов.

Шифры, в которых используется метод перестановки, напротив изменяют лишь порядок символов в исходном тексте, но сами символы при этом не изменяются.

В традиционной криптографии использовались простые алгоритмы. В современной криптографии, однако, цель состоит в том, чтобы создать алгоритм настолько сложный и запутанный, что никакой пользы не может быть получено из зашифрованного текста без ключа. Существует два класса алгоритмов шифрования – с симметричным и открытым ключами.

1.2.1 Алгоритмы шифрования с симметричным ключом

Класс алгоритмов с симметричным ключом получил свое название потому что ключ, используемый для шифрования на стороне отправителя, совпадает с ключом дешифрования, используемым получателем. Основной проблемой таких алгоритмов являются способы установления секретного ключа между собеседниками, так как необходимо распределить ключ между всеми участниками общения совершенно секретным способом.

Наиболее известными алгоритмами шифрования с симметричным ключом являются:

1. DES (Data Encryption Standard) – в данный момент устарел и не является безопасным.
2. Triple-DES – тройной DES
3. AES (Advanced Encryption Standard) или Rijndael – наиболее распространенный алгоритм, чаще всего используемый на практике
4. Twofish – также широко распространенный алгоритм
5. ГОСТ 28147-89 – Российский стандарт шифрования

1.2.2 Алгоритмы шифрования с открытым ключом

Диффи и Хеллман - исследователи из Стэнфордского университета в 1976 году предложили совершенно новую криптосистему, в которой ключ шифрования и ключ дешифрования были различны. В их системе алгоритмы шифрования (E) и дешифрования (D) должны были удовлетворять трем требованиям:

1. D(E(P)) = P
2. Крайне сложно вывести D из E
3. E нельзя взломать при помощи произвольного открытого текста.

С тех пор, многие алгоритмы были созданы с использованием концепции открытых ключей. Алгоритм публичный, нет необходимости в секретных каналах связи. Общая схема выглядит следующим образом:

1. Каждый пользователь генерирует пару ключей: один для шифрования, другой для расшифровки.
2. Каждый пользователь публикует свой ключ шифрования (открытый ключ), размещает его в открытом для всех доступе. Второй ключ (секретный ключ), соответствующий открытому, сохраняется в секрете.
3. Если пользователь *A* собирается послать сообщение пользователю *B*, он шифрует сообщение открытым ключом пользователя *B*.

4. Когда пользователь *B* получает сообщение, он расшифровывает его с помощью своего секретного ключа. Другой получатель не сможет расшифровать сообщение, поскольку личный ключ *B* знает только *B*.

Наиболее надежным и распространенным алгоритмом для шифрования с открытыми ключами на сегодняшний день является алгоритм RSA названный по начальным буквам имен людей, его придумавших (Rivest, Shamir, Adleman). Его математическое описание довольно интересно и не требует много времени в отличии от остальных алгоритмов, поэтому приведем его ниже.

1.2.2.1 Описание алгоритма RSA

Сперва вспомним некоторые математические утверждения, необходимые для понимания алгоритма.

Во-первых, нам понадобится функция Эйлера.

Функция Эйлера $\varphi(n)$ - Арифметическая функция, равная количеству натуральных чисел меньших своего аргумента и взаимно простых с ним. Функция Эйлера обладает следующими интересующими нас свойствами:

1. Если n, m – взаимно просты, то

$$\varphi(nm) = \varphi(n) * \varphi(m). \quad (1)$$

2. Если p – простое число, то

$$\varphi(p) = p - 1. \quad (2)$$

Также нам понадобятся две следующие теоремы:

Теорема 1: Малая теорема Ферма

Если p – простое число и a – целое число, такое, что НОД (a, p) = 1, то

$$a^{p-1} \equiv 1(\mathrm{mod\ p}). \quad (3)$$

Теорема 2: Китайская теорема об остатках

Пусть к – натуральное число и $m_1, \ldots, m_k$ – целые, взаимно простые числа, произведение которых равно

$$M = \prod_{j=1}^{k} m_j. \quad (4)$$

Тогда для любого набора целых чисел $a_1, \ldots, a_k$ решение системы сравнений

$$\begin{cases} x \equiv a_1 \ (mod\ m_1) \\ \quad\quad ... \\ x \equiv a_k \ (mod\ m_k) \end{cases} \tag{5}$$

единственно по модулю M и удовлетворяет сравнению

$$x \ \equiv \ \sum_{i=1}^{k} a_i b_i c_i \ \ (mod\ M), \tag{6}$$

где

$$b_i = \frac{1}{m_i} * \left(\prod\nolimits_{j=1}^{k} m_j \right) = \frac{M}{m_i} \tag{7}$$

и

$$c_i \ \equiv \ b_i^{-1} \ (mod\ m_i). \tag{8}$$

На этом необходимая математическая база заканчивается.

Теперь опишем процесс шифрования. Сперва каждый абонент вырабатывает свою пару ключей. Для этого он генерирует два больших простых числа p и q, и вычисляет их произведение N.

$$N = p * q. \tag{9}$$

Затем он вырабатывает случайное число e, взаимно простое со значением $\varphi(N)$ - функции Эйлера от числа N, причем из упомянутых выше свойств следует, что

$$\varphi(N) = (p - 1) * (q - 1). \tag{10}$$

Далее находится число d из условия

$$e * d \equiv 1 \ (mod\ \varphi(N)). \tag{11}$$

Так как НОД $(e, \varphi(N)) = 1$, то такое число d существует, и оно единственно. Пару (N, e) абонент объявляет открытым ключом и помещает в открытый доступ. Пара (N, d) является секретным ключом и сохраняется в тайне. Для расшифровки достаточно знать секретный ключ.

Для зашифровки текст должен быть переведен в числовую форму каким-либо способом, в результате чего он примет вид некоторого большого числа.

Затем полученное число разбивается на части так, чтобы каждая из них была числом в промежутке $[0, N-1]$. Процесс шифрования одинаков для каждой части. Поэтому мы можем считать, что часть исходного текста представлена числом x, $0 \leq x < N$.

Пользователь A, отправляющий сообщение x абоненту B, выбирает из открытого каталога пару (N, e) абонента B и вычисляет шифрованное сообщение по формуле

$$y = x^e (mod\ N). \tag{12}$$

Абонент B вычисляет исходный текст по формуле

$$x = y^d (mod\ N). \tag{13}$$

Покажем, что вычисленное значение действительно равно исходному тексту. Имеем:

$$y^d = (x^e)^d = x^{ed}\ (mod\ N). \tag{14}$$

Так как

$$e * d\ \equiv 1\ mod\big(\varphi(N)\big), \tag{15}$$

то оно представимо в виде

$$e * d = 1 + k * \varphi(N) \tag{16}$$

для некоторого целого числа k, значит

$$x^{ed} = x^{1+k\phi(N)} (\mathrm{mod}\ N) = x^{1+k(q-1)(p-1)} (\mathrm{mod}\ pq). \tag{17}$$

Далее рассмотрим два случая,

1) x = 0 mod(p), тогда

$$x^{1+k(q-1)(p-1)} = 0 = x\ mod\ p. \tag{18}$$

2) x ≠ 0 mod(p), тогда

$$x^{1+k(q-1)(p-1)} = x * (x^{p-1})^{k(q-1)}\ mod\ p. \tag{19}$$

Согласно малой теореме ферма:

$$(x^{p-1})^{k(q-1)}\ mod\ p\ \equiv 1^{k(q-1)}\ mod\ p\ \equiv 1\ mod\ p. \tag{20}$$

Имеем

$$y^d = x^{ed} = x * (x^{p-1})^{k(q-1)} \, mod \, p = \; x * 1 \, mod \, p = \; x \, mod \, p. \qquad (21)$$

Аналогичные рассуждения можно провести, поменяв p и q местами, то есть данное сравнение выполняется для числа p и для числа q, а, следовательно, по китайской теореме об остатках данное сравнение выполняется и для произведения чисел p и q. Следовательно

$$y^d = \; x^{ed} = x \, mod \, N. \qquad (22)$$

Что и требовалось доказать.

1.3 Цифровые подписи

В некоторых ситуациях люди могут попытаться отказаться от своих обязательств, принятых ранее. В современном мире обязанности, которые берет на себя на себя кто-либо фиксируются юристом на бумаге, а затем человек, берущий эти обязанности ставит на ней свою роспись, как знак согласия с ними. Компьютерные системы также нуждаются в подобном механизме.

Требуется система, которая в переписке двух пользователей гарантирует выполнение следующих условий:

1. Получатель имеет возможность проверить личность отправителя;
2. Отправитель не может отрицать содержимое сообщения после его отправки;
3. Получатель не имеет возможности модифицировать подписанное сообщение.

Цифровые подписи как раз и являются такой системой. Также, как и шифры цифровые подписи делятся на две разновидности – подписи с открытым и закрытым ключом.

1.3.1 Подписи с симметричным ключом

Метод подписей с симметричным ключом заключается в создании центрального органа (ЦО), которому все доверяют. Каждый пользователь выбирает секретный ключ и делится им с ЦО. Когда Алиса хочет послать Бобу текст P, она формирует сообщение, зашифрованное KA (ключом Алисы): KA(B, RA, t, P) , где B – идентификатор Боба, RA – случайное число, выбранное Алисой, t – метка времени, подтверждающая свежесть сообщения. Затем Алиса посылает это сообщение в ЦО. ЦО расшифровывает сообщение Алисы, после чего формирует свое сообщение, зашифрованное KB (ключом Боба): KB(A, RA, t, P,

KC(A, t, P)), KC(A, t, P) – сообщение зашифрованное ключом ЦО и являющееся подписью ЦО.

1.3.2 Подписи с открытым ключом

Основной проблемой подписи с симметричным ключом является то, что каждый должен довериться некоторому Центральному Органу, который к тому же сможет читать все сообщения, проходящие через него. Будет лучше, если никому не придется доверять.

Шифрование с открытым ключом может помочь в этом. Для этого мы требуем, чтобы алгоритмы шифрования и дешифрования помимо свойства D(E(P)) = P имели еще одно: E(D(P)) = P. Алгоритм RSA обладает этим свойством. При такой схеме Алиса посылает Бобу подписанное сообщение EB(DA(P)) (текст, зашифрованный сначала закрытым ключом Алисы, а затем открытым ключом Боба), получая такое сообщение Боб сначала расшифровывает его своим закрытым ключом, а затем открытым ключом Алисы. При этом подписью будет DA(P), так как DA известно только Алисе.

1.4 Профили сообщений

Методы цифровых подписей критикуются за сочетание двух различных функций: аутентификации и секретности. Часто требуется только аутентификация.

Схема аутентификации, не требующая шифрования, основана на идее необратимой хеш-функции. Хеш-функции – это функции, предназначенные для сжатия произвольного сообщения, из заданного текста произвольной длины она вычисляет значение фиксированной длины. Эта хеш-функция, которую называют профилем сообщения (Message Digest, MD) должна иметь следующие свойства:

1. По заданному тексту P легко посчитать значение хеш-функции MD(P)
2. По цифровой подписи MD(P) практически невозможно определить исходный текст P
3. Для заданного P практически невозможно подобрать P’ так, чтобы

   MD(P) = MD(P’)
4. Изменение хотя бы одного бита входной последовательности приводит к совершенно другому результату.

Профиль сообщения по части открытого текста вычисляется намного быстрее, чем шифруется все сообщение. Поэтому профили сообщений можно использовать для ускорения цифровых подписей.

Рассмотрим как профиль сообщения может помочь в случае цифровой подписи с симметричным ключом: ЦО вместо сообщения KB(A, RA, t, P, KC(A, t, P)) будет отправлять KB(A, RA, t, P, KC(A, t, MD(P))). Использование профиля сообщения экономит время на шифрование и затраты на транспортировку и хранение.

Профиль сообщения также может применяться для гарантии сохранности сообщения при передаче его по сети в системах шифрования с открытым ключом. Так Алиса сначала вычисляет профиль своего сообщения P – MD(P), а затем шифрует (подписывает) профиль сообщения – DA(MD(P)) и посылает Бобу пару P, DA(MD(P)) – открытый текст и подписанный профиль.

Наиболее известными алгоритмами цифровой подписи являются MD5 (Message Digest 5) – который сейчас считается ненадежным и широко распространенный SHA-1 (Secure Hash Algorithm 1), который тоже начал устаревать и его потихоньку заменяет SHA-2.

1.5 Управление ключами

Благодаря ассиметричной криптографии пользователи могут обмениваться секретными данными, не устанавливая перед этим общего секретного ключа, подписывать свои сообщения электронными подписями, не доверяя их третьему лицу, а также проверять не повреждены ли полученные данные.

Но как два незнакомца обмениваются открытым ключом перед началом разговора? Просто оставить открытый ключ на каком-то сайте в Интернете не получится, потому что когда один из собеседников сделает запрос на сайт для получения открытого ключа, злоумышленник может заменить оригинальный ключ в ответе сайта на свой, а затем перехватить и даже изменить чужие сообщения. Поэтому необходимо придумать механизм, который позволял бы безопасно обмениваться открытыми ключами.

1.5.1 Сертификаты

Таким механизмом стали организации, занимающиеся сертификацией открытых ключей, которые называются центрами сертификации (ЦС).

Теперь, если кто-то хочет установить безопасное соединение с незнакомыми людьми, ему нужно прийти в центр сертификации со своим открытым ключом и зарегистрировать его. Центр сертификации выдаст сертификат, аналогичный показанному на рисунке 2, Хеш SHA-1 которого будет подписан закрытым ключом этого центра. Задачей этого сертификата является связывание открытого ключа с его владельцем. Сам по себе этот сертификат не защищается и не хранится в тайне.

