Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования
Направление подготовки Программная инженерия

РАБОТА ПРОВЕРЕНА
Рецензент,

_____
«____»_____20___г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н., доцент
_____/А.А.Замышляева
«____»_____2017 г.

Сеть обмена сообщениями с использованием протоколов луковичной маршрутизации и UDP

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–090404.2017.158.ПЗ ВКР

Руководитель работы, к.ф.-м.н., доцент кафедры ПМиП
_____/С.М. Елсаков
«____»_____2017 г.
Автор работы
студент группы ЕТ-223
_____/ Д.В. Бобыкин
«____»_____2017 г.
Нормоконтролер, к.э.н., доцент кафедры ПМиП
_____/ Д.А. Дрозин
«____»_____2017 г.

Челябинск, 2017

АННОТАЦИЯ

Бобыкин Д.В. Сеть обмена сообщениями с использованием протоколов луковичной маршрутизации и UDP.– Челябинск: ЮУрГУ, ЕТ-223, 43 с., 13 ил., 2 табл., библиогр. список – 50 наим., 3 прил.

В работе исследованы анонимные сети. Рассмотрены устройство, достоинства и недостатки некоторых известных анонимных сетей, включая сети луковичной маршрутизации и сети с архитектурой P2P. Предложен новый метод построения сети обмена сообщениями, обеспечивающий анонимность пользователей, через одностороннюю связь между участниками с помощью протоколов луковой маршрутизации и UDP. Предложена схема гарантии доставки сообщений, использующая протоколы конфиденциального вычисления и забывчивой передачи. Предложенный метод может обеспечить более надёжное сокрытие факта общения между отправителем и получателем сообщений.

Оглавление

ВВЕДЕНИЕ

В настоящее время обеспечение анонимности клиента в сети Интернет становится всё более обсуждаемой и актуальной темой. Повышенное внимание общества к цензуре и методам борьбы с ней, постоянный рост количества преступлений в информационной сфере, забота государства о частной жизни и персональных данных граждан, — данные тезисы всё чаще появляются в средствах массовой информации и являются предметом постоянных дискуссий в правительстве и обществе.

Журналистам, которым всегда было необходимо сохранять анонимность источников, с развитием технологий стало еще сложнее обеспечивать конфиденциальность своих контактов. Важно отметить, что и коммерческую тайну может составлять не только содержание переговоров, но и сам факт их ведения. В самом деле, фирма-конкурент, не гнушающаяся корпоративного шпионажа, может повлиять на ценообразование поставщиков.

Проблема обеспечения математически доказуемой абсолютной анонимности субъекта при взаимодействии его с адресатом в информационной системе сейчас не решена и относится к ряду концептуальных проблем современных информационных технологий.

Одной из научных задач в данной сфере, решению которой посвящена данная работа, является изучение и разработка новых способов обеспечения анонимности.

Целью данной работы является разработка нового способа обеспечения конфиденциальности с помощью протоколов луковой маршрутизации и транспортного протокола UDP. Использование этих протоколов позволяет устанавливать односторонние соединения, при которых не требуется указания адреса отправителя.

Решение данной задачи открывает перспективу практическим реализациям защищенных программно-аппаратных систем электронного анонимного голосования, средств обеспечения и преодоления цензурирования, системам обмена конфиденциальной информации.

Для демонстрации работы реализована сеть обмена текстовыми сообщениями. Также рассматриваются связанные с использованием такой технологии вопросы гарантии доставки сообщений и противодействия злоумышленникам.

ГЛАВА 1. МЕТОДЫ И МОДЕЛИ ИСПОЛЬЗОВАНИЯ ТРАНСПОРТНОГО ПРОТОКОЛА В АНОНИМНЫХ СЕТЯХ С ЛУКОВОЙ МАРШРУТИЗАЦИЕЙ

1.1. Луковая маршрутизация

Луковая маршрутизация — это такой вид передачи информации, при котором сообщения передаются из источника к месту назначения через последовательность прокси («луковых маршрутизаторов») [1, 2, 3]. Для этого отправитель зашифровывает сообщение с помощью открытых ключей промежуточных участников и получателя, а затем пересылает первому промежуточному узлу. Каждый промежуточный участник этой цепи получает сообщение в зашифрованном виде, расшифровывает его и пересылает полученное сообщение следующему участнику. При этом ни один из таких участников не может гарантировать, что отправивший пакет узел данному участнику является отправителем сообщения, а получивший пакет узел является получателем сообщения.

Таким образом, возможно обеспечение анонимной отправки сообщений и для этого не требуется доверие к каждому промежуточному узлу. За счет того что каждый участник расшифровывает всё сообщение, полученное и отправленное сообщения отличаются, а следовательно стороннему наблюдателю будет трудно проследить путь сообщения, даже если он контролирует часть луковых маршрутизаторов на пути.

Однако, чем больше участников контролирует злоумышленник, тем легче ему установить факт общения между неподконтрольными ему участниками. Захваченный промежуточный маршрутизатор не позволит сторонним лицам прочесть или достоверно подменить сообщение, не зная закрытых ключей остальных участников и получателя.

Промежуточные участники также могут пересылать полученные ими сообщения не напрямую, а через посредников. Что затруднит определение получателя и отправителя, но ухудшит скорость и вероятность доставки сообщения.

1.2. Некоторые представители анонимных сетей

1.2.1. Tor

Tor – это сеть позволяющая устанавливать анонимное сетевое соединение с использованием луковичной маршрутизации [4, 5, 6]. Пользователи сети осуществляют передачу данных через несколько последовательных виртуальных туннелей, что и обеспечивает анонимность соединения.

Tor использует протокол обмена сообщениями основанный на луковой маршрутизации. Пользователи подключаются к серверам Tor через локальный прокси-сервер и образуют цепочку соединения через три различных узла сети Tor, которые выбираются случайным образом. Каждый пакет данных шифруется последовательно тремя открытыми ключами участников цепочки, начиная с конца. Далее происходит пересылка пакета по луковому протоколу.

Важно то, что устанавливается полноценное TCP-соединение между любыми двумя участниками цепочки, что позволяет организовать полноценную работу пользователя сети Tor с любыми другими внешними сетями.

Локальный прокси-сервер Tor основан на интерфейсе SOCKS [7]. SOCKS — сетевой протокол, предназначенный для пересылки данных от одного участника сети к другому таким образом, чтобы можно было использовать сервисы, находящиеся под защитой межсетевых экранов. Это достигается путем передачи всех данных без модификаций, в отличие от обычных прокси-серверов. То есть с точки зрения получателя, данные, полученные от прокси с использованием SOCKS, совпадают с данными, переданными напрямую, без проксирования. SOCKS оперирует на уровне TCP-соединений и может быть применен различными приложениями.

Tor предназначен для скрытия факта связи между участниками и не производит шифрование самих данных. Таким образом, для обеспечения конфиденциальности пользователю необходимо самостоятельно использовать криптографические или стеганографические методы сокрытия информации. Что требует применения специальных средств совместимых с работой сети Tor, так как Tor работает только по протоколу SOCKS.

1.2.2 I2P

I2P — это анонимная самоорганизующаяся распределённая сеть, которая использует модифицированные распределённые хеш-таблицы Kademila и архитектуру сети P2P [8, 9]. В хеш-таблицах хранятся хеши, зашифрованных с помощью AES, IP-адресов узлов сети, а также публичные ключи шифрования. Сеть использует «чесночую маршрутизацию» — модификацию луковой маршрутизации, при которой несколько пакетов могут объединяться в один. Данная модификация затрудняет возможность прослеживания маршрута для стороннего наблюдателя.

Сеть предоставляет приложениям транспортный механизм для анонимной и защищённой пересылки сообщений друг другу. Причем, благодаря библиотеке Streaming lib [10] даётся возможность использовать в сети I2P потоковые протоколы и сервисы. Соединение устанавливается либо с помощью протокола TCP либо с помощь протокола UDP, но работа по UDP организована аналогично TCP: «тройное рукопожатие», подтверждения о доставке пакета и так далее [11]. Данный подход применен из-за проблем вызываемых большим числом открытых TCP-соединений.

В сети различаются конечные пользователи и промежуточные маршрутизаторы. Любой узел сети может быть пользователем, посредником или

обоими сразу. Когда в сети появляется новый пользователь, он обращается к специальным серверам имен для получения базы адресов посредников. С помощью полученного списка строится однонаправленный туннель из посредников, на конце которого находится конечный пользователь. Туннель может быть как входящим, так и исходящим. Разные приложения использующие I2P строят разные туннели, таким образом, один участник может обладать несколькими туннелями.