Также, как и ранее, чтобы два незнакомца начали общаться, им нужно выложить свои сертификаты в открытый доступ. Однако теперь, если злоумышленник попытается подделать сертификат или открытый ключ, это не сработает, так как Хеш его сертификата при проверке не совпадет с Хешем, вычисленным по открытому ключу центра сертификации и блоку подписи.

Настоящим удостоверяю, что открытый ключ
19836A8B03030CF83737E3837837FC3s87092827262643FFA82710382828282A
принадлежит

имя
адрес проживания
год рождения
Электронный адрес

Хеш SHA-1 данного сертификата подписан закрытым ключом Управления сертификации

Рисунок 2 - примерный вид сертификата

На самом деле естественно, для того чтобы получить сертификат вам не придется никуда идти, так как был разработан специальный стандарт сертификатов X.509, широко распространенный в интернете.

Одного Центра Сертификации на весь мир конечно недостаточно. По этой причине был создан способ сертификации открытых ключей под называнием PKI (Public Key Infrastructure – инфраструктура систем с открытыми ключами).

Одним из типов PKI является иерархия управлений, представленная на рисунке 3. На рисунке 3 представлены три уровня, однако их может быть как больше, так и меньше. Центр сертификации верхнего уровня мы будем называть центральным управлением (ЦУ), оно сертифицирует управления второго уровня, назовем их Региональными отделами (РО). Региональные отделы в свою очередь сертифицируют настоящие Управления сертификации, которые выдают клиентам сертификаты стандарта X509.

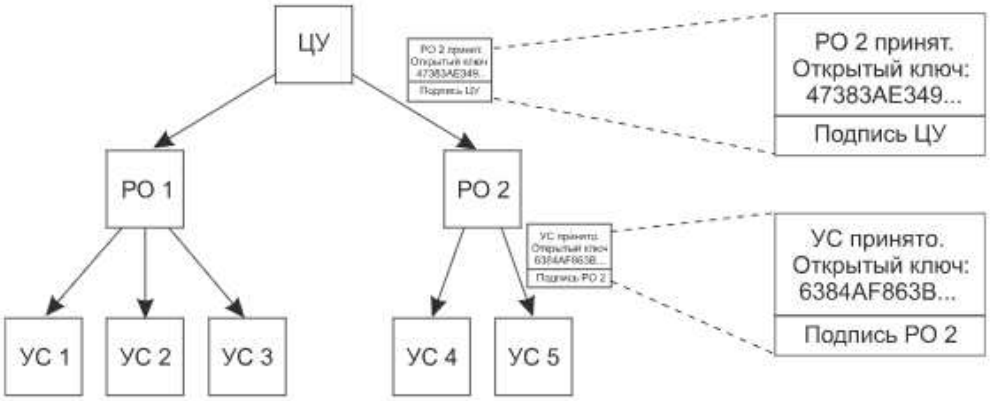

Рисунок 3 - иерархия управлений

Теперь, когда кто-то, например, Алиса, хочет проверить сертификат своего собеседника Боба, она запрашивает у него сам сертификат, а также цепочку сертификатов управлений верхних уровней. Такая цепочка называется цепочкой доверия или путем сертификата. Таким образом Алиса может проверить все интересующие ее организации по цепочке. Но как проверить открытый ключ Центрального управления? Предполагается, что открытый ключ Центрального управления знают все, обычно ключи различных этих управлений «вшиты» внутрь браузеров и других клиентских приложений.

1.6 Защита информации во всемирной паутине. SSL и TLS

В момент появления интернета никто особо не заботился о безопасности передаваемых данных, однако, когда люди поняли все его преимущества и начали разрабатывать различные приложения для него, например, стали использовать интернет для проведения финансовых операций стало понятно, что передаваемые данные нуждаются в защите. В 1995 году публике была представлена новая система безопасности под названием SSL (Secure Sockets Layer — протокол защищенных сокетов). Соответствующее программное обеспечение, как и сам протокол SSL, сегодня широко используется.

Данный протокол устанавливает между двумя пользователями безопасное соединение позволяя:

1. согласовать параметры, которые будут использоваться для передачи данных;
2. подтвердить личности пользователей;
3. организовать между пользователями тайное общение;
4. обеспечить целостность и защиту передаваемых данных.

После того, как безопасное соединение установлено SSL занимается поддержкой сжатия и шифрования передаваемых данных. Существует несколько версий SSL, последней и самой надежной является SSLv3. В зависимости от версии, SSL может иметь различные дополнительные функции, включая наличие или отсутствие сжатия, тот или иной алгоритм шифрования, а также некоторые вещи, связанные с ограничениями экспорта в криптографии.

SSL состоит из двух подпротоколов, один из которых предназначен для установления защищенного соединения, а второй — для использования этого соединения. Начнем с рассмотрения вопроса установления соединения. Работа подпротокола, занимающегося этим, показана на рисунке 4. Все начинается с сообщения, в котором Алиса посылает Бобу запрос на установку соединения. В нем она указывает используемую верси. SSL, а также свои предпочтения насчет алгоритмов сжатия и шифрования. В этом сообщении также содержится число

Алисы RA, называемое нонс (одноразовый код), который будет использован впоследствии.

Далее вступает Боб. В ответном сообщении он выбирает один алгоритм из тех, что были предложены Алисой, а также добавляет свой нонс RB. В следующем своем сообщении он отсылает свой сертификат с открытым ключом и цепочку доверия этого сертификата, если она необходима. После того как Боб выполнил свою часть протокола он посылает сообщение, в котором говорит, что настала очередь Алисы. Алиса в ответ генерирует 384-разрядный подготовительный ключ, шифрует его с помощью своего открытого ключа и отсылает его Бобу. Настоящий ключ сеанса будет вычислен на основе нонсов обеих сторон и подготовительного ключа. После получения этого сообщения Алиса, и Боб вычисляют сеансовый ключ. Для этого Алиса просит Боба сменить шифр, а также сообщает о том, что она считает подпротокол установления соединения оконченным, после чего Боб соглашается с ней.

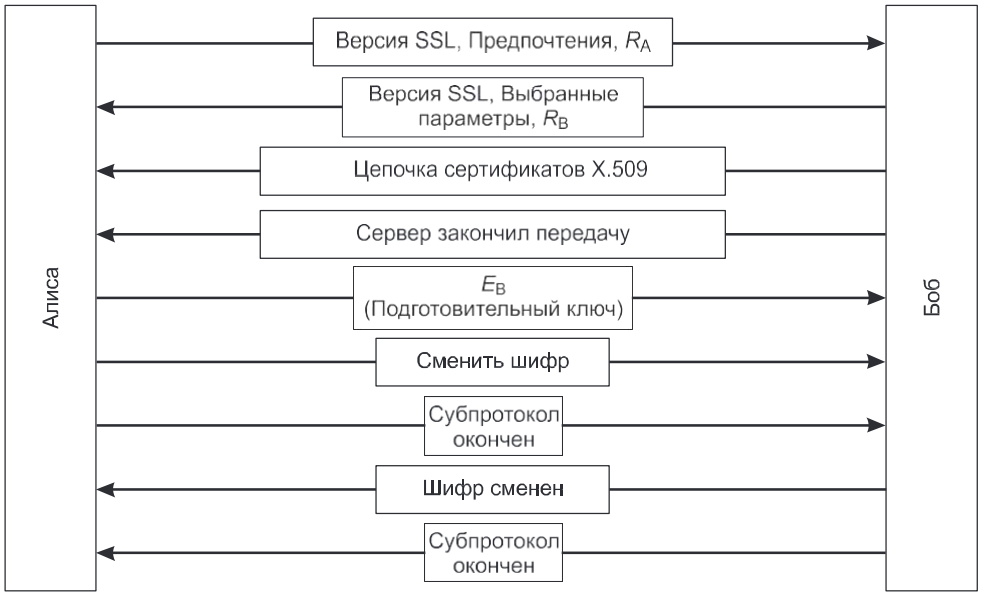

Рисунок 4 - установление SSL соединения

Второй подпротокол, используемый для передачи данных, представлен на рисунке 5. Сообщения, поступающие от пользователя, разбиваются на единицы данных размером до 16 Кбайт и, при необходимости, сжимаются независимо друг от друга. Затем со сжатым текстом объединяется закрытый ключ, а результат хешируется по согласованному заранее алгоритму. Хеш добавляется к каждому фрагменту в виде MAC (Message Authentication Code — код аутентификации сообщения). Этот сжатый фрагмент вместе с MAC кодируется согласованным алгоритмом с симметричным ключом и передается по каналу связи получателю.

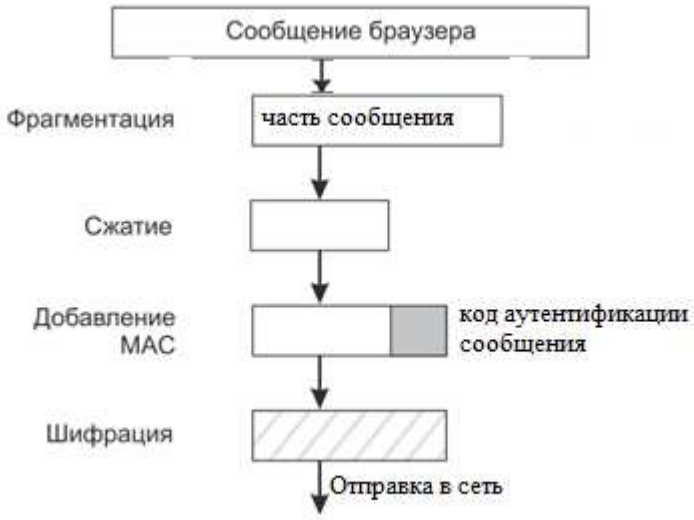

Рисунок 5 - передача данных через ssl

В 1996 году протокол SSL был отправлен на стандартизацию. Результатом стал стандарт TLS (Transport Layer Security – протокол безопасности транспортного уровня). TLS был основан на SSLv3. При создании стандарта TLS в SSL было внесено не так много изменений, но этих изменений хватило, чтобы сделать SSLv3 и TLS несовместимыми. В результате, для устранения проблем с совместимостью протоколов множество клиентских приложений используют оба протокола, и при необходимости TLS преобразуется обратно в SSL, такое решение называется SSL / TLS.

2 Архитектура системы обмена сообщениями в сети интернет

Для выполнения работы была реализована система обмена сообщениями в сети интернет, в которую входят серверное приложение, клиентское приложение, а также некоторые дополнительные модули, необходимые для их работы. Данная система является кроссплатформенной, т.е. может работать на различных операционных системах: Windows, macOS и системах, основанных на ядре Linux таких как android (с некоторыми ограничениями), Ubuntu, Linux Mint, Raspbian и прочих. Также эта система позволяет передавать и хранить сообщения подписанными, зашифрованными и с гарантией целостности, таким образом, получить доступ к ним смогут только те люди, для которых они предназначены, а любые изменения, которым их могут подвергнуть злоумышленники будут обнаружены. Архитектура системы в общем виде представлена на рисунке 6.

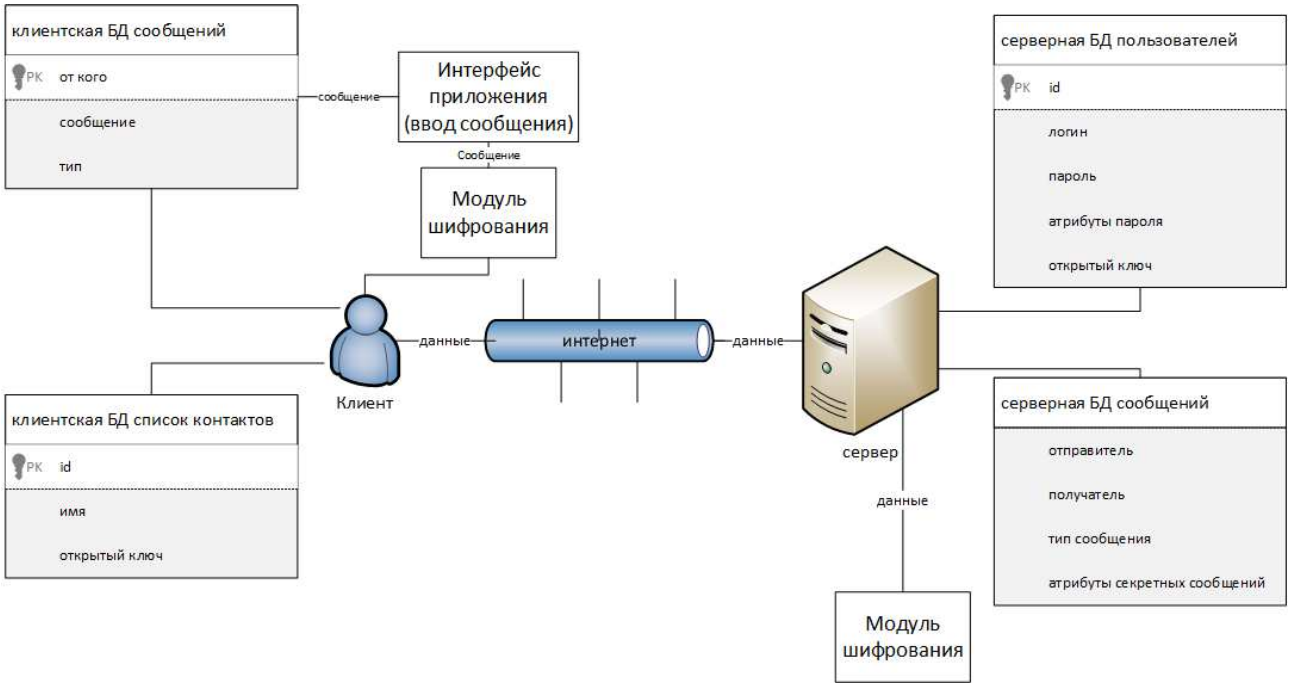

Рисунок 6 - архитектура разработанной системы

Объем программного кода занимает 188 КБ места на диске. Состав и краткое описание основных модулей описано в приложении А.