Данные о построенном туннеле сообщаются серверам имен, и включают в себя адрес конечного маршрутизатора построенного туннеля, идентификатор туннеля, время создания туннеля, необходимые ключи и подпись этих данных. Сервера имен распределяют данную информацию между собой.

Для того чтобы отправить сообщение другому участнику сети необходимо найти нужный входящий туннель в локальной базе адресов или сделать запрос к серверам имен. Далее между двумя туннелями организовывается пересылка данных по принципу чесночной маршрутизации. Для того чтобы получатель смог ответить отправителю необходимо явно указать информацию о входящем туннеле отправителя.

Определение адреса получателя в сети обладает следующими особенностями.

1). С помощью локальной базы адресов определяется хеш-сумма адреса получателя.

2). Локальная база адресов обновляется периодически при помощи специальных серверов имен.

3). Поддомены не привязаны к основному домену.

4). Возможна работа с несколькими серверами имен одновременно, вследствие чего, возможны конфликты имен. Для их устранения применяются специальные механизмы.

5). Хеш адреса используется посредниками в туннеле для определения адреса получателя.

6). Серверы имен не нарушают одноранговость сети, так как являются распределенными.

7). Для передачи данных каждый раз генерируется новый маршрут через случайные узлы.

8). Время передачи через такой маршрут не может превышать 10 минут после его создания.

9). Большинство серверов имен не требуют платы за регистрацию доменов в своей базе.

Поскольку сеть является одноранговой и децентрализованной, скорость и надежность сети напрямую зависят от участия людей в передаче чужого трафика.

Для работы с I2P необходимо установить специальный клиент, а также настроить браузер, если требуется осуществлять работу с сайтами в сети I2P. Для доступа в Интернет необходимо использовать специальные прокси-серверы сети I2P (outproxy). Альтернативным методом работы с сайтами в сети I2P является использование специальных прокси-серверов, которые перенаправляют запросы из внешних сетей в I2P.

Весь трафик шифруется четыре раза.

1). Сквозное шифрование защищает данные от прочтения узлами маршрута.

2). Чесночное шифрование защищает данные от прочтения посредниками при передаче от туннеля к туннелю.

3). Туннельное шифрование защищает данные внутри туннеля.

4). Шифрование транспортного уровня защищает от анализа сторонними наблюдателями.

Для того чтобы ещё больше затруднить анализ содержимого пакетов, данные модифицируются добавлением случайного количества случайных байт перед каждым шифрованием. Сетевые пакеты могут быть разбиты на несколько частей отправителем и отправлены через различные маршруты, а получатель может объединить полученные данные. Эти меры позволяют достаточно надежно защитить пересылаемые данные от попыток анализа или прослушки с помощью снифферов, вне зависимости от того кто контролирует промежуточные маршрутизаторы. Однако, это не гарантирует анонимности при контроле злоумышленником большого числа посредников [12].

В I2P сети используются (для разных уровней и протоколов) следующие системы и методы шифрования и подписи:

1). 256 бит AES режим CBC с PKCS#5;

2). 2048 бит схема Эль-Гамаля;

3). 2048 бит алгоритм Диффи — Хеллмана;

4). 1024 бит DSA;

5). 256 бит HMAC;

6). 256 бит SHA256.

1.2.3. Bitmessage

Bitmessage — криптографическое приложение для обмена сообщениями, основанное на архитектуре сети P2P [13]. Сеть Bitmessage целиком и полностью основана протоколе TCP.

Механизм работы таков:

1). Подключаясь к сети каждый участник получает список некоторых известных узлов и генерирует открытый и закрытый ключ. Версия протокола, а также хеш открытого ключа в кодировке Base58 служат адресом участника.

2). Каждый участник периодически рассылает все полученные им зашифрованные сообщения всем другим доступным участникам, включая свои исходящие сообщения. Это делается, чтобы было трудно четко определить отправителя. Исходящие сообщения шифруются открытым ключом получателя. Запросы на получение открытых ключей также передаются всем участникам.

3). Каждый раз получая новые сообщения, участник пытается расшифровать их своим ключом. Это позволяет избежать публикации адреса получателя. Получатель может также разослать всем соседям подтверждение о доставке сообщения тем же способом, что и обычное сообщение. Таким образом, отправитель может узнать, что сообщение было доставлено, но не может точно определить кто получатель.

4). Все сообщения хранятся не более двух дней, а затем удаляются. Это позволяет избежать чрезмерного расхода памяти.

Важно отметить, что для того чтобы организовать анонимное общение достаточно знать только один открытый и закрытый ключ. Шифруя сообщения открытым ключом, его смогут прочесть все, кому известен закрытый ключ. Из-за механизма распространения сообщений будет достаточно трудно отследить отправителя и получателей.

Для того чтобы избежать неконтролируемого наплыва сообщений используется система Proof-of-Work [14], алгоритм которой подгоняется так, чтобы средний компьютер тратил в среднем четыре минуты времени на отправку стандартного сообщения и при этом количество работы совершаемого по этому алгоритму зависело бы от длины отправляемого сообщения.

1.2.4. Freenet

Freenet — одноранговая сеть, предназначенная для децентрализованного распределённого хранения данных, созданная с целью обеспечения анонимности пользователей [15]. Сеть использует разновидность маршрутизации по ключам, похожую на распределённую хеш-таблицу, для определения местонахождения пользовательских данных. В качестве транспортного протокола применяется TCP.

Информация, сохранённая в сети, шифруется и распределяется по всем компьютерам участников по частям. Таким образом, для участника тяжело определить какой именно файл он хранит. Помимо этого, участники периодически обмениваются сохраненной информацией. Анонимность отправителя в процессе отправки не гарантирована, поэтому рекомендуется использовать либо только доверенные узлы для отправки файлов, либо использовать дополнительные средства анонимизации.

Извлечение информации происходит по ключу, который может быть либо хешем файла(или его части) либо его подписью, что защищает от подмены злоумышленниками. Также помимо ключа извлечения необходимо знать ключ расшифровки файла, что предотвращает прочтение файлов посторонними лицами. При внесении файла можно не шифровать его, тем самым открыв для широкого круга лиц.

Так как в основе сети лежит P2P архитектура, запрос может пройти неопределённую цепочку узлов, а, следовательно, тяжело установить является ли получатель получателем или очередным промежуточным узлом.

Из-за механизма реализации, Freenet затруднено хранение динамически модифицируемых файлов или их удаление. Однако, несмотря на это существует возможность организовать их хранение с помощью ключей специального вида. Данные ключи хранят подпись файла и требуемую ревизию файла. Ревизия файла обновляется каждый раз при модификации и сохраняется в имени файла. При запросе возвращается файл с наибольшей ревизией, но не меньше чем указанная в запросе. Кроме того, файлы не востребованные в течение некоторого времени будут удалены.

Хоть сама сеть и не реализует обмен сообщениями, существуют сторонние приложения основанные на протоколе данной сети. Например, Freenet Messaging System [16] или Frost [17].

1.3. Использование UDP в луковой маршрутизации

UDP — более простой, чем TCP [18], протокол без установления соединения [19]. Связь достигается путём передачи информации в одном направлении от источника к получателю без проверки готовности или состояния получателя. Благодаря этому достигается более высокая скорость отправки сообщений, чем с помощью TCP. Но имеются и минусы: когда сообщение посылается, неизвестно, достигнет ли оно своего назначения — оно может потеряться по пути.

Главной особенностью применения протокола UDP в луковой маршрутизации является возможность дополнительного обеспечения анонимности отправителя пакета. Поскольку, устанавливается одностороннее соединение, отправитель пакета данных может не указываться в заголовке, а, следовательно, получатель не может точно знать отправителя, только канал передачи. Однако, задача гарантии доставки только усложняется.

Стоит отметить, что для того чтобы воспользоваться этим преимуществом, необходимо использовать драйвер сетевой карты, позволяющий подобный режим работы. Реализация такого драйвера в работе не рассматривается, так как не представляет собой научного интереса.

Для того чтобы решить проблему гарантии доставки и при этом сохранить анонимность, могут применяться различные методы – такие как передача об успешной доставке сообщения с помощью децентрализованных сетей (например, как в сети Bitmessage) или с помощью анонимного обмена с некоторым специальным сервером. В первом случае, каждый участник будет сообщать обо всех известных случаях доставки сообщений всем своим соседям. Поскольку будут передаваться все случаи доставки сообщений, анонимность с этой стороны вопроса будет обеспечена. Хотя при большом размере сети, может возникнуть перегрузка каналов сети из-за большого количества необходимых для передачи сообщений. Кроме того, существуют специфические атаки связанные с данной архитектурой сети [20, 21].