2.1 Инструменты использованные при реализации

Для написания основного программного кода системы был выбран язык программирования Python версии 3.7, графический интерфейс приложения был реализован с помощью модуля tkinter, поставляемого в составе стандартной библиотеки Python. Хранение данных приложения, таких как сообщения, список

пользователей и других осуществляется в базе данных SQLite, предпочтение было отдано именно ей так как это легковесная, быстрая, надежная и широко распространенная база данных, отлично подходящая как для настольных компьютеров, так и для мобильных устройств. Для реализации криптографических функций таких как цифровые подписи, хеширование, симметричное и ассиметричное шифрование использовалась библиотека PyCryptodome с открытым исходным кодом. Написание кода происходило в среде разработки PyCharm Community edition от компании JetBrains. Для просмотра содержимого баз данных использовалось расширение SQLite Manager для браузера Mozilla Firefox. Поясняющие иллюстрации и схемы были нарисованы с помощью программ GIMP и Microsoft Visio.

2.2 Возможности реализованной системы

Возможности серверной части данной системы довольно просты и в основном сводятся к обслуживанию клиентских запросов таких как аутентификация, запросы к базе данных и пересылка сообщений от одного пользователя к другому. Поэтому перейдем сразу к рассмотрению клиентской части системы.

Для того чтобы воспользоваться клиентским приложением пользователь должен первым делом пройти процесс регистрации или, если он уже зарегистрирован, процесс аутентификации, таким образом обеспечивается гарантия того, что пользователь является тем, за кого он себя выдает. При регистрации пользователь должен выбрать себе имя (логин) из тех, что еще не были использованы, т.е. не содержаться в базе данных пользователей, хранящейся на сервере и которое будет отображаться другим пользователям при общении с ним, а также с помощью которого, другие пользователи смогут добавить этого человека в свой список контактов.

Для того чтобы добавить интересующего нас человека в список своих контактов нужно просто ввести его логин в специальное поле ввода и нажать на кнопку «Добавить друга», после чего он отобразиться в списке контактов.

После того, как пользователь прошел процесс аутентификации и нашел людей, с которыми он планирует начать общение, он выбирает интересующего его человека из своего списка контактов и способ передачи сообщений.

Передача сообщений может осуществляться одним из четырех способов:

1. Передача не зашифрованных сообщений от одного пользователя другому через сервер.
2. Передача зашифрованных сообщений от одного пользователя другому через сервер. Для этого перед передачей сообщения нужно сперва установить секретный ключ, нажав на кнопку «установить секретный ключ».

3. Передача не зашифрованных сообщений напрямую между клиентами исключая промежуточное звено в виде сервера. Для этого перед передачей сообщения нужно сперва установить соединение напрямую, нажав на кнопку «установить p2p соединение».
4. Передача зашифрованных сообщений напрямую между клиентами исключая промежуточное звено в виде сервера. Для этого нужно установить и секретный ключ, и прямое соединение, если они не были установлены ранее.

Схемы передачи сообщений для способов 1 и 2 представлены на рисунке 7, а для способов 3 и 4 на рисунке 8.

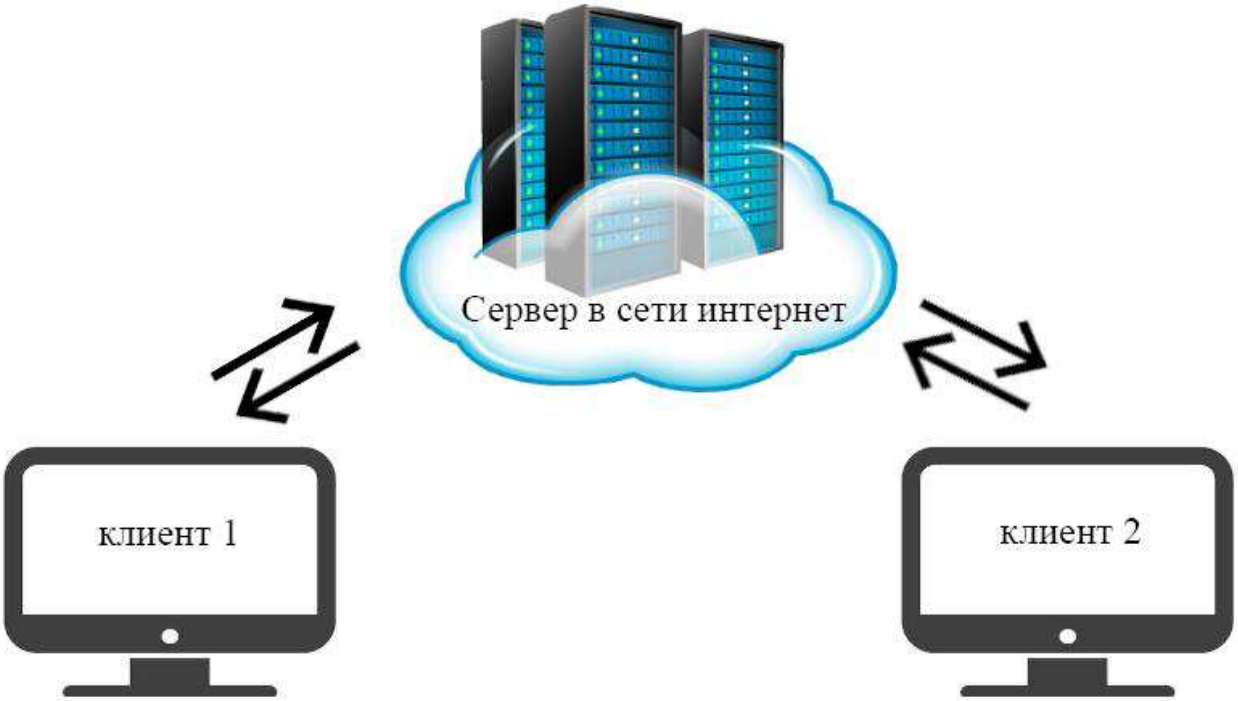

Рисунок 7 - схема передачи данных способами 1 и 2

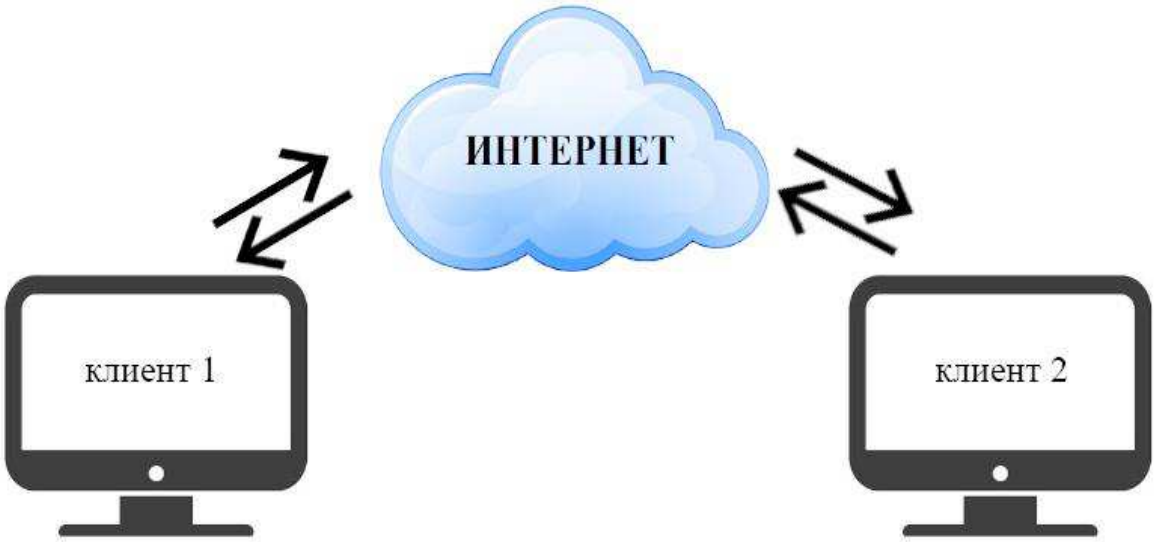

Рисунок 8 - схема передачи данных способами 3 и 4

При передаче сообщений способами 1 и 3 нет никакой гарантии, что сообщения, отправленные пользователем, не будут перехвачены и прочитаны посторонними людьми, однако подменить или каким-то образом изменить передаваемое между пользователями сообщение без их ведома не получится, так как сообщение будет помечено как поврежденное, данные способы подходят для обычного общения без передачи какой-либо тайной информации, личной или служебной, паролей и так далее.

При передаче сообщения способами 2 и 4 никому не удастся не только каким-либо образом незаметно повредить сообщение, но также и прочитать его или добыть какую-то важную информацию, так как сообщения будут зашифрованы при передаче от первого клиента и расшифрованы только при получении вторым клиентом (даже сервер, в случае передачи способом 2, не сможет расшифровать передаваемое сообщение). Данные способы подходят для передачи информации любого типа, однако будут также и накладные временные расходы на зашифровку и расшифровку этой информации.

Этими четырьмя способами передачи данных гарантируется целостность сообщений, и, в случае шифрованной передачи, их секретность.

Отметим, что любым из этих способов можно передавать не только сообщения, но и какие угодно файлы.

В любой момент времени пользователь может выйти из своего аккаунта или совсем удалить его из базы данных сервера.

Также клиентское приложение способно работать и без графического интерфейса, то есть в консольном режиме, что менее удобно, но может быть полезно при отсутствии графического менеджера (обычно такое может быть на

системах Linux, собираемых самостоятельно). Консольный режим работы является на текущий момент единственно возможным для системы android (как раз об этом ограничении упоминалось ранее).

2.3 О выборе алгоритмов и способах шифрования

Как уже было отмечено ранее наиболее часто используемыми и распространенными алгоритмами шифрования для симметричных и асимметричных криптосистем являются AES и RSA соответственно, поэтому именно они и были выбраны для шифрования данных в реализованной системе.

Однако алгоритм RSA используется только для того, чтобы зашифровать и передать секретный ключ, который будет использоваться алгоритмом AES (об этом более подробно будет рассказано в пункте 2.5). Так было сделано потому, что RSA работает значительно медленнее, чем AES. Значительная разница в скорости работы обусловлена тем, что алгоритм AES использует в своей работе в основном перестановки и сдвиги, которые не являются вычислительно затратными операциями, в то время как RSA использует в своей работе возведение в степень, умножение и взятие остатка по некоторому модулю больших чисел, что даже для современных компьютеров является довольно трудной задачей. Результаты сравнения скоростей для шифрования ста сообщений и ста файлов приведены в таблице 1.

Таблица 1 - сравнение скоростей работы алгоритмов AES и RSA

Алгоритм\тип сообщения	Текст (10 – 20 Байт)	Файл (10-20 КБайт)
RSA (полное время, сек)	2.3835	354.8178
RSA (среднее время, сек)	0.0238	3.5481
AES (полное время, сек)	0.0648	0.0717
AES (среднее время, сек)	0.0006	0.0007

Из этой таблицы видно, что RSA работает в десятки раз медленнее чем AES для небольших текстовых сообщений и в тысячи раз медленнее для относительно маленьких файлов. На практике же файлы, передающиеся по сети, весят десятки или даже сотни мегабайт, и шифровать их алгоритмом RSA плохая идея.

2.4 Описание работы серверной части системы

Для начала опишем принцип работы сервера. Как только сервер запустился в первый раз предполагается, что он всегда будет активен. При первом запуске сервер проводит начальную инициализацию, т.е. создает две базы данных и все необходимые таблицы в них, одну для хранения клиентских сообщений, другую для хранения списка клиентов. После инициализации сервер переходит в режим ожидания подключения клиентов.

Подключение клиента происходит следующим образом: как только сервер принимает входящее подключение он устанавливает с клиентом ssl соединение, для того, чтобы передать ему ключ симметричного шифрования, который представляет из себя 32 случайно сгенерированных байта и будет использован в дальнейшем. После передачи ключа ssl соединение закрывается, и канал связи снова становится не защищенным. Далее сервер создает новый поток, в котором будет обслуживаться только что подключенный клиент, а текущий поток продолжает ожидать новых входящих подключений.

Обслуживающий клиента поток всегда находится в состоянии ожидания какого-либо сообщения от клиента. Сообщение может быть одного из двух видов: сообщение, предназначенное для пересылки другому пользователю или сообщение, предназначенное непосредственно серверу, то есть некоторая команда. Первым сообщением от клиента сервер ожидает получить команду – запрос аутентификации, если он получает какие-то другие сообщения, то они просто игнорируются до тех пор, пока клиент не будет аутентифицирован. После успешной аутентификации клиента сервер будет выполнять любые его допустимые команды и пересылать сообщения другим клиентам.