Анонимный обмен с сервером организовать сложнее. Пусть последний участник передачи (кроме получателя) после доставки добавит запись в базу данных. Тогда отправитель, чтобы получить сведения о доставки должен сделать запрос в базу, причем такой, чтобы сервер не мог знать, какое сообщение проверяет его клиент. Эту задачу можно решить с помощью протокола конфиденциального вычисления [22].

1.4. Постановка задачи

В силу того, что ни одна сеть из рассмотренных не может быть адаптирована под схему использования односторонних соединений без перепроектирования, было решено разработать собственную реализацию сети обмена сообщениями с использованием протоколов луковой маршрутизации и UDP.

Данная сеть должна обладать следующими свойствами.

1). Обеспечивать сокрытие факта общения между двумя пользователями сети.

2). Сообщать отправителю – доставлено сообщение или нет. При этом анонимность общения не должна нарушаться.

3). Обеспечивать защиту от прочтения посторонними лицами.

Для этого необходимо.

1). Реализовать систему гарантии доставки сообщения.

2). Разработать сеть обмена сообщениями.

В качестве ограничений принято, что сеть не обеспечивает сокрытие использования сети пользователем и что необходимый сетевой драйвер для сокрытия адреса отправителя не будет описан.

1.5. Выводы

Рассмотрены принципы организации анонимных сетей с помощью луковой маршрутизации. Рассмотрены ограничения накладываемые на сети данной архитектуры.

Рассмотрены особенности применения протокола UDP в анонимных сетях с луковой маршрутизацией. На основе этих особенностей была предложена новая схема организации работы анонимных сетей с помощью односторонних соединений по протоколу UDP.

Рассмотрены основы построения и функциональные особенности анонимных сетей Tor, I2P, BitMessage, Freenet.

Tor полностью основан на луковой маршрутизации и направлен на сокрытии адреса отправителя пакетов в сети Интернет. Поддерживается только транспортный протокол TCP, а, следовательно, использование односторонних соединений невозможно.

I2P представляет собой сильно модифицированный протокол луковой маршрутизации. Старые версии сети работали на протоколе TCP, новые версии работают на протоколе UDP, но организация односторонних соединений на протоколе UDP невозможна из-за основных принципов работы заложенных при создании сети.

BitMessage и Freenet являяются децентрализованными сетями. Не используют луковую маршрутизацию для обеспечения анонимности и поэтому использование односторонних соединений не имеет смысла.

На основе изученной информации, поставлена задача о разработке программного обеспечения реализующего сеть обмена сообщениями основанной

на протоколах луковой маршрутизации и UDP, с использованием односторонних соединений.

ГЛАВА 2. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА СЕТИ ОБМЕНА СООБЩЕНИЯМИ И СХЕМЫ ГАРАНТИИ ДОСТАВКИ

2.1. Протокол конфиденциального вычисления

В качестве решения задачи гарантии доставки можно предложить протокол конфиденциального вычисления — криптографический протокол, позволяющий нескольким участникам произвести вычисление, зависящее от тайных входных данных каждого из них, таким образом, чтобы ни один участник не смог получить никакой информации о чужих входных данных [23].

Реализации протокола разделяют на двухсторонние и многосторонние подходы [24]. Двухсторонний подход, как правило, представляет функцию для вычисления в виде логического контура. В многостороннем подходе функция представляется, в так называемом, арифметическом контуре. Отличие арифметического контура от логического заключается в том, что логический контур построен над полем GF(2), а арифметический над GF(p). Многосторонний подход применяются тогда, когда известно, что большинство участников честные (что возможно только тогда, когда число участников больше двух). Примерами первого подхода являются протоколы Яо [25] и GMW (Goldreich, Micali and Wigderson) [26]. Примерами второго подхода является протоколы BGW (Ben-Or, Goldwasser and Wigderson) [27] и CCD (Chaum, Crepeau and Damgard) [28].

В предложенном Яо протоколе программа выполняется следующим образом. Целевая функция программы представляется в виде логического контура из соответствующих вентилей. Далее каждый вентиль контура шифруется – для каждого i-го провода вместо нулей и единиц подставляются случайные наборы по N бит (наборы k_i^0 и k_i^1 соответственно).

Тогда значения в таблице истинности вентиля вычисляются как

$$c_{a,b} = E_{k_1^a}\left(E_{k_2^b}\left(k_3^{f(a,b)}\right)\right) \quad , \qquad (1)$$

где $E_k(x)$ – симметричное шифрование x с ключом k; 1,2 – номера входных проводов, 3 – номер выходного провода, f(i,j) – значение логической операции. Получив такую таблицу из 4 значений, нельзя определить какую логическую операцию реализует данный вентиль. Зная таблицу и входные параметры, пользователь может вычислить значение операции вентиля. Дважды выполнив дешифрование с соответствующими ключами, пользователь найдет единственное корректное значение, не зная, что оно шифрует.

Вместо того, чтобы дважды шифровать значение можно использовать схему, в которой шифрование происходит один раз, но тогда ключ должен считаться как некоторая функция от входных наборов. Причем на эту функцию накладывается требование разнообразия, то есть выходные данные должны отличаться при разных входных данных [29].

В общем виде протокол выглядит следующим образом.

1). Алиса передаёт зашифрованный контур Бобу.

2). Боб получает у Алисы свои входные параметры с помощью протокола забывчивой передачи. При такой передаче Алиса не узнает значения Боба.

3). Теперь Боб последовательно вычисляет значения контура, и затем передаёт зашифрованный результат Алисе.

4). Алиса расшифровывает результат и сообщает его Бобу.

2.2. Протокол забывчивой передачи

Забывчивая передача (oblivious transfer) —тип протокола передачи данных, где пользователь получал в точности 1 часть информации, а сервер не знал, какую именно; кроме того, пользователь не знал ничего об оставшихся частях, которые не были получены [30, 31, 32].

Алгоритм забывчивой передачи, используемый в протоколе конфиденциального вычисления, выглядит следующим образом.

1). Сервер знает пару ключей RSA, содержащие модули N, публичную экспоненту e и скрытую d. Сервер знает выходные маски очередного входного вентиля m_0, m_1.

2). Клиент знает публичную экспоненту и модуль N. А также очередной бит b входного данного.

3). Сервер генерирует случайные числа x_0, x_1 размером 10 байт и пересылает их клиенту.

4). Клиент по известному биту b выбирает x_b.

5). Клиент генерирует случайное значение k, сохраняет его в списке под номером бита и шифрует x_b, рассчитывая

$$v = x_b + k^e \pmod N, \tag{2}$$

которое возвращает серверу.

6). Сервер не знает, какое из x0 и x1 выбрал клиент, и пытается расшифровать оба случайных сообщения, получая два возможных значения k:

$$k_0 = (v - x_0)^d \pmod N, \tag{3}$$

$$k_1 = (v - x_1)^d \pmod N. \tag{4}$$

Одно из них будет соответствовать k, будучи корректно расшифрованным, тогда как другое будет случайным значение, не раскрывающем никакой информации о k.

7). Сервер шифрует оба выхода вентиля с каждым возможным ключом:

$$c_0 = m_0 + k_0, \tag{5}$$

$$c_1 = m_1 + k_1. \tag{6}$$

И посылает полученный вентиль клиенту.

8). Клиент знает какое из двух сообщений может быть расшифровано с помощью k, и он получает возможность расшифровать только одно сообщение

$$m_b = c_b - k. \tag{7}$$

2.3. Сравнение средств извлечения информации из базы данных

2.3.1. SFDL

Коллектив разработчиков во главе с Далией Малкхи реализовал язык высокого уровня SFDL (Secure Function Definition Language), исполняемый приложением Fairplay [33, 34], основанный на протоколе предложенный Яо. Приводится такой пример: Боб имеет базу данных из 16 предметов. Каждый предмет описывается 6-битным ключом и 24-битным кодом. Алиса хочет тайно узнать код одного из предметов, обладая 6-битным ключом. Количество входных данных со стороны Боба 480 бит(16*(6+24)), со стороны Алисы 6 бит. Всего программа имеет 1229 вентилей. Выполнение всей программы при передаче по локальной сети заняло около 0,49 секунды, а при передаче через Интернет около 3,38 секунды.

2.3.2. VIFF

Ещё одной достойной упоминания реализации протокола является библиотека VIFF на языке Python [35]. В ней реализован собственный протокол конфиденциальных вычислений [36]. Этот протокол основан на использовании мультипликативных троек в поле GF(p) для обмена разделенным секретом [37]. Из-за особенности языка Python, в данной программе контур строится не напрямую, а интерактивно. То есть если интерпретатором было встречено выражение a+b, то инициируется соответствующее конфиденциальное вычисление. А после него интерпретатор переходит к следующему выражению.

2.3.3. Сравнение

Для сравнения была написана программа на языке SFDL, принимающая на вход N 64-битных чисел отражающих идентификаторы таблицы базы данных и ещё одно 64-битное число – идентификатор, проверяемый пользователем. На выход подавался один бит (успех или неуспех), причем значение бита известно только пользователю. Также была разработана программа на языке Python, использующая библиотеку VIFF. В этой программе операции проводились над полем GF(36 893 488 147 419 103 363). Тестирование проводилось на виртуальной машине с процессором Intel i3 2120 и ОЗУ 2 гигабайта [38].

Компилятор SFDL смог сгенерировать контур только для N=35 входных значений. Время генерации контура составило 5 минут. Время выполнения около 1 секунды. Программа на языке Python запускалась при N=10 000 и время выполнения оказалось больше 15 минут. Результаты тестирования данных программных продуктов следует признать неудовлетворительным для использования в сетях передачи сообщений.

2.4. Требования к системе.

Для демонстрации возможностей применения технологии, описанной в предыдущей главе, было решено разработать программное обеспечение удовлетворяющее следующим требованиям.

2.4.1. Требования к системе в целом.

Система состоит из двух уровней: уровень клиентов и уровень сервера. Приложение может выполнять как функции клиента, так и сервера. Но с точки зрения проектирования эти режимы программы следует отличать друг от друга.

Клиенты общаются между собой по механизму луковой маршрутизации, используя только протокол UDP. Для того чтобы сохранить анонимность общения и исключить обратную связь между клиентами требуется специальный сервер для хранения информации о доставке сообщения. Сообщение считается доставленным, если его хеш, посчитанный по алгоритму MD5 [39], хранится на сервере.

После получения сообщения, получатель должен добавить хэш сообщения на сервер. Для того чтобы не допустить возможности установления факта общения между получателем и отправителем, следует использовать протокол конфиденциального вычисления Яо при извлечении отправителем информации о доставке с сервера. В качестве вспомогательных алгоритмов используется шифрование RSA [40] и протокол забывчивой передачи.

Кроме того, сервер должен хранить список текущих пользователей, для того чтобы использовать их в качестве промежуточных узлов луковичной маршрутизации. Каждый участник сети может являться промежуточным узлом. Получатель и отправитель сообщения должны храниться в списке текущих пользователей сервера.

Пользователи должны иметь возможность выбрать сервер по своему усмотрению. Любой пользователь может активировать режим сервера в приложении.

Для обеспечения мультиплатформенности системы, она должна быть написана на языке Java, в среде выполнения JavaSE-1.7 [41].

2.4.2. Требования к серверу.

Взаимодействие с сервером происходит только по протоколу TCP. Сервер должен давать следующие возможности:

1). выдачу списка текущих участников для клиента;

2). добавление и удаление текущих участников;

3). добавление хеша сообщения от клиента;

4). взаимодействие по протоколу конфиденциального вычисления Яо для извлечения хеша сообщения.

На сервере должно храниться не более 10 000 последних полученных хешей по дате доставке на сервер и не более 10 000 текущих пользователей.

Список участников хранится в виде списка их интернет-адресов и открытых ключей этих участников размером 2048 бит сгенерированных по алгоритму шифрования RSA, и не содержит какой-либо другой информации.

Список хешей сообщений хранится в виде массива байт.

Сервер запускается в консоли и поддерживает только одну команду: «exit» — остановку сервера. Информация о хешах или текущих участников обнуляется при каждом новом запуске сервера.

2.4.3. Требования к клиенту.

Взаимодействие с сервером происходит только по протоколу TCP. Взаимодествие с другим клиентом только по UDP. Клиент должен иметь следующие возможности:

1). выбор сервера для хранения информации о доставке сообщения;

2). получения списка текущих участников с сервера;

3). добавления и удаления пользователя как текущего участника сервера;

4). добавления хеша сообщения на сервер;

5). взаимодействие по протоколу конфиденциального вычисления Яо для извлечения хеша сообщения;

6). отправку, пересылку и получение сообщений другим пользователям;

Список участников хранится в виде списка их адресов и открытых ключей этих участников размером 2048 бит сгенерированных по алгоритму шифрования RSA, и не содержит какой-либо другой информации.

При отправке сообщения отправитель указывает адрес получателя и само сообщение. Более никаких данных не требуется. Каждое сообщение шифруется с помощью гибридного шифрования алгоритмов AES [42] и RSA.

Клиент запускается в консоли и поддерживает следующие команды. В качества параметров при запуске указывается номер порта локального адреса и адрес сервера.

«connect X» — добавление к серверу с адресом X. Допускается подключение только к одному серверу в один момент времени.

«send X Y» — отправка текстового сообщения Y получателю с адресом X. Максимальный размер сообщения — 20 000 символов.

«exit» — команда завершения работы клиента.

2.5. Описание классов приложения.

2.5.1. Классы запуска.

2.5.1.1. Main.

Данный класс запускает выполнение программы. Управление программой происходит в потоке этого класса.

public static void main(String[] args) — метод, являющийся точкой входа в программу. Схема алгоритма на рис. 1.

public static void serverMode() — метод, реализующий запуск сервера. Запускает потоки классов ServerAck и ServerUsers. Схема алгоритма на рис. 2.

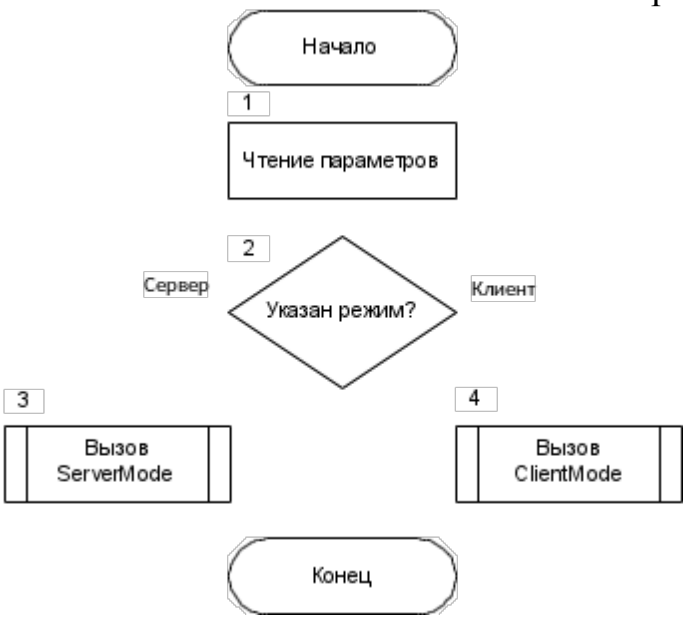

Рис. 1. Схема алгоритма main.

Рис. 2. Схема работы serverMode.

public static void clientMode(int port) — метод, реализующий запуск клиента. Запускает поток класса Client. Схема алгоритма на рис. 3.

Рис. 3. Схема работы clientMode.

2.5.1.2. TestGate.

Класс, предназначенный для теста взаимодействия сервера и клиента по протоколу Яо.

public static void main(String[] args) — метод, являющийся точкой входа в программу. Параметром является строка аргументов, переданная при запуске программы.

public static void serverMode() — метод, реализующий запуск сервера.

public static void clientMode() — метод, реализующий запуск клиента.

2.5.1.3. TestOT.

Класс, предназначенный для тестирования протокола забывчивой передачи. Тестирование может производиться без передачи информации по сети.

public static void main(String[] args) — метод, являющийся точкой входа в программу. Параметром является строка аргументов, переданная при запуске программы.

2.5.2. Классы сервера.

2.5.2.1. ServerAck.

Данный класс обрабатывает запросы на добавление хеша, хранит актуальные хеши и взаимодействует с клиентами по протоколу Яо. Класс выполняется в отдельном потоке, так как в нём происходит ожидание сообщений передаваемых по сети.