Приведем список допустимых команд, которые выполняет сервер и их описание:

- Регистрация – клиент прислал данные для регистрации, логин и пароль (а также некоторые другие данные, о которых мы поговорим, когда будем рассматривать работу клиента) будут занесены в базу данных на сервере, если не произойдет никакой ошибки, например, если клиент попытается зарегистрироваться с логином, который уже занят, после чего клиенту присвоится идентификационный номер и будет отправлен ему обратным сообщением. Логин и пароль всегда приходят в зашифрованном виде.
- Вход – клиент прислал логин и пароль, сервер проверит есть ли такая пара в его базе данных и в зависимости от результата проверки клиент может получить следующие ответы:
    1. Логин, с которым пытается войти в систему клиент не найден в базе данных клиентов.
    2. Пароль, с которым пытается войти в систему клиент не соответствует логину

3. Сообщение об успехе, в случае если пара логин/пароль была найдена сервером в своей базе данных.

- Удаление аккаунта – клиент прислал запрос на удаление своего аккаунта из базы данных клиентов. Сервер найдет логин текущего клиента и удалит его из базы данных вместе с остальной информацией об этом клиенте, однако история его сообщений удалена не будет, т.к. она может потребоваться другим пользователям, с которыми он вел диалог.
- Выход из аккаунта – клиент вышел из своего аккаунта, сервер снова ожидает сообщение с запросом аутентификации, остальные сообщения игнорируются
- Поиск другого клиента по логину – клиент прислал логин клиента, которого хочет добавить в список своих контактов. Если сервер находит такой логин в своей базе данных, то ответным сообщением отправляет идентификационный номер, соответствующий этому логину. Если сервер такого логина не нашел, то в ответ будет отправлено сообщение об ошибке.
- Получение истории сообщений с другими пользователями – как только пользователь успешно проходит аутентификацию сервер высылает ему историю его сообщений, с другими пользователями, хранящуюся в базе данных.
- Создание прямого соединения между клиентами – клиент прислал запрос на установление прямого соединения, чтобы общаться в обход сервера. Об установлении такого соединения будет рассказано чуть позже.
- Обмен ключами шифрования между клиентами – клиент прислал два одинаковых симметричных ключа шифрования, зашифрованных двумя разными способами, зачем это требуется будет объяснено при подробном рассмотрении работы пользовательского приложения. Сервер сохранит эти два ключа в базу данных и присвоит им определенный номер (идентификатор сессии), по которому их можно будет найти в дальнейшем. Один из ключей будет перенаправлен клиенту - получателю.
- Завершение соединения со стороны клиента – клиент закрыл приложение. Сервер сразу прервет соединение со своей стороны.

При получении от клиента сообщения, предназначенного другому клиенту, сервер проверяет является ли это сообщение зашифрованным, и если да, то просто пересылает его получателю, не применяя к нему никаких действий, так как ему не известен ключ, которым его зашифровали и подписали (этот ключ известен только отправителю и получателю и отличается от тех которые сервер высылал им при подключении), если же сообщение не является зашифрованным, то оно обязательно является подписанным тем ключом, который сервер выслал клиенту в начале обслуживания, так как клиенту-получателю данный ключ не известен, то сервер проверяет, что сообщение не было повреждено по пути от

клиента-отправителя к серверу и, если все в порядке, то сервер пере подписывает сообщение ключом известным ему и клиенту-получателю, после чего отправляет сообщение этому клиенту.

Наглядная схема работы серверного приложения приведена в виде блок-схем на рисунках 9 и 10.

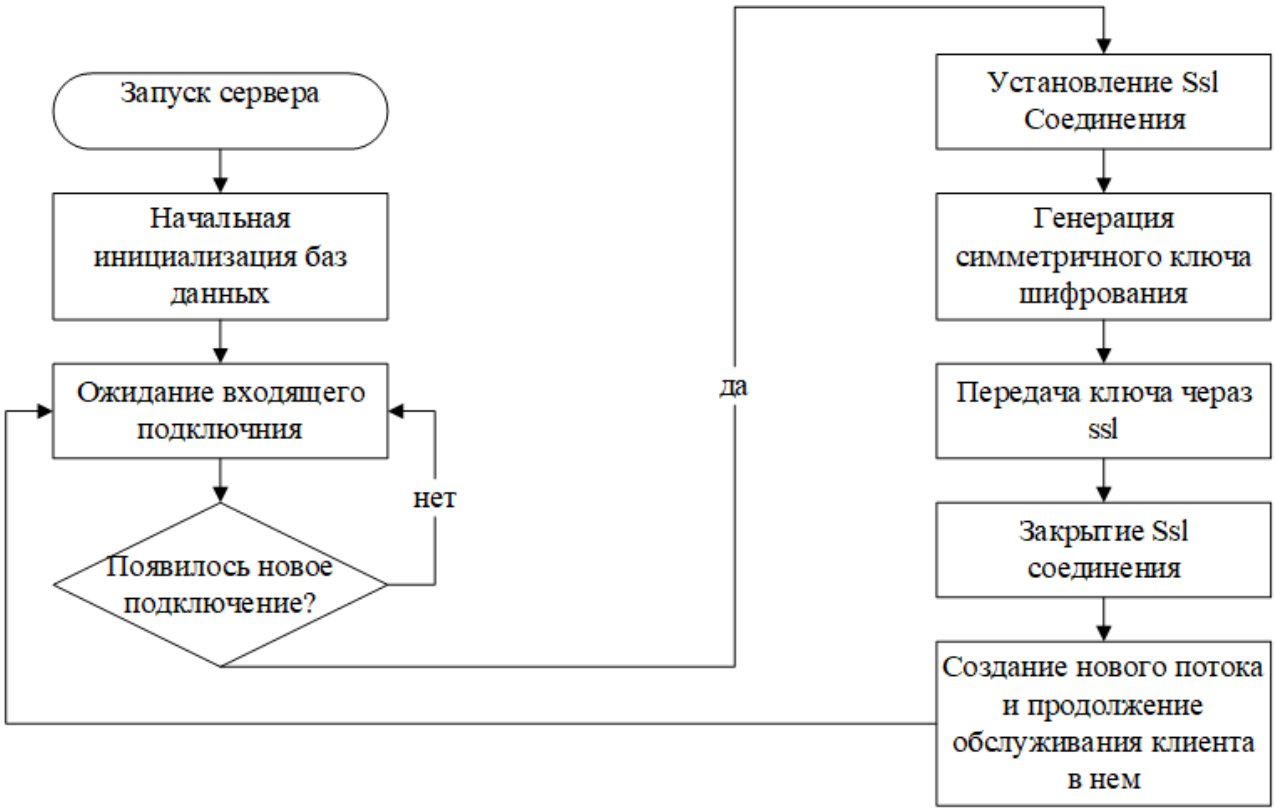

Рисунок 9 - блок-схема работы серверного приложения

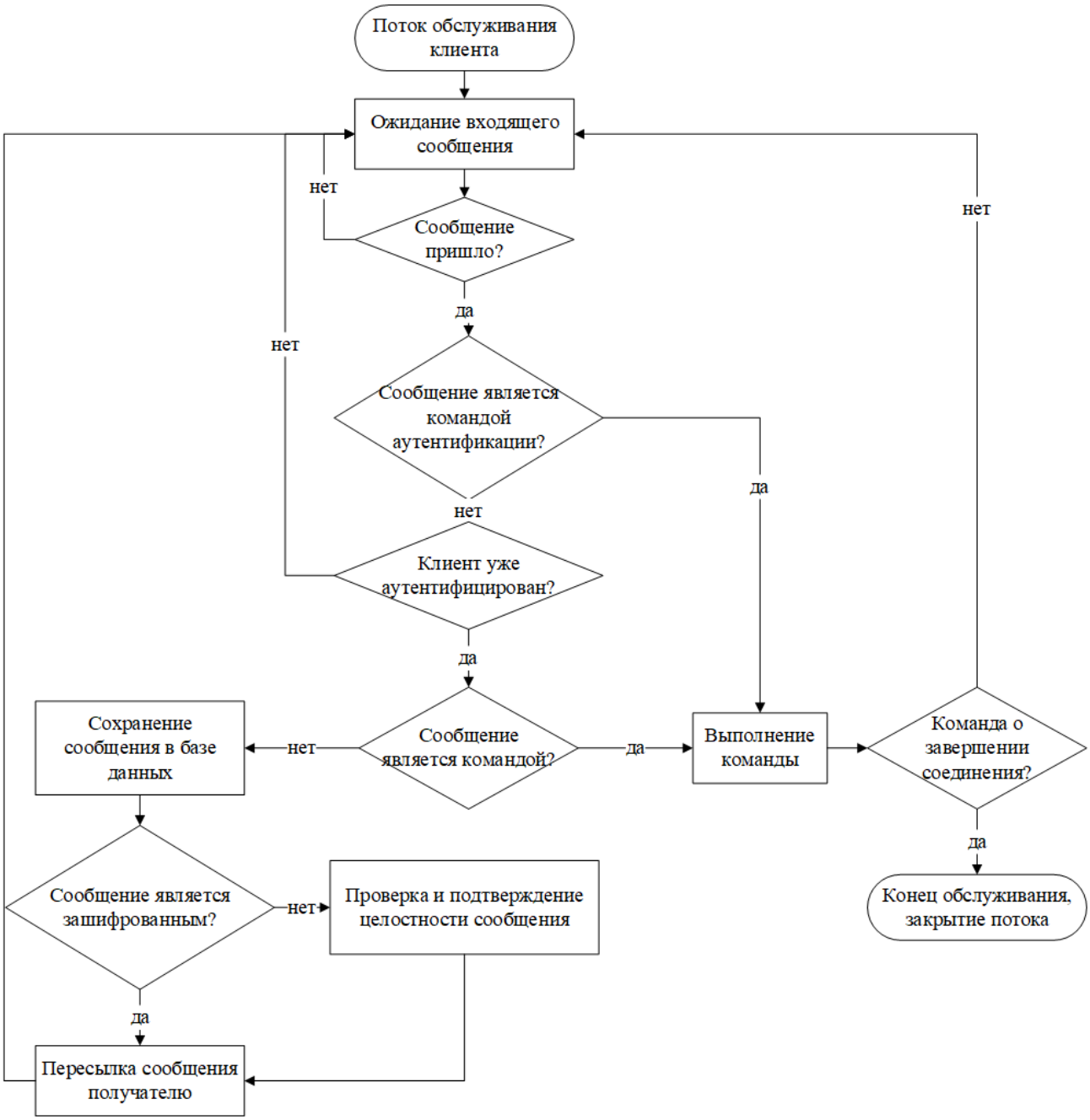

Рисунок 10 - блок-схема работы потока обслуживания клиента на сервере

2.4.1 Хранение данных на сервере

Теперь поговорим подробнее о том, как и какие данные сервер сохраняет в своих базах данных. Всю необходимую информацию сервер хранит не в исходном виде, а преобразовывает в байты. Это было сделано для того, чтобы, во-первых, сократить используемое дисковое пространство, а во-вторых потому, что большинство данных требуется пересылать клиентам, т.е. их все равно пришлось бы преобразовывать к набору байт.

Для начала рассмотрим сведения, которые сервер сохраняет о клиенте. Для сохранения данных о клиенте сервер использует базу данных состоящую из одной таблицы, которая называется user_list и содержит следующие поля:

- uid – идентификационный номер, присваиваемый сервером клиенту в момент регистрации.
- login – логин клиента в виде набора байт.
- hashed_password – хеш клиентского пароля и специальной добавки, называемой солью, что это такое и зачем это нужно будет пояснено далее.
- saltl – часть «соли», приписываемая паролю слева в момент хеширования.
- saltr – часть «соли», приписываемая паролю справа в момент хеширования.
- public_key – открытый ключ пользователя для ассиметричного шифрования.

Список записей в таблице может выглядеть как показано на рисунке 11 (изображение разбито на 2 части левую и правую):

uid	login	hashed_password
1	117,115,101,114,49	54,66,248,94,167,16,2,26,91,54,171,54,150,80,91,85,26,193,35,67,147,48,42,179,109,105,40,88,200,242,19,209
2	117,115,101,114,50	145,73,117,127,244,208,148,147,232,176,58,156,226,185,33,173,179,56,46,165,138,209,132,241,166,227,211,79,194,160,177,170
3	117,115,101,114,51	124,242,121,191,195,32,147,29,194,20,81,19,251,11,65,41,232,65,40,29,113,91,43,249,203,39,33,131,66,246,227,111
4	117,115,101,114,52	21,40,197,23,147,72,138,175,194,145,168,127,190,88,176,212,76,166,254,242,157,75,6,247,210,252,52,153,130,170,154,245
5	97,100,109,105,110	2,46,34,25,119,160,211,248,121,161,148,25,76,50,120,194,226,112,76,10,54,65,140,215,64,77,198,177,163,32,64,203

saltl	saltr	public_key
202,72,249,217,7,83,195,152	78,167,205,116,197,137,122,213	45,45,45,45,45,66,69,71,73,78,32,80,85,66,76,73,67,32,75,69,89,45,4
43,132,172,156,233,96,106,113	66,159,83,152,190,74,3,239	45,45,45,45,45,66,69,71,73,78,32,80,85,66,76,73,67,32,75,69,89,45,4
255,39,17,78,29,78,44,56	100,109,185,186,153,216,129,240	45,45,45,45,45,66,69,71,73,78,32,80,85,66,76,73,67,32,75,69,89,45,4
192,152,23,159,141,112,158,33	177,18,117,117,72,110,238,63	45,45,45,45,45,66,69,71,73,78,32,80,85,66,76,73,67,32,75,69,89,45,4
204,89,253,4,45,133,246,8	128,67,218,235,62,243,207,44	45,45,45,45,45,66,69,71,73,78,32,80,85,66,76,73,67,32,75,69,89,45,4

Рисунок 11 - пример базы данных паролей левая часть(сверху) и правая (снизу).