Общий цикл работы.

Схема общего цикла работы приведена на рис. 4. К участнику может приходить три типа запросов: добавление хеша; проверки хеша; закрытия программы. Последний запрос необходим, так как класс выполняется в потоке отличном от главного потока программы. При этом помимо запроса, необходимо установить соответствующие флаги внутри класса, что исключает возможность использование этого запроса с удаленного компьютера.

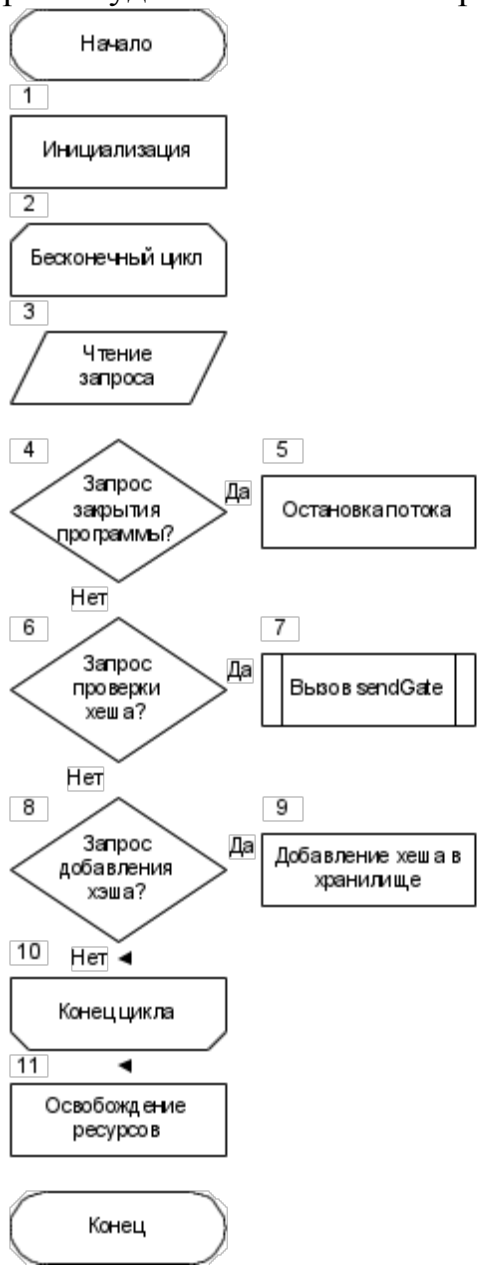

Рис. 4. Схема работы run.

При инициализации сервер создает пару ключей RSA.

При добавлении участник отправляет код запроса размером 1 байт и хеш сообщения размером 16 байт. Сервер определяет код запроса и заносит хеш в список хешей. В ответ отправляется код подтверждения операции.

При запросе хеша пользователь присылает требуемый код запроса и далее происходит взаимодействие между пользователем и сервером по протоколу конфиденциальных вычислений Яо. Подробно взаимодействие описано в следующем пункте.

При закрытии программы происходит освобождение использованных ресурсов.

Проверка хеша.

Схема алгоритма на рис. 5.

Рис. 5. Схема работы sendGate

1). Сервер формирует список хешей, каждый из которых размером 16 байт. Всего не более 100 000 хешей. Сервер строит логический контур из этого списка с входными данными размером 16 байт с помощью класса GateConstructor.

2). Сервер генерирует пару ключей RSA, если это не было сделано. Сервер пересылает клиенту открытый ключ.

3). Вызывается метод obliviousTransfer класса OTServer.

4). Все входные биты контура устанавливаются по значениям, полученным с помощью протокола забывчивой передачи, использованным на предыдущем шаге.

5). Весь логический контур записывается в бинарный файл.

6). Сервер пересылает выходные эталоны клиенту.

7). Сервер пересылает получившийся файл с контуром клиенту.

2.5.2.2. OTServer.

Вспомогательный класс класса ServerAck, реализующий работу с клиентом с помощью протокола забывчивой передачи.

public void obliviousTransfer(InputStream in, OutputStream out) — метод реализует взаимодействие с клиентом по протоколу забывчивой передачи в целом. В качестве параметров передаётся входной и выходной потоки сокета сервера. Схема алгоритма на рис. 6.

Для каждого входного бита контура выполняется следующий алгоритм.

Сервер знает пару ключей RSA, содержащие модули N, публичную экспоненту e и скрытую d. Сервер знает выходные маски очередного входного вентиля m0, m1.

Клиент знает публичную экспоненту и модуль N. А также b – значение очередного бита входного данного.

1). Сервер генерирует случайные числа x0, x1 размером 10 байт и пересылает их клиенту.

2). Клиент по известному биту b выбирает x0 или x1. Клиент генерирует случайное значение k и шифрует xb, рассчитывая некоторое v с помощью метода encryptOT класса CryptoService. Значение v возвращается серверу.

3). Сервер не знает, какое из x0 и x1 выбрал клиент, и пытается расшифровать оба случайных сообщения с помощью метода decryptOT класса CryptoService, получая два возможных значения k:

Одно из них будет соответствовать k клиента, будучи корректно расшифрованным, тогда как другое будет случайным значение, не раскрывающем никакой информации о k.

4). Сервер шифрует оба выхода вентиля с каждым возможным ключом

$$c0 = m0 + k0, \quad (8)$$

$$c1 = m1 + k1, \quad (9)$$

где c0 и c1 — выходные значения таблицы истинности класса GateInput.

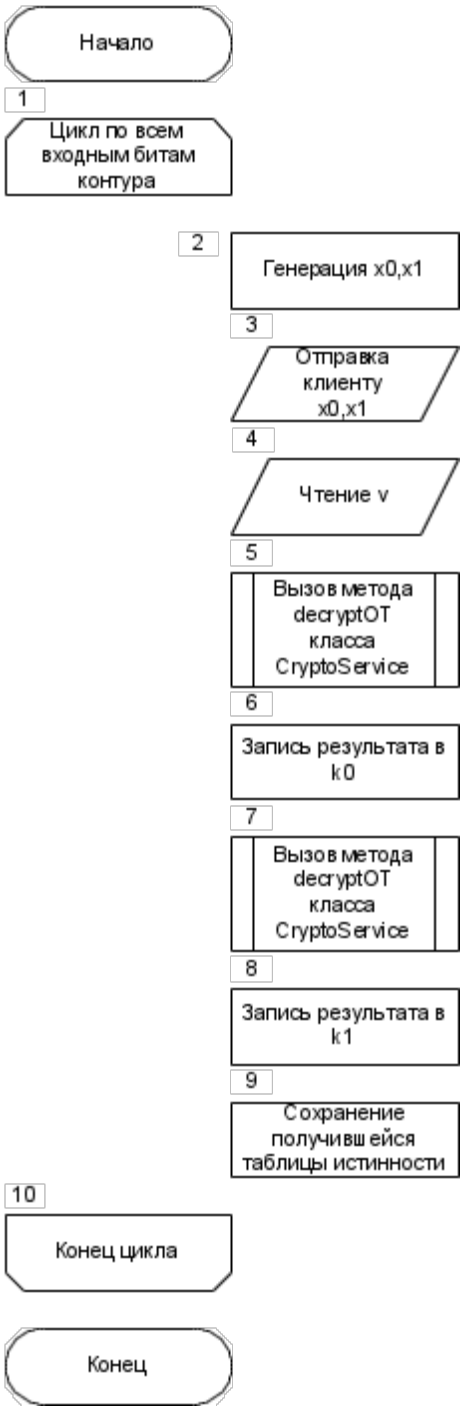

Рис. 6. схема работы obliviousTransfer

2.5.2.3. ServerUsers.

Класс предназначен для хранения актуальных данных пользователей и обработки запросов, связанных с этими данными. Класс выполняется в отдельном потоке, так как в нём происходит ожидание сообщений передаваемых по сети.

Схема общего цикла работы приведена на рис. 7. К участнику может приходить четыре типа запросов: добавление участника; удаления участника; выдачи списка участников; закрытия программы. Последний запрос необходим, так как класс выполняется в потоке отличном от главного потока программы. При этом помимо запроса, необходимо установить соответствующие флаги внутри

класса, что исключает возможность использование этого запроса с удаленного компьютера.

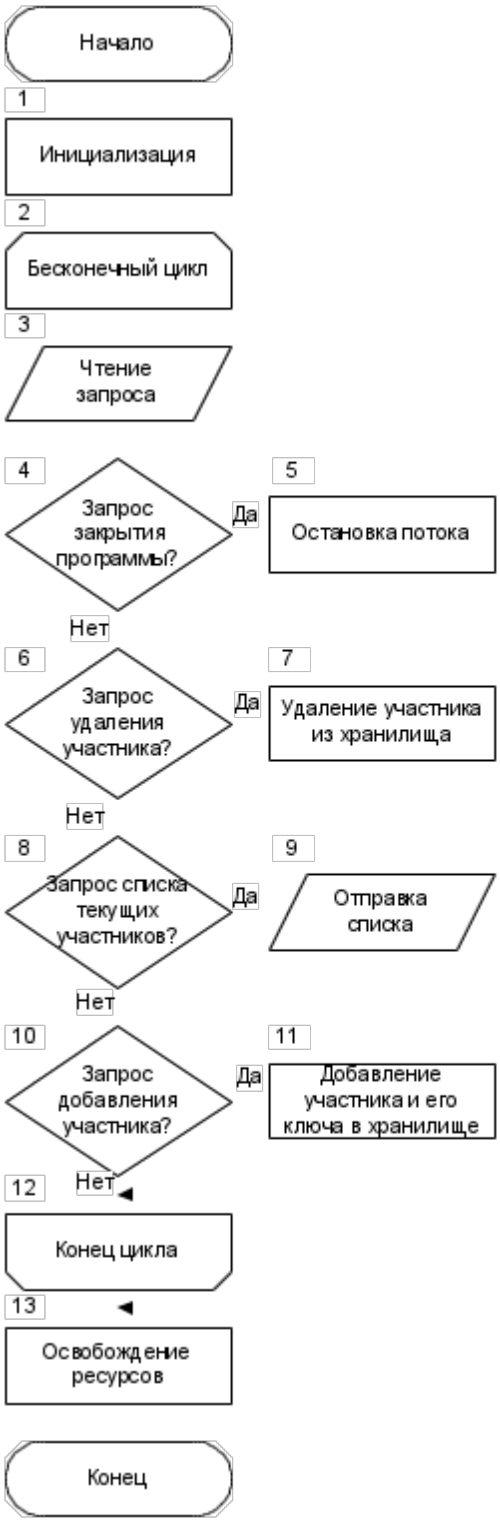

Рис. 7. схема работы run

При добавлении участник отправляет нужный код запроса и открытый ключ. Сервер определяет код запроса и заносит интернет-адрес участника и его ключ в список текущих участников. Ключ берётся из тела запроса, а адрес из заголовков TCP пакета. В ответ генерируется случайным образом и отправляется код участника на сервере.

При удалении участник отправляет требуемый код запроса и код участника. Информация о данном пользователе исключается из списка текущих участников. Использование кода участника позволяет избежать злоупотреблений со стороны других пользователей.

Для получения списка участников отправляется соответствующий код. В ответ приходит число участников, далее сообщаются адрес и открытый ключ каждого пользователя, хранящиеся в текущем списке участников класса.

При закрытии программы происходит освобождение использованных ресурсов.

2.5.3. Классы клиента.

2.5.3.1. Client.

Данный класс реализует обмен сообщениями между клиентами. Класс выполняется в отдельном потоке, так как в нём происходит ожидание сообщений передаваемых по сети.

Все сообщения между участниками шифруются с помощью схемы гибридного шифрования [43] алгоритмами RSA с длиной ключа 2048 бит и AES в режиме CBC с длиной ключа 128 бит. Открытый ключ RSA отправляется на сервер при регистрации. Закрытый ключ RSA известен только пользователю. Ключ AES выступает в роли сессионного ключа и генерируется автоматически.

Каждое зашифрованное сообщение состоит из трёх частей. Первые четыре байта содержат в себе информацию, сколько байт составляет вторая часть. Следующая часть содержит в себе ключ алгоритма AES, зашифрованного с помощью RSA. Третья часть содержит в себе сообщение, зашифрованное с помощью ключа AES. Данное разбиение на части позволяет избежать деления сообщения по блокам, необходимое при шифровании с помощью только RSA, и при этом предоставляет достаточно гибкости, если потребуется сменить алгоритмы шифрования.

Общий цикл работы.

Схема алгоритма приведена на рис. 8. К участнику может приходить четыре типа запросов: пересылки сообщения; получения сообщения; отправки сообщения; закрытия программы. Последние два запроса необходимы, так как класс выполняется в потоке отличном от главного потока программы. При этом помимо запроса, необходимо установить соответствующие флаги внутри класса, что исключает возможность использование этих типов запросов с удаленного компьютера.

Первым шагом выполняется инициализация всех переменных класса, в том числе внутри класса создаются экземпляры классов ClientUsers и ClientAck. С помощью класса ClientUsers делаются запросы регистрации и получения участников к серверу, адрес которого является параметром при создании класса Client.

Далее запускается бесконечный цикл ожидания сообщений. Цикл останавливается, только после обработки запроса закрытия программы. Каждый

пришедший запрос сначала расшифровывается, а затем проверяется первый байт запроса, являющийся кодом запроса.

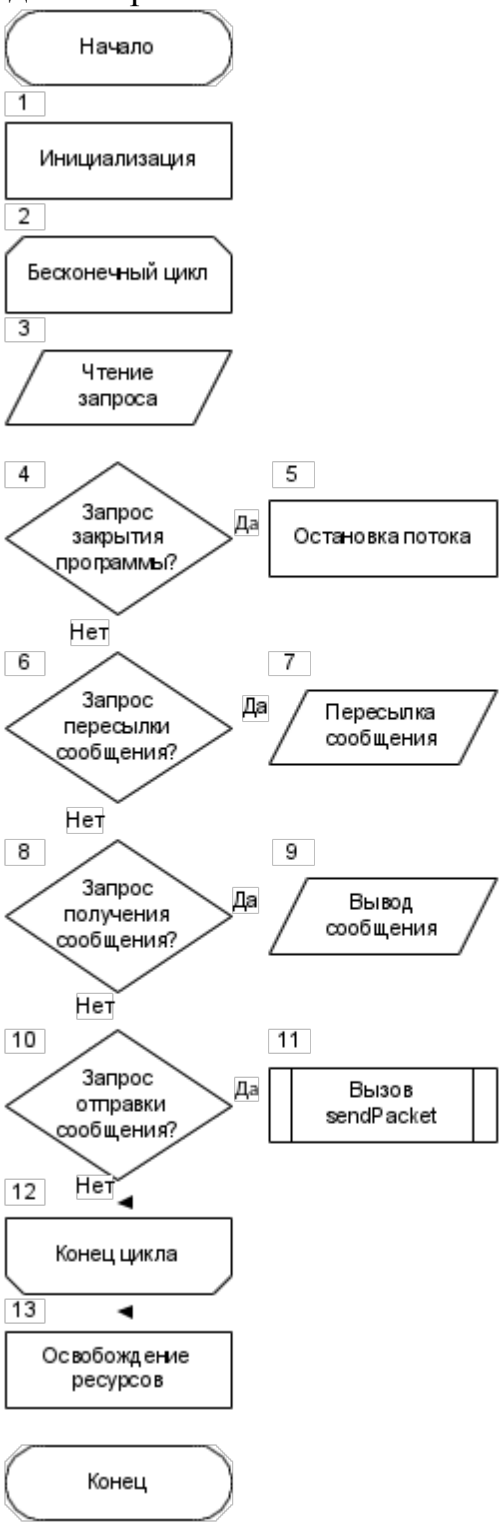

Рис. 8. схема работы run.

Обработка запроса отправки сообщения описывается в следующем пункте.

При запросе пересылки считывается адрес, после кода запроса, а всё оставшиеся пересылается по этому адресу без каких-либо правок.

При запросе получения сообщения выводится текст на экран содержащее адрес отправителя и само сообщение. При этом вызывается метод getAcknowl-

edgement класса ClientAck, который обеспечивает доставку хеша сообщения на сервер.

При запросе закрытия, сначала с помощью класса ClientUsers на сервер делается запрос удаления информации об участнике, а после совершается выход из цикла и освобождение ресурсов.

Отправка сообщения.

Отправка сообщения осуществляется методом sendPacket. Схема алгоритма данного метода приведена на рисунке 9. До вызова метода отправитель выбирает получателя и пишет ему сообщение. Считается, что отправителю уже известен актуальный список участников на сервере.