Разберемся для чего нужно «солить» пароли перед помещением их в базу данных. Хранить пароли в исходном виде просто глупо, так как если кто-либо сумеет получить доступ к базе данных на сервере, то он сможет беспрепятственно получить доступ и к любому интересующему его аккаунту. Для того, чтобы этому помешать обычно сохраняют не сам пароль, а только его хеш, тогда тому, кто получит доступ к базе данных уже не удастся так просто получить доступ к нужному аккаунту, так как придется перебирать все возможные пароли и сравнивать их хеш с тем, что хранится в базе данных, что может потребовать очень много времени.

Однако на сегодняшний день и этот метод не считается достаточно безопасным, так как в интернете существуют так называемые «радужные таблицы». Радужная таблица – это такая таблица, в которой сохраняются пароли соответствующие определенным хешам. Таким образом, имея хеш пароля и выполнив его поиск по радужной таблице, можно получить список паролей, которым соответствует этот хеш при разных алгоритмах хеширования. В таком случае может помочь так называемая «соль». Соль – это дополнительная строка, сгенерированная случайным образом, которая приписывается к паролю и Хешируется вместе с ним. Из полученного таким образом Хеша по радужной таблице пароль уже не восстановишь. Зная соль и выходной Хеш, злоумышленник обречен на полный перебор пароля.

Рассмотрим теперь в каком виде сервер хранит различные клиентские сообщения. Зашифрованные и не зашифрованные сообщения хранятся в различных таблицах. История сообщений клиентов с идентификационными номерами a и b сохраняется в таблице с названием message_history_a_b, или secret_message_history_a_b – если это зашифрованные сообщения.

Таблица не зашифрованных сообщений содержит следующие поля:

- sender_id – идентификационный номер отправителя
- receiver_id - идентификационный номер получателя
- message - сообщение
- message_type – тип сообщения (текстовое или файл)

В таблице зашифрованных сообщений хранятся все те же самые поля, что и для не зашифрованных сообщений, но помимо них, есть и несколько других полей:

- session_id – номер секретной сессии
- message_tag, message_nonce – данные необходимые для расшифровки сообщения

Каждый раз при передаче секретных сообщений клиенты должны договориться об известном только им ключе, с помощью которого они будут шифровать свои данные. При установлении клиентами нового ключа сервер получает два зашифрованных ключа, которые он должен сохранить, для того чтобы клиенты в будущем смогли расшифровать эти сообщения. Более того все сообщения, которые передаются между пользователями с использованием этого ключа должны быть каким-то образом ассоциированы с ним. Поле session_id как раз и указывает какому ключу соответствуют текущие сообщения. Чтобы получить ключи зная session_id сообщений серверу необходимо обратиться к еще одной таблице, называемой encrypted_session_keys_a_b, где a и b – идентификационные номера пользователей. В этой таблице содержатся такие поля:

- session_id – номер секретной сессии
- key_encrypted_by_first_id, key_encrypted_by_second_id – ключи для текущей сессии.

Пример сообщения содержащегося в таблице не секретных сообщений представлен на рисунке 12:

sender_id	receiver_id	message	message_type
2	1	103,104,98,100,116,110	1

Рисунок 12 - пример записи в таблице не секретных сообщений

Вся необходимая информация, касающаяся хранения данных сервером, была нами рассмотрена. Теперь перейдем к завершающей части рассмотрения работы сервера и поговорим об установлении соединения напрямую между клиентами.

2.4.2 Прямое соединение клиентов

Стандарт IP версии 4 насчитывает всего чуть более четырех миллиардов интернет адресов. В современном мире этих адресов уже давно не хватает на всех. Проблему нехватки адресов призван решить стандарт IP версии 6, однако внедрение этого стандарта происходит довольно медленно, а доступ в интернет людям нужен здесь и сейчас. Поэтому был придуман NAT (Network Address Translation – преобразование сетевых адресов).

Локальные сети внутри дома, организации или города, могут использовать огромное количество IP адресов. NAT позволяет это огромное количество адресов преобразовать в один. Преобразование адреса методом NAT может производиться почти любым маршрутизирующим устройством, например, роутером. Наиболее популярным является SNAT, суть механизма которого состоит в замене адреса источника при прохождении интернет пакета в одну сторону и обратной замене адреса назначения в ответном пакете. Наряду с адресами источник/назначение могут также заменяться номера портов источника и назначения.

Принимая пакет от локального компьютера, роутер смотрит на IP-адрес назначения. Если это локальный адрес, то пакет пересылается другому локальному компьютеру. Если нет, то пакет надо переслать наружу в интернет. Но ведь обратным адресом в пакете указан локальный адрес компьютера, который из интернета будет недоступен. Поэтому роутер «на лету» транслирует (подменяет) обратный IP-адрес пакета на свой внешний (видимый из интернета) IP-адрес и меняет номер порта (чтобы различать ответные пакеты, адресованные разным локальным компьютерам). Комбинацию, нужную для обратной подстановки, роутер сохраняет у себя во временной таблице. Через некоторое время после того, как клиент и сервер закончат обмениваться пакетами, роутер сотрет у себя в таблице соответствующую запись.

Пример работы устройства NAT показан на рисунке 13.

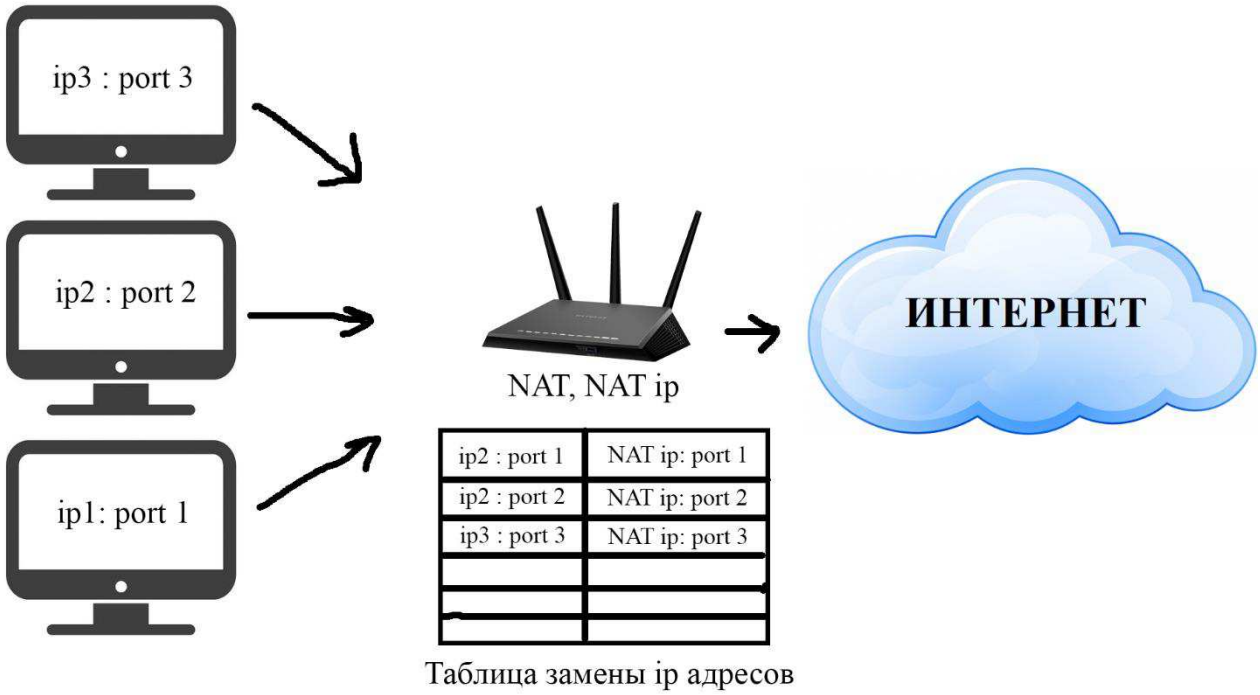

Рисунок 13 - схема работы устройства NAT

Несмотря на всю пользу NAT, он привносит также и некоторые ограничения в виде того, что из сети, находящейся за пределами NAT нет возможности подключиться к устройству, находящемуся «за» NAT, не прибегая к некоторым хитростям. Наиболее часто срабатывающий и хорошо описанный метод установки прямого соединения называется Hole punching (пробитие дыр).

«Пробитие» TCP-дыры позволяет двум клиентам установить прямой одноранговый TCP-сеанс с помощью известного сервера, даже если оба клиента находятся за NAT. Hole punching предполагает, что два клиента, A и B, уже установили соединение с сервером S. Когда клиент присылает запрос на установку прямого соединения с другим клиентом, сервер S записывает два адреса для этого клиента: пару (IP-адрес, порт), которую клиент считает, что использует для общения с S, т.е. его адрес внутри локальной сети и пару (IP-адрес, порт), которую сервер наблюдает при получении клиентского сообщения. Мы назовем первую пару приватным адресом клиента, а вторую публичным адресом клиента. Если клиент не находится за NAT, то его приватный и публичный адреса должны совпадать.

Основная практическая проблема для приложений, желающих реализовать Hole punching по протоколу TCP, - это не проблема протокола, а проблема интерфейса сокетов. Поскольку стандартный интерфейс сокетов Беркли был разработан вокруг парадигмы клиент / сервер, он позволяет использовать сокет протокола TCP для инициации исходящего соединения или для прослушивания входящих соединений, но не оба действия сразу. Кроме того, сокеты TCP обычно

имеют однозначное соответствие номерам портов TCP на локальном компьютере: после того, как приложение привязывает один сокет к определенному локальному порту TCP, попытка привязать второй сокет к тому же порту TCP завершается неудачей.

Однако для того, чтобы «пробить дыру», мы должны использовать один локальный порт TCP, чтобы прослушивать входящие TCP-соединения и одновременно инициировать несколько исходящих TCP-соединений. К счастью, все основные операционные системы поддерживают специальный параметр сокета TCP, обычно называемый SO_REUSEADDR, который позволяет приложению связывать несколько сокетов с одной и той же локальной конечной точкой, если этот параметр установлен на всех задействованных сокетах. Системы BSD представили опцию SO_REUSEPORT которая контролирует повторное использование порта отдельно от повторного использования адреса; в таких системах должны быть установлены обе эти опции.

Предположим, что клиент A хочет установить соединение с клиентом B. Мы предполагаем, что A и B уже установили соединения с известным сервером S. Протокол hole punching работает следующим образом:

1. Клиент A использует свой активный сеанс с S, чтобы попросить S помочь подключиться к B.
2. S высылает клиенту A публичный и приватный адреса B, и в то же время отправляет B публичный и приватный адреса A.
3. С тех же портов, которые A и B использовали для связи с S, каждый из них начинает выполнять попытки соединения с приватным и публичным адресами второго, одновременно прослушивая входящие соединения.
4. A и B дожидаются успешной попытки исходящего соединения и/или появления входящего соединения. Если одна из попыток исходящего соединения завершается неудачно из-за сетевой ошибки, компьютер просто повторяет эту попытку соединения после небольшой задержки (например, одной секунды), до достижения максимального количества попыток.
5. Когда установлено соединение, клиенты аутентифицируют друг друга, чтобы убедиться, что они подключены к кому хотели. Если аутентификация не удалась, клиенты закрывают это соединение и продолжают ждать, пока другие попытки соединения добьются успеха. Клиенты используют для общения первое успешно аутентифицированное соединение, полученное в результате этого процесса.

Рассмотрим сценарий общего случая, в котором клиенты A и B находятся за разными NAT, как показано на рисунке. Попытки соединения A и B с приватными адресами друг друга провалятся или приведут к подключению к неправильному хосту. Важно, чтобы приложения аутентифицировали свои одноранговые сеансы из-за вероятности ошибочного подключения к случайному

хосту в локальной сети, который, может иметь тот же приватный IP-адрес, что и нужный хост на удаленном компьютере. Процесс работы алгоритма hole punching представлен на рисунках 14-16.

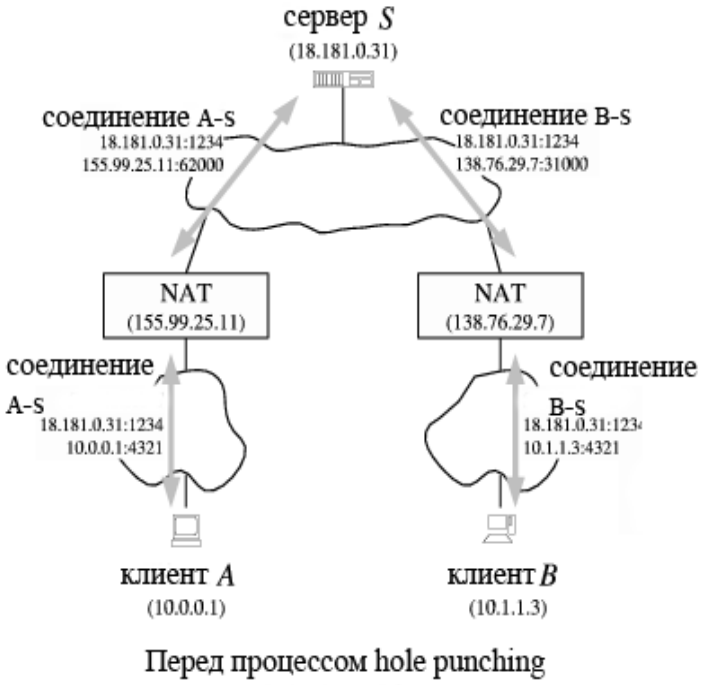

Рисунок 14 - схема соединений перед началом процесса hole punching

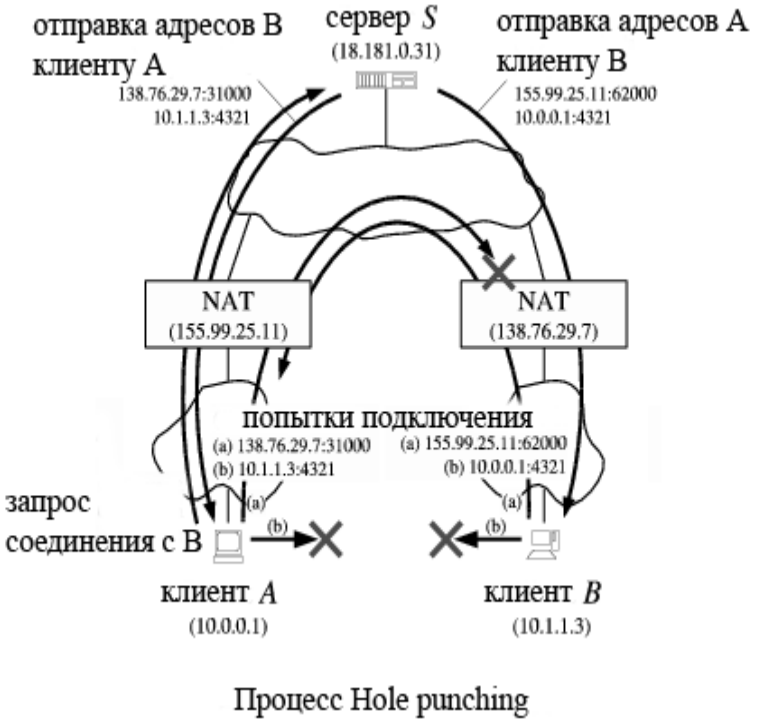

Рисунок 15 - схема соединений во время процесса hole punching

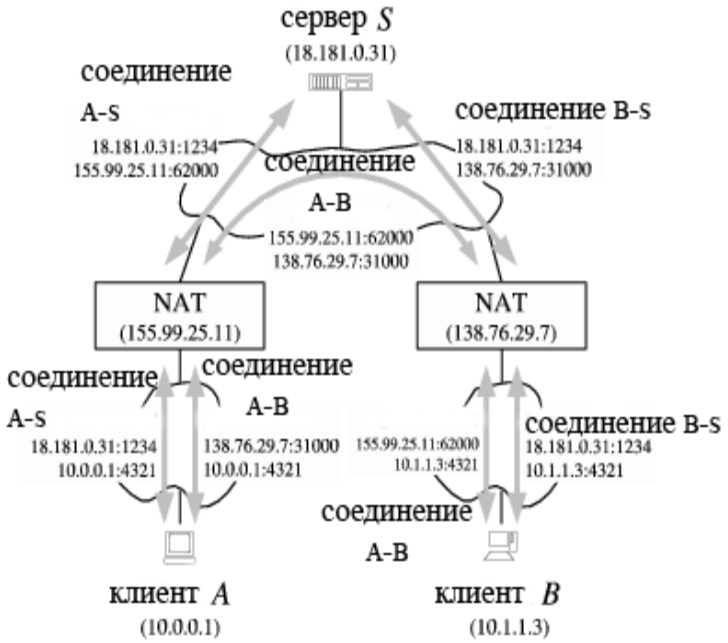

Рисунок 16 - схема соединений после процесса hole punching

Попытки исходящего подключения клиентов к публичным адресам друг друга приводят к тому, что соответствующие NAT открывают новые «дыры», обеспечивая прямую связь TCP между A и B. Если NAT является «дружественным» к методу hole punching, то между клиентами автоматически создается новое одноранговое соединение. Если первая попытка соединения с B со стороны A достигает устройство NAT клиента B перед тем, как первая попытка соединения с A со стороны B достигает своего NAT, тогда NAT клиента B может интерпретировать соединение со стороны A как попытку не запрошенного входящего соединения и отбросить его. Однако соединение со стороны B при этом должно пройти, потому что устройство NAT клиента A видит это соединение как часть исходящего сеанса которое было начато его клиентом.

К сожалению, некоторые NAT устройства не допускают соединения таким образом, поэтому установить прямую связь между клиентами удастся не всегда.

2.5 Описание работы клиентской части системы

Перейдем к рассмотрению деталей работы клиентского приложения.

При запуске приложение одновременно запускает процесс подключения к серверу, и процесс инициализации в разных потоках. Процесс подключения к серверу мы уже разобрали ранее, поэтому рассмотрим сразу процесс инициализации пользователя. Первым делом проверяется наличие у пользователя пары ключей, открытого и закрытого, с помощью которых будет производиться ассиметричное шифрование. При отсутствии этих ключей у пользователя, они генерируются и сохраняются на устройстве. После успешного соединения с сервером и аутентификации, пользовательское приложение инициализирует, в случае необходимости, базы данных которые содержат список контактов пользователя и сообщения, которые не сохраняются на сервере, то есть полученные в ходе прямого соединения с другим пользователем, создавая там необходимые таблицы. Обычно эта инициализация требуется только при первом запуске приложения, однако если пользователь вдруг удалил намеренно или случайно нужные приложению данные, они будут создаваться заново.

После локальной инициализации клиентское приложение посылает серверу запрос на получение истории сообщений из его базы данных, одновременно с этим подгружая сообщения и список контактов из своей базы данных.

Когда все необходимые данные получены, приложению остается только реагировать на действия, совершаемые пользователем и выполнять команды, приходящие от сервера или реагировать на сообщения других пользователей.

Самое важное действие пользователя, на которое нужно реагировать приложению это, конечно же, отправка сообщения. Самое важное в отправке

сообщения, если оно производится клиенту напрямую, это сохранить его в локальной БД. Если сообщение отправляется в секретной переписке, то его необходимо зашифровать, процесс передачи секретного сообщения будет рассмотрен чуть позже.

Команды, приходящие от сервера, сводятся к ответам сервера на запросы, полученные от пользователя. В основном они уведомляют клиента об успешном или неудачном завершении того или иного действия, например, аутентификации. Однако некоторые сообщения от сервера содержат данные, необходимые для работы приложения, рассмотрим эти данные подробнее.

- Данные о публичном и приватном адресе другого пользователя – приватный и публичный ip адреса и порты клиента необходимые для установления прямого соединения с ним.
- Данные приходящие при добавлении человека в список контактов – успешном выполнении запроса на добавление человека в список контактов сервер высылает идентификационный номер этого человека и его открытый ключ ассиметричного шифрования.
- Сообщения из базы данных сервера – сообщение может быть двух видов зашифрованное или нет, а также оно может быть получено напрямую от другого пользователя или путем пересылки сообщения с сервера. Если сообщение было получено от пользователя напрямую, то его необходимо сохранить в локальной базе данных, т.к. сервер данного сообщения у себя сохранить не мог. Если сообщение было не зашифровано, то оно просто отображается пользователю, если же оно было зашифровано, то перед отображением его необходимо расшифровать, этот процесс рассмотрим подробнее далее.
- Зашифрованный ключ из базы данных для расшифровки истории секретных сообщений – ключ необходимый для расшифровки секретного сообщения.
- Зашифрованный ключ симметричного шифрования для секретного общения с другим пользователем – ключ который сгенерировал пользователь, желающий вести секретное общение с нами.

Сейчас поговорим более подробно о том, что происходит, когда два пользователя решают начать секретный чат друг с другом: как они обмениваются секретным ключом без раскрытия его серверу, и как восстанавливается история секретных сообщений.

Когда пользователь А начинает секретную переписку с пользователем В ему необходимо выбрать ключ, которым он будет шифровать отправляемые сообщения. Несмотря на то, что он знает открытые ключи всех пользователей из своего списка контактов применять криптографию с открытым ключом не желательно, так как это требует больших затрат времени чем шифрование симметричным ключом, особенно это будет заметно при попытке переслать какой-либо файл. Поэтому приложение генерирует ключ симметричного

шифрования. Для того, чтобы передать ключ пользователю B, пользователь A шифрует его открытым ключом пользователя B с помощью алгоритма RSA, также пользователь A шифрует этот ключ и своим открытым ключом, таким образом у него есть два одинаковых ключа зашифрованных с помощью разных открытых ключей. Именно об этой паре ключей и шла речь, при рассмотрении работы сервера. Сервер сохранит эту пару у себя в базе данных, а также перешлет пользователю B ключ, зашифрованный его открытым ключом. Пользователь B также с помощью алгоритма RSA и своего закрытого ключа расшифрует его и получит ключ симметричного шифрования, для секретного общения с пользователем A. Схема доставки симметричного ключа представлена на рисунке 17.

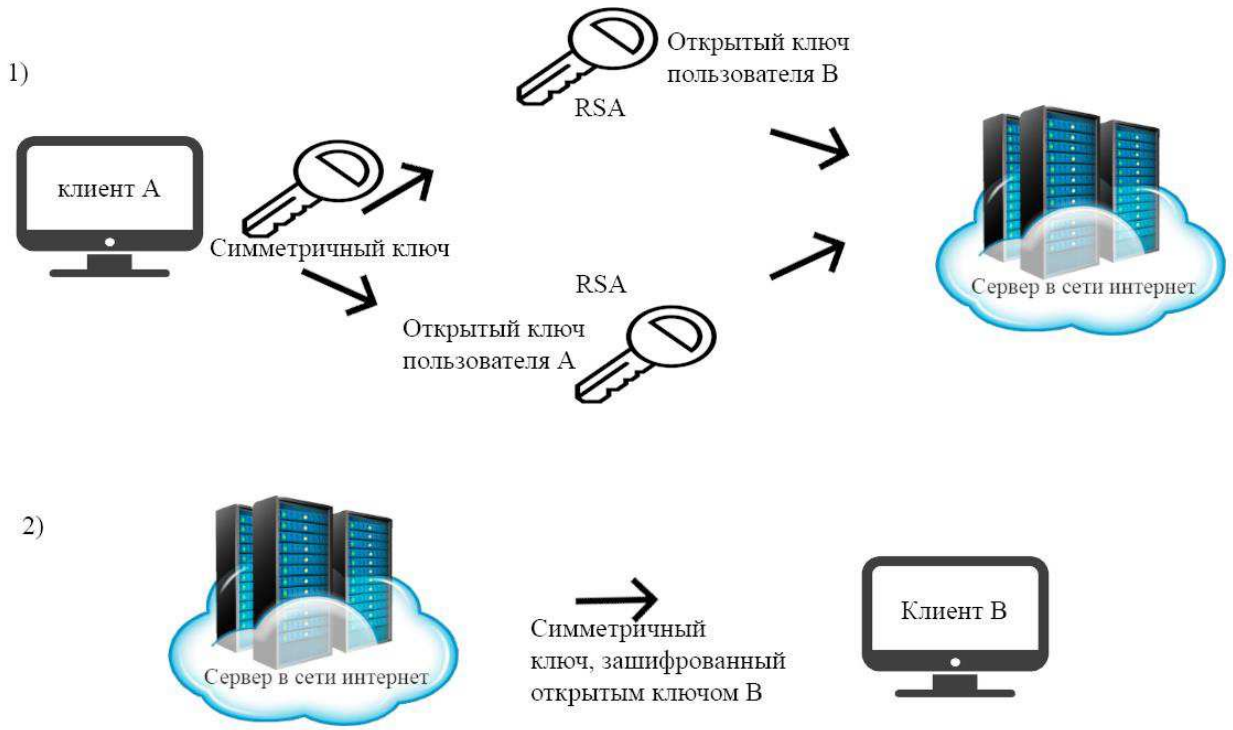

Рисунок 17 - схема установления общего симметричного ключа между пользователями

На данном этапе произведен обмен ключом таким образом, что никто кроме двух конечных пользователей не знает его. Таким образом обеспечивается так называемое сквозное шифрование (E2EE – end to end encryption), оно не позволяет третьим лицам получить доступ к частной переписке.

Как уже говорилось ранее сервер сохраняет все сообщения в исходном виде, поэтому при получении доступа к серверу, секретная переписка также не может быть скомпрометирована.

Теперь нужно понять, как восстановить историю зашифрованных сообщений. Как было рассмотрено ранее, когда пользователь запрашивает историю сообщений ему также приходят и зашифрованные ключи, которые сервер сохранял в своей базе данных. Именно для этого пользователь при начале секретного общения отправлял серверу ключ который был зашифрован не только открытым ключом второго пользователя, но и своим собственным. Теперь при получении данного ключа пользователь расшифровывает его своим закрытым ключом и получая зашифрованные сообщения с сервера может их расшифровать.

В целом работа клиентского приложения устроена так, как показано на блок-схеме, представленной на рисунке 18.