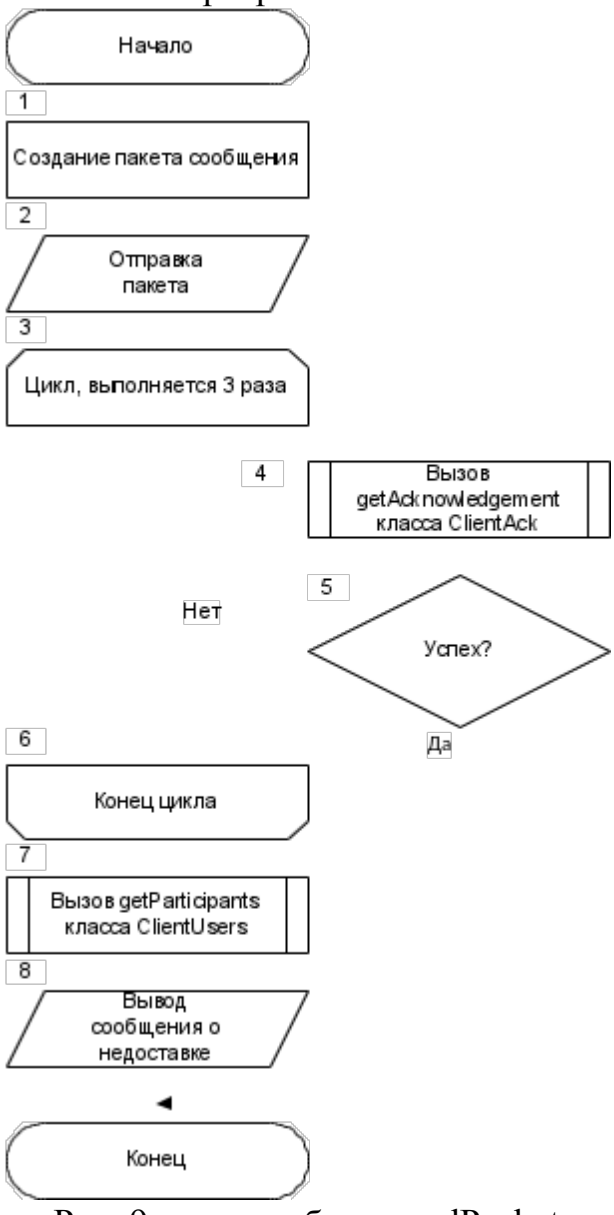

Рис. 9. схема работы sendPacket

1). Первым шагом идет создание пакета сообщения. Первым байтом пакета записывается код операции получения сообщения. Далее записывается адрес отправителя и само сообщение. Сообщение зашифровывается. Случайно выбирается число от 1 до 5 (но не более количества текущих участников на сервере) промежуточных маршрутизаторов.

Формируется новый пакет для очередного промежуточного маршрутизатора выбранного случайным образом. Пакет состоит из кода операции пересылки, адреса следующего участника маршрута и уже созданного пакета сообщения. Пакет зашифровывается. Это повторяется до тех пор, пока число созданных таким образом пакетов не станет равным числу маршрутизаторов, полученным на предыдущем шаге.

2). Полученный пакет пересылается промежуточному маршрутизатору выбранному последним.

3). Отправитель делает три попытки получить подтверждение о получении сообщения получателем на сервере с помощью метода getAcknowledgement класса ClientAck с интервалом в 5 секунд. В качестве исходного данного используется хэш исходного сообщения.

4). Если подтверждение не получено, обновляется список участников на сервере, посылка считается неудачной и выводится сообщение об ошибке.

2.5.3.2. AckClient.

Вспомогательный класс для взаимодействия с сервером по протоколу Яо и отправке подтверждений о доставленных сообщениях.

При отправке подтверждения о доставке участник отправляет код соответствующего запроса размером 1 байт и хеш сообщения размером 16 байт. В ответ принимается код подтверждения операции.

При запросе хеша пользователь присылает требуемый код запроса и далее происходит взаимодействие между пользователем и сервером по протоколу конфиденциальных вычислений Яо. Схема алгоритма на рис. 10.

Рис. 10. схема работы getAcknowledgement

1). Отправка запроса серверу.

2). Сервер пересылает клиенту открытый ключ.

3). Вызывается метод obliviousTransfer класса OTClient.

4). Все значения для расшифровки входных битов контура сохраняются в памяти.

5). Сервер пересылает выходные эталоны клиенту.

6). Сервер пересылает получившийся файл с контуром клиенту.

7). Клиент восстанавливает значения хеша записанные во входные вентили с помощью шага 3 протокола забывчивой передачи для каждого бита входного данного.

8). Клиент вычисляет контур, подставляя вместо вентилей с номерами 0-127 вычисленные на предыдущем шаге значения.

9). Полученное значение возвращается из метода.

2.5.3.3. OTClient.

OTClient — вспомогательный класс для взаимодействия с сервером по протоколу забывчивой передачи.

public void obliviousTransfer(InputStream in, OutputStream out, byte[] check) — метод реализует взаимодействие с клиентом по протоколу забывчивой передачи в целом. В качестве параметров передаётся входной и выходной потоки сокета сервера, проверяемое значение. Схема алгоритма на рис. 11.

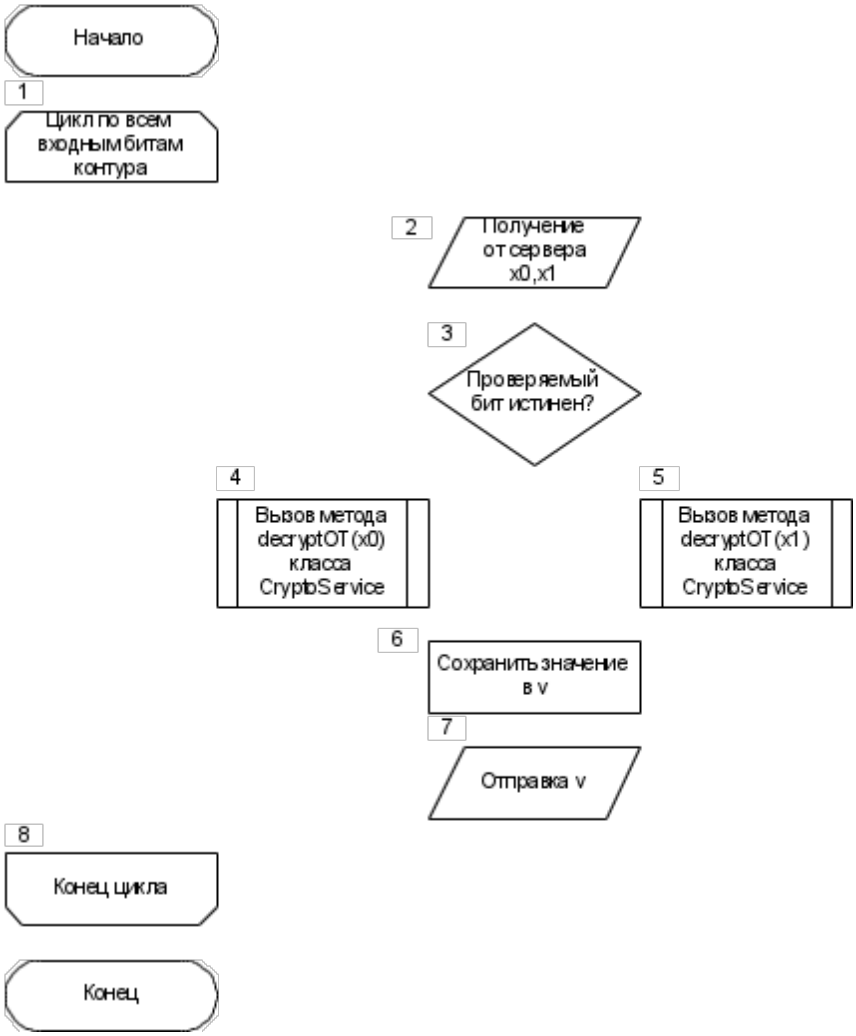

Рис. 11. схема работы obliviousTransfer

Для каждого входного бита контура выполняется следующий алгоритм.

Сервер знает пару ключей RSA, содержащие модули N, публичную экспоненту e и скрытую d. Сервер знает выходные маски очередного входного вентиля m0, m1.

Клиент знает публичную экспоненту и модуль N. А также b – значение очередного бита входного данного.

1). Сервер генерирует случайные числа x0, x1 размером 10 байт и пересылает их клиенту.