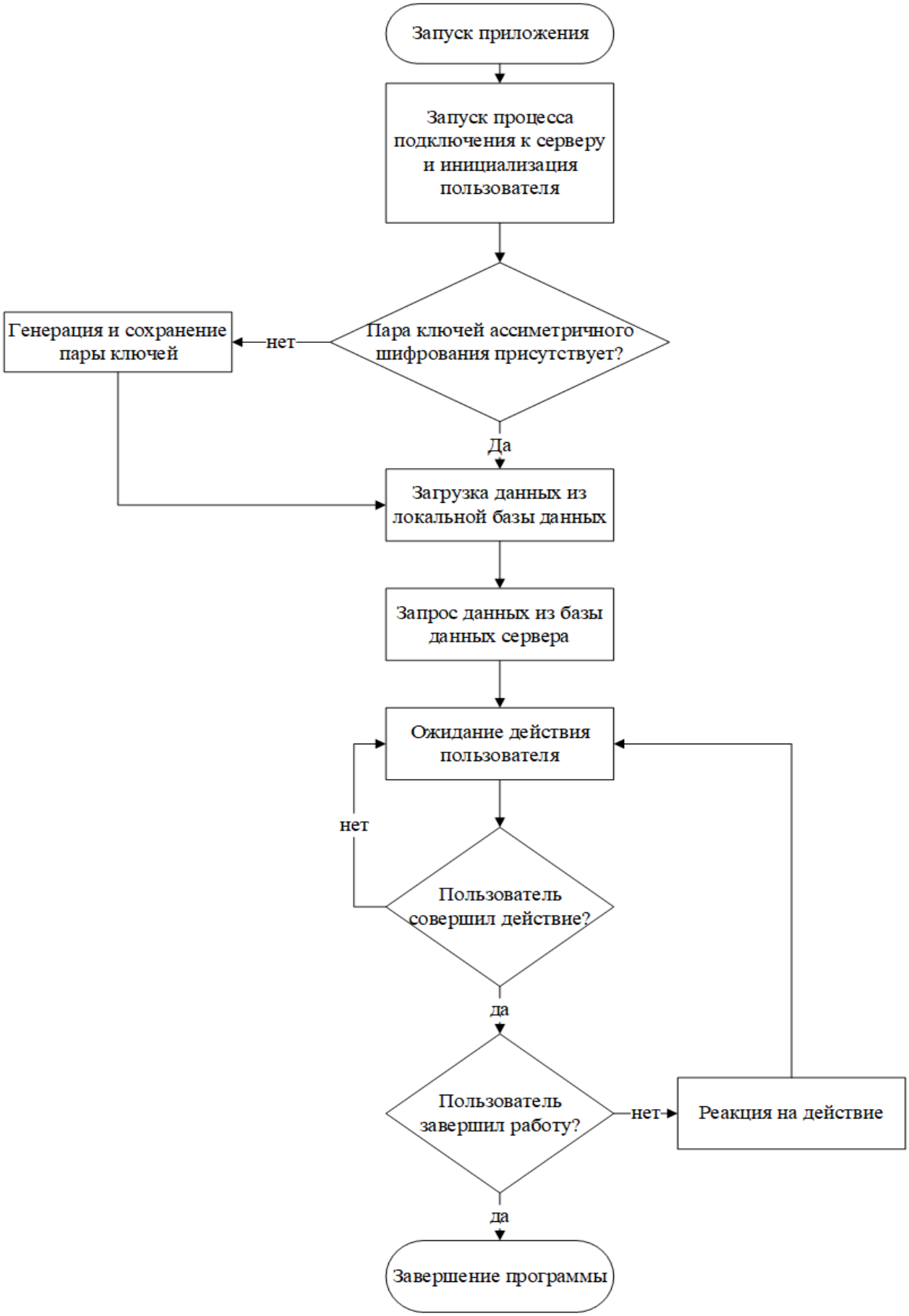

Рисунок 18 - блок схема работы клиентского приложения

Теперь рассмотрим какие данные хранит пользователь на своем устройстве. Пользовательские базы данных устроены намного проще, чем базы данных сервера, чтобы не расходовать много дискового пространства.

Все сообщения в базе данных пользователя хранятся в открытом виде, хранение их в зашифрованном виде не имеет смысла. Все сообщения хранятся в таблицах с названиями вида message_history_id или secret_message_history_id, где id – идентификационный номер второго пользователя. Данные таблицы содержат одинаковые поля:

- message_received – поле хранящее информацию о том, было данное сообщение получено или отправлено
- message – само сообщение
- mes_type – тип сообщения, файл или текст

Пример сообщения из табицы показан на рисунке 19.

message_received	message	mes_type
0	104,106	1

Рисунок 19 - пример записи в базе данных клиентских сообщений

База данных, содержащая список контактов пользователя содержит единственную таблицу friend_list и сохраняет в ней следующую информацию:

- id – идентификационный номер пользователя
- login – логин пользователя
- public_key – открытый ключ пользователя

Пример записи из таблицы friend_list показан на рисунке 20.

id	login	public_key
1	user1	45,45,45,45,45,66,69,71,73,78,32,80,85,66,76,73,67,32,75,69,89,45,45,45,45,45,10,77,73,73,66,73,106,65,78,66,103,107,113,104,107,105,71,57,119,48,6
5	admin	45,45,45,45,45,66,69,71,73,78,32,80,85,66,76,73,67,32,75,69,89,45,45,45,45,45,10,77,73,73,66,73,106,65,78,66,103,107,113,104,107,105,71,57,119,48,6

Рисунок 20 - пример записи в базе данных списков контактов пользователя

3 Описание работы системы

Данная система состоит из нескольких модулей, написанных на языке программирования python, преимущественно в стиле ООП. Приведем краткое описание используемых модулей, для лучшего понимания работы системы.

1. client.py – содержит основной функционал клиентского приложения
2. client_database.py – клиентская часть базы данных. В ней сохраняются список контактов пользователя, а также те сообщения, которые не хранятся на сервере.
3. common_functions_and_data_structures.py – общие функции и структуры данных, которые используются как в клиентской, так и в серверной частях системы.
4. constants.py – список констант
5. database.py – содержит основной класс базы данных, от которого унаследованы классы, находящиеся в модулях client_database.py и server_database.py
6. gui2.py – интерфейс клиентского приложения
7. main.py – работа клиентского приложения начинается в этом модуле. Он связывает между собой интерфейс клиентского приложения из модуля gui2.py и его функционал из модуля client.py.
8. server.py – содержит функционал, необходимый для работы серверной части системы.
9. server_database.py – серверная часть базы данных. В ней сохраняются все сообщения, проходящие через сервер, список пользователей, зарегистрированных в системе вместе с информацией о них, а также некоторая вспомогательная информация нужная для правильной работы системы.

Организация модулей в программе, зависимости между ними и основные классы, которые они содержат отображены на рисунках ниже для клиентского приложения на рисунке 21 и для серверного на рисунке 22. Модули изображены прямоугольниками, а стрелочками изображено отношение включения между ними, т.е. если модуль А включен в модуль Б, то стрелочка выходит из модуля А и входит в модуль Б.

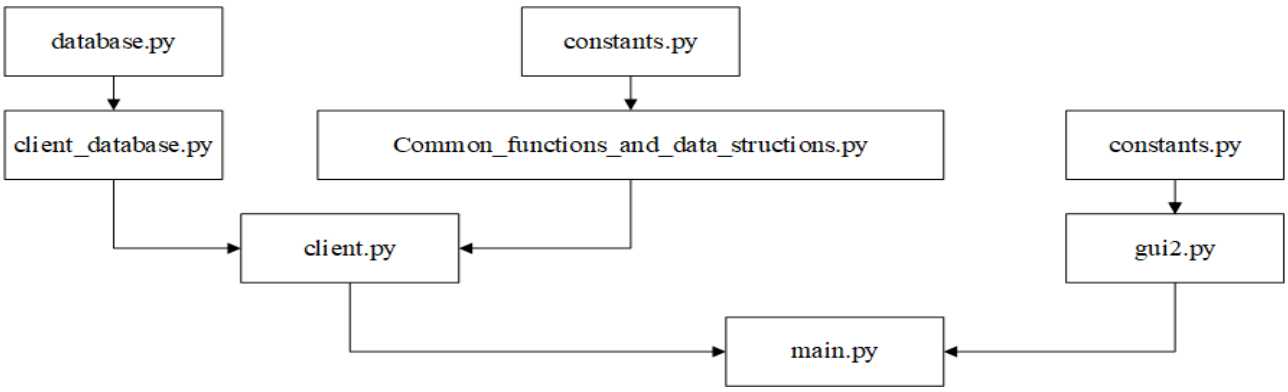

Рисунок 21 - схема организации модулей в клиентском приложении

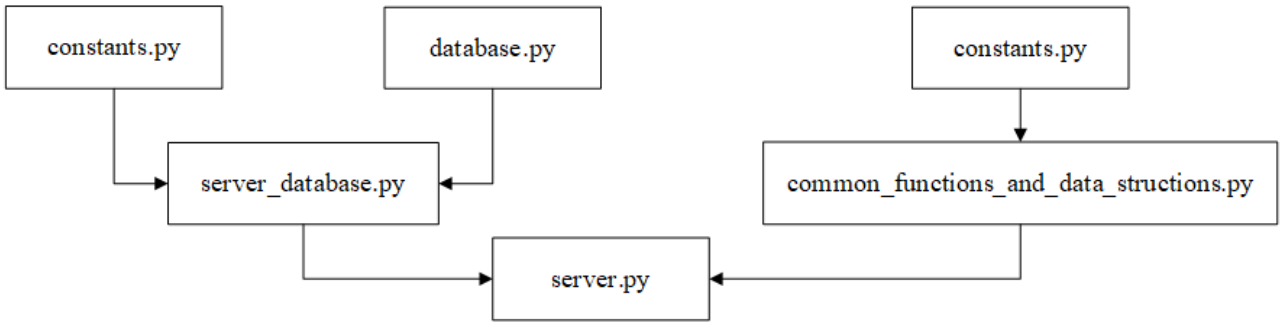

Рисунок 22 - схема организации модулей в серверном приложении

3.1 Системные требования приложения

Запуск серверной части приложения требует около 30 мб оперативной памяти, но затраты памяти увеличиваются с подключением новых клиентов.

Запуск пользовательского приложения в графическом режиме требует около 40 мб оперативной памяти, без графического режима около 30 мб.

3.2 Примеры работы клиентского приложения

При запуске приложения пользователь увидит окно аутентификации, как показано на рисунке 23, для продолжения работы необходимо ввести свои данные, а затем войти или зарегистрироваться, нажав соответствующую клавишу.

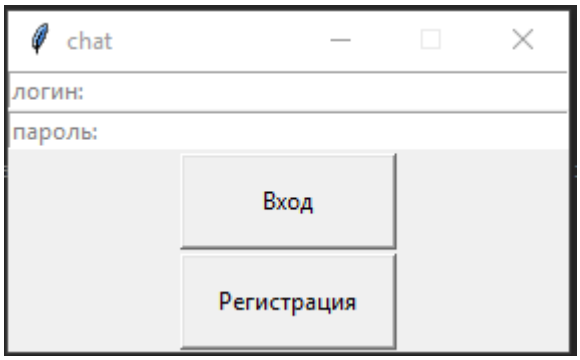

Рисунок 23 - окно аутентификации

При попытке войти с неправильным логином или паролем, пользователь получит одно из предупреждений, как показано на приведенном ниже рисунке 24.

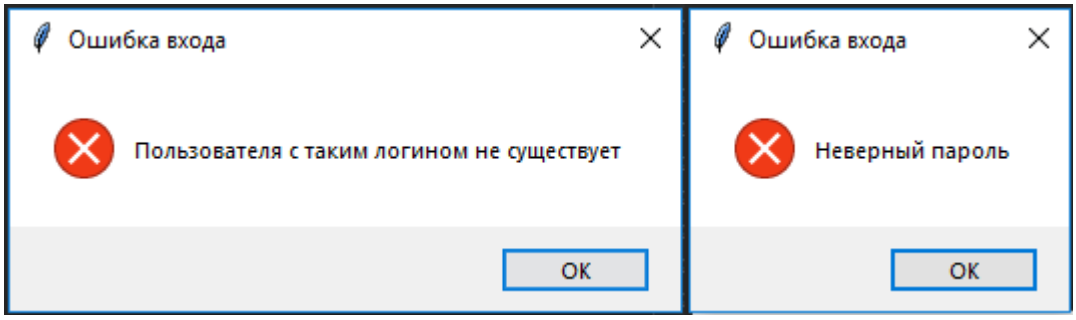

Рисунок 24 - скриншоты ошибок при входе. Несуществующий логин (слева) и неправильный пароль (справа)

При попытке регистрации с уже существующим логином, пользователь получит предупреждение, показанное на рисунке 25, а при попытке входа или регистрации с незаполненными полями логина или пароля предупреждение, показанное на рисунке 26.

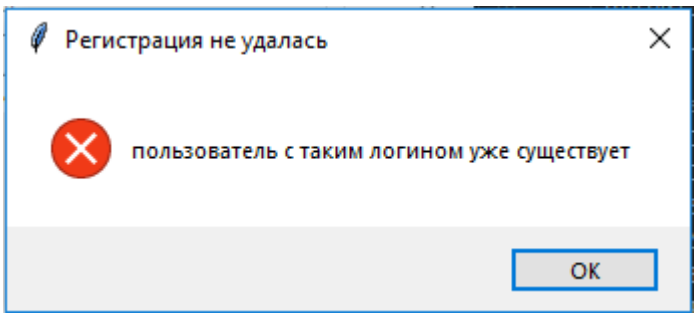

Рисунок 25 - логин, указанный при регистрации занят

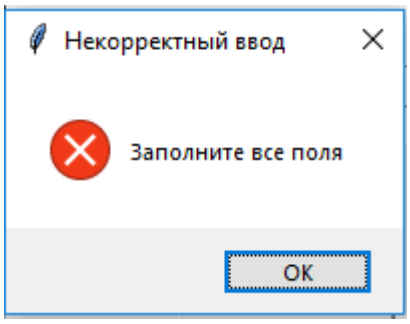

Рисунок 26 - поле ввода логина и/или пароля не заполнено

После успешного прохождения аутентификации пользователю откроется основное окно клиентского приложения, показанное на рисунке 27, в котором

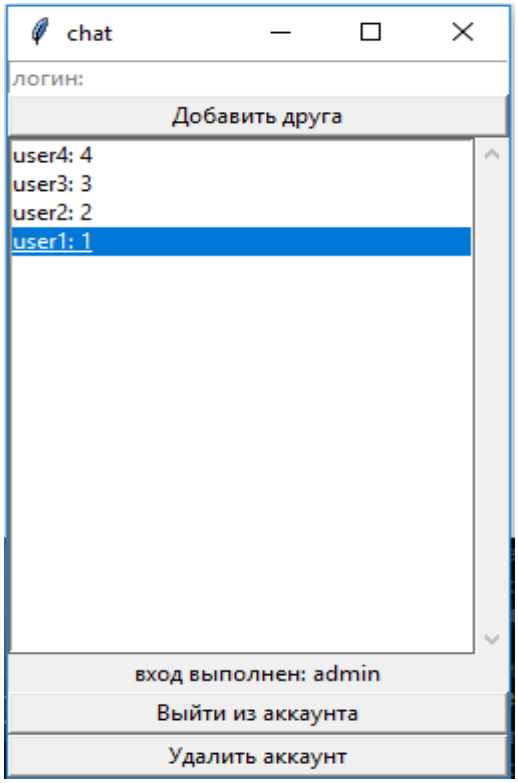

Рисунок 27 - окно списка контактов пользователя

отображается список его контактов. Также в главном окне имеется возможность добавить нового пользователя в этот список, выйти из аккаунта или удалить его.

При выборе пользователя из отображаемого списка открывается окно чата с ним, показанное на рисунке 28. В данном окне можно выбрать один из перечисленных ранее способов передачи сообщений, нажав соответствующую кнопку. Для отправки файлов, нужно нажать на кнопку файл и выбрать его в открывшемся окне выбора файлов, показанном на рисунке.

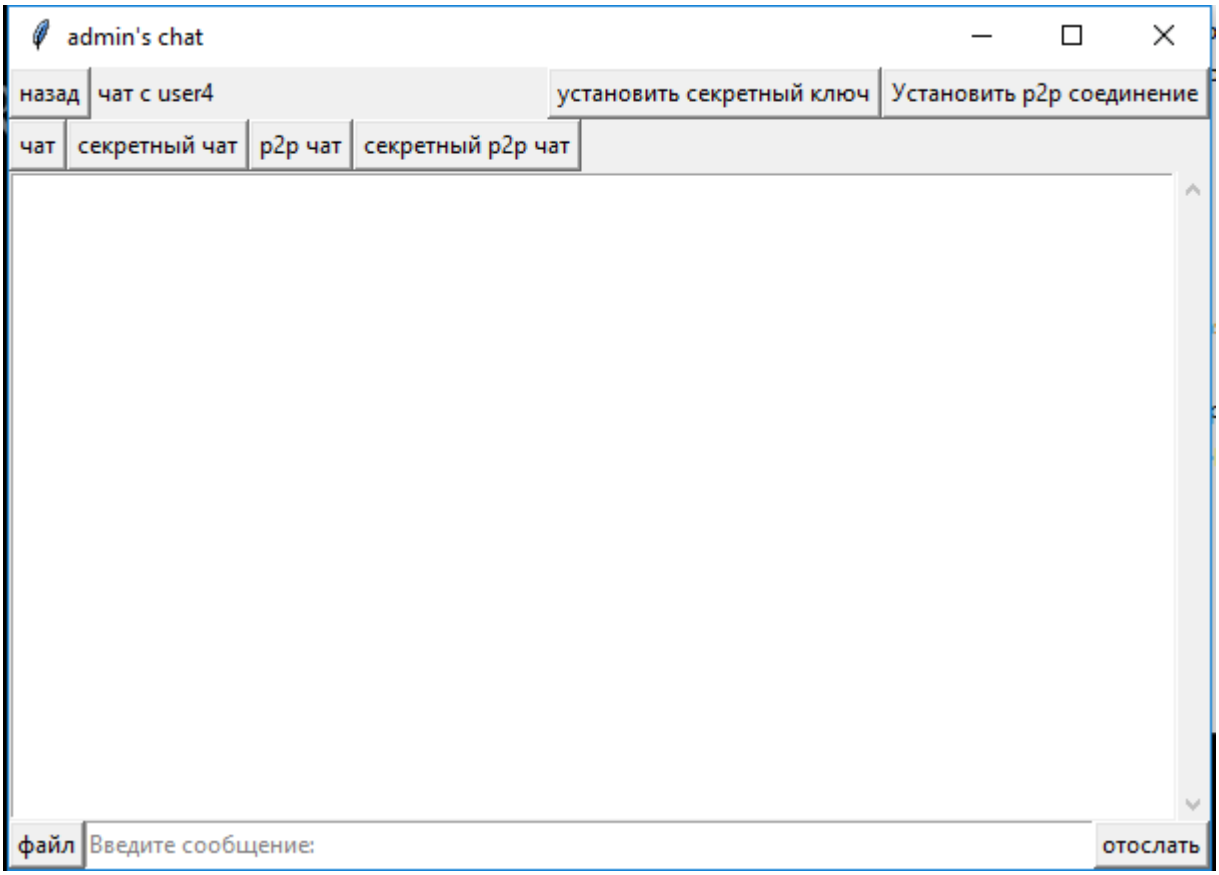

Рисунок 28 - окно чата

В целом интерфейс приложения достаточно прост, поэтому разобраться в нем не составит никакого труда.

На этом рассмотрение работы системы можно считать завершенным.

ЗАКЛЮЧЕНИЕ

В ходе выполнения бакалаврской работы были рассмотрены методы передачи данных по различным сетям, а также методы и технологии защиты передаваемых данных. Для лучшего понимания используемых методов безопасности разобраны основные принципы современной криптографии, рассмотрены алгоритмы как симметричного, так и асимметричного шифрования на которых они построены.

Была реализована система на языке программирования python, состоящая из двух приложений, клиентского и серверного. Пользователи данной системы имеют возможность общаться друг с другом в сети интернет, причем общение может проводиться секретным образом, то есть так, что третьи лица не получат доступ к переписке. Помимо обычных сообщений система может передавать по сети файлы. Все сообщения хранятся в базе данных, при получении доступа к которой третьим лицом, никакая секретная информация все равно не будет скомпрометирована, благодаря тому, что она хранится в зашифрованном виде.

Конечно, данная система не идеальна и имеет свои недостатки, в плане удобства использования, однако в плане защиты данных она не уступает системам, над которыми трудятся целые команды разработчиков в больших компаниях.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Бауэр Ф. Расшифрованные секреты. Методы и принципы криптологии. / Ф. Бауэр – Москва: Мир, 2007. – 550с.

2. Бейли Л. Изучаем SQL / Л. Бейли – СПб.: Питер, 2012. – 592с.

3. Галуев Г.А. Математические основы криптологии: Учебно-методическое пособие. / Г. А. Галуев – Таганрог: изд-во ТРТУ, 2003. – 120с.

4. ГОСТ 28147-89. Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования. – Москва: ИПК Издательство стандартов, 1990. – 28с.

5. Документация по криптографическому модулю для языка Python3 PyCryptodome [Электронный ресурс]. – Режим доступа: https://pycryptodome.readthedocs.io/en/latest/src/api.html - (дата обращения 15.06.2019)

6. Документация по языку программирования Python3 [электронный ресурс]. – Режим доступа: https://docs.python.org/3/library/ - (дата обращения 15.06.2019)

7. Жданов О.Н. Задачник-практикум по криптографическим методам защиты информации.: Учебное пособие /О.Н. Жданов, Ю.Ю. Ушаков – Москва: Национальный открытый университет «ИНТУИТ», 2016. – 384с.

8. Коробейников А.Г. Математические основы криптологии. Учебное пособие / А.Г. Коробейников, Ю.А. Гатчин – СПб.: СПб ГУ ИТМО, 2004. – 106с.

9. Кригель А. SQL. Библия пользователя, 2-е издание / А. Кригель, Б. Трухнов – Москва: Диалектика, 2010. – 752с.

10. Лутц, М. Программирование на Python, 4-е издание: в 2-х т./ М. Лутц - СПб.: Символ-Плюс, 2011. – 992с. – 2 т.

11. Молинаро Э. SQL. Сборник рецептов / Э. Молинаро – СПб.: Символ-Плюс, 2009. – 672с.

12. Нестеренко, А.Ю. Теоретико-числовые методы в криптографии/ А.Ю. Нестеренко – Москва: Моск. гос. ин-т. электроники и математики, 2012. – 224с.

13. Олифер, В. Компьютерные сети. Принципы, технологии, протоколы 5-е издание / В. Олифер, Н. Олифер – СПб.: Питер, 2016. – 992с.

14. Основы криптографии Учебное пособие / А. П. Алферов, А. Ю. Зубов, А. С. Кузьмин, А. В. Черемушкин – Москва: Гелиос АРВ, 2001. - 479с.

15. Таненбаум, Э. Компьютерные сети 5-е издание / Э. Таненбаум, Д. Уэзеролл – СПб.: Питер, 2012. - 960с.

16. Учебный видеокурс «Защита информации» [Электронный ресурс]. - Режим доступа: https://www.youtube.com/playlist?list=PL2jwxGybEFiuQVQtrLPaH7GNB8ak29634 - (дата обращения 15.06.2019)

17. Учебный видеокурс «компьютерные сети и системы телекоммуникаций» [Электронный ресурс]. – Режим доступа: https://www.youtube.com/watch?v=OLFA0soYGhw&list=PLtPJ9lKvJ4oiNMvYbOzCmWy6cRzYAh9B1 - (дата обращения 15.06.2019)

18. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. / Б. Шнайер – Москва: Триумф, 2003. – 806с.

19. Ford B. Peer-to-Peer Communication Across Network Address Translators [Электронный ресурс]/ B. Ford, P. Srisuresh, D. Kegel – Режим доступа: http://bford.info/pub/net/p2pnat/ - (дата обращения 15.06.2019)

20. Raschka S. A thorough guide to SQLite database operations in Python [Электронный ресурс] / S. Raschka – Режим доступа: https://sebastianraschka.com/Articles/2014_sqlite_in_python_tutorial.html - (дата обращения 15.06.2019)

ПРИЛОЖЕНИЕ 1 Описание классов основных компонентов системы

Модуль client.py содержит следующие классы:

- Client – класс, описывающий основную логику пользовательского приложения, содержит методы для инициализации пользовательского приложения, для взаимодействия с ним в консольном и графическом режимах, а также методы, обрабатывающие пользовательские запросы.
- ReceivedMessageManager – класс, обрабатывающий входящие сообщения от других клиентов или сервера.
- Peer2PeerConnector – класс, отвечающий за соединение двух клиентов напрямую.
- Friend – класс, представляющий собой описание другого пользователя, находящегося в списке ваших контактов.

Модуль client_database.py содержит следующие классы:

- ClientMessageDatabase - класс, управляющий базой данных локальных сообщений пользователя.
- ClientUserDatabase – класс, управляющий списком контактов пользователя.

Модуль gui2.py содержит следующие классы:

- PixelSizedButton, PixelSizedLabel, EntryWithTemplateString -
- LoginWindow – класс, отвечающий за установку обработчиков окна входа, и обработку событий, происходящих в этом окне.
- ChatWindow - класс, отвечающий за установку обработчиков окна чата, и обработку событий, происходящих в этом окне.
- FriendListWindow - класс, отвечающий за установку обработчиков окна списка пользователей, и обработку событий, происходящих в этом окне.
- GUI - класс, отрисовывающий необходимые окна на экране, а также связывающий интерфейсную часть пользовательской программы с ее алгоритмической частью, т.е. обрабатывающий события, приходящие из модуля client.py.

Модуль server.py содержит следующие классы:

- Server - класс, реализующий серверную часть системы. Отвечает за обработку всех клиентских запросов.

Модуль server_database.py содержит следующие классы:

- ClientMessageDatabase - класс, управляющий базой данных сообщений пользователей, хранящихся на сервере.
- ClientUserDatabase – класс, управляющий списком пользователей, зарегистрированных в приложении.

Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт математики и фундаментальной информатики
Базовая кафедра вычислительных и информационных технологий

УТВЕРЖДАЮ
Заведующий кафедрой
_____ / В. В. Шайдуров

«17» июня 2019 г.

БАКАЛАВРСКАЯ РАБОТА

Направление 02.03.01 Математика и компьютерные науки

Защищенная кроссплатформенная система обмена сообщениями в сети Интернет

Научный руководитель
кандидат технических наук,
доцент _____ / С.В. Исаев
17.06.19

Выпускник _____ / А.М. Сухих
17.06.19

Красноярск 2019