2). Клиент по известному биту b выбирает x0 или x1. Клиент генерирует случайное значение k, сохраняет его, и шифрует xb, рассчитывая некоторое v с помощью метода encryptOT класса CryptoService. Значение v возвращается серверу.

2.5.3.4. UsersClient.

UsersClient — вспомогательный класс для отправки и обработки запросов, связанных с информацией о клиентах, серверу.

Для регистрации на сервере участник отправляет код запроса регистрации и открытый ключ RSA на сервер. Если ключ ещё не сгенерирован, то он генерируется. Адрес участника не указывается. Сервер отправляет код участника.

Для удаления информации об участнике отправляется код запроса удаления и код участника.

Для получения списка участников отправляется соответствующий код. В ответ приходит число участников, далее сообщаются адрес и открытый ключ каждого участника.

2.5.4. Вспомогательные классы.

2.5.4.1. CryptoService.

CryptoService — класс реализующий взаимодействие с алгоритмами шифрования, расшифрования, хеширования, в том числе необходимых для протокола забывчивой передачи.

public static byte[] encryptOT(byte[] x, RSAPublicKey key, byte[] save) — метод шифрует значение x по формуле:

$$v = x + k^e \pmod N, \quad (10)$$

где k — случайно сгенерированное значение, а e и N — параметры открытого ключа RSA key. Значение k сохраняется в массив байт save. Схема алгоритма на рис. 12.

public static byte[] decryptOT(byte[] v, byte[] x, RSAPrivateKey key) — метод вычисляет k с помощью зашифрованного v и оригинального x по формуле:

$$k = (v - x)^d \pmod N, \quad (11)$$

где d,N — параметры закрытого ключа RSA key. Схема алгоритма на рис. 13.

public static byte[] finalizeOT(byte[] m, byte[] k, RSAPublicKey key) — метод вычисляет res с помощью зашифрованного m и известного ключа k по формуле:

$$res = m - k \pmod N, \quad (12)$$

где N — параметр открытого ключа RSA key.

Рис. 12. encryptOT

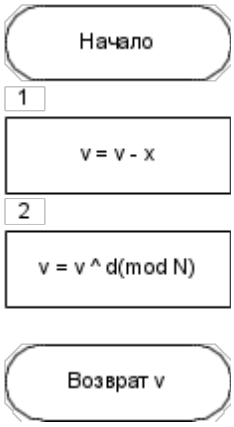

Рис. 13. decryptOT

2.5.4.2. StreamMediator.

Класс реализующий безопасное запись и чтение информации в потоках предоставляемые сетевыми сокетами.

2.5.4.3. UserInfo.

Данный класс описывает необходимую информацию о пользователе: адрес, открытый ключ.

2.5.4.4. Address.

Класс описывающий ip-адрес и порт. Применяется в классе UserInfo.

2.5.4.5. Queue.

Класс реализующий хранение хешей на сервере.

2.5.4.6. ByteArrayWrapper.

Вспомогательный класс класса Queue, предназначенный для обертывания массива байт представляющих собой хеш.

2.5.4.7. Constants.

Класс хранящий все «магические» константы используемые в программе.

2.5.5. Классы, описывающие логические вентили.

2.5.5.1. GateConstructor.

Класс реализующий генерацию логического контура по списку хешей. Данный контур проверяет равенство какого-либо хеша из списка и входного параметра. В контуре таблицы истинности каждого вентиля зависят от двух битовых параметров.

Каждому логическому вентилю контура ставится в соответствии уникальный идентификатор — неотрицательное целое число размером 4 байта. Входные значения логического контура описываются вентилями с номерами 0-127. Таблицы истинности вентилей входных значений при входе равном 00 или 01 выдают 0, при входе равном 11 или 10 выдают 1.

Каждый логический вентиль преобразуется в форму требуемую протоколом Яо. Каждое элементарное значение вентиля маскируется маской 80 бит.

2.5.5.2. Gate.

Gate — класс реализующий базовый интерфейс логического вентиля.

При выводе формирует бинарный файл. Первые 4 байта файла содержат в себе целое число, описывающее количество вентилей. Далее записаны все вентили контура. Каждый вентиль описывается 52 байтами. Первые 4 байта идентификатор вентиля, следующие 4 байта идентификатор первого входного вентиля, следующие 4 байта идентификатор второго входного вентиля. Последующие 40 байт описывают маски таблицы истинности вентиля. Если вентиль не имеет входного вентиля, то в идентификаторе входного вентиля записывается отрицательное число

2.5.5.3. GateInput.

Класс-наследник Gate, реализующий входной вентиль.

2.5.5.4. GateAnd.

Класс-наследник Gate, реализующий операцию И.

2.5.5.5. GateOr.

Класс-наследник Gate, реализующий операцию ИЛИ.

2.5.5.6. GateNop.

Класс-наследник Gate, реализующий пустую операцию передачи сигнала.

2.5.5.7. GateNot.

Класс-наследник Gate, реализующий операцию логического отрицания.

2.6. Выводы

В качестве средства гарантии доставки сообщений выбран и обоснован протокол конфиденциальных вычислений. Изучен и обоснован алгоритм работы протокола забывчивой передачи, необходимый для того чтобы гарантировать анонимность участников при взаимодействии по протоколу конфиденциальных вычислений.

Проведен анализ готовых решений способных удовлетворить данному протоколу. Приложение на языке SFDL оказалось не способно решить поставленную задачу, а библиотека VIFF оказалась слишком медленной. На основе анализа решено разработать данную задачу самостоятельно.

Были разработаны требования к программному обеспечению, реализующему анонимную сеть обмена сообщениями с помощью протоколов луковой маршрутизации и UDP. На основе этих требований были спроектированы и реализованы все классы программы.

На примере классов Client, ClientUsers, ClientAck, OTClient, ServerUsers, ServerAck, OTServer были рассмотрены все взаимодействия между участниками сети. Описаны все вспомогательные классы, реализованные для работы программы.

В описании класса GateConstructor описана структура логического контура. В описании класса Gate приведена структура файла описывающего логический контур.

ГЛАВА 3. АНАЛИЗ ЭФФЕКТИВНОСТИ ПРЕДЛОЖЕННОГО МЕТОДА

3.1. Сравнение с другими анонимными сетями

На таблице 1 приведено сравнение по основным параметрам анонимных сетей, описанных в подразделе 1.2., и приложения «TorUDP», реализация которого приведена в предыдущем разделе.

Таблица 1. Основные характеристики сетей.

	Tor	I2P	BitMessage	Freenet	TorUDP
Транспортный протокол	TCP	TCP/UDP	TCP	TCP	UDP
Луковая маршрутизация	Да	Да	Нет	Нет	Да
Передача сообщений	Да*	Да*	Да	Да*	Да
Передача файлов	Да*	Да*	Нет	Да	Нет
Одностороннее соединение	Нет	Да**	Нет	Нет	Да

* — с помощью сторонних приложений;

** — в I2P участвуют «односторонние» туннели, но по ним может передаваться в обе стороны техническая информация [11].

На таблице 2 приведено сравнение анонимных сетей по времени передачи файла размером 2 мегабита. В сетях BitMessage и TorUDP, вместо файлов передаётся текст аналогичного объёма. В сети Tor для передачи файла использовался OnionShare [44], а в сети I2P qtI2P-Messenger 0.2.24 [45].

Передача проводилась между двумя виртуальными машинами с процессорами Intel i3 2120 и ОЗУ 2 гигабайта. Между каждой попыткой проходило не менее 2 часов, для того чтобы избежать проблем связанных в нагрузкой сетей.

Таблица 2. Время передачи файла размером 2 Мбит по сети.

	Tor	I2P	BitMessage	Freenet	TorUDP
Измерение 1	1 мин 2 с	1 мин	10 мин 13 с	14 мин 2 с	6 мин 50 с
Измерение 2	56 с	57 с	11 мин 20 с	11 мин 15 с	6 мин 57 с
Измерение 3	57 с	неудача	10 мин 40 с	12 мин 42 с	6 мин 48 с
Измерение 4	58 с	3 мин 17 с	10 мин 52 с	13 мин 19 с	6 мин 51 с
Измерение 5	57 с	неудача	10 мин 33 с	10 мин 7 с	6 мин 54 с

Во время передачи файла по сети I2P приложение qtI2P-Messenger дважды выдавало ошибку, что привело к двум неудачным попыткам. Данная ошибка не связана с работой сети I2P.

3.2. Известные угрозы

У луковой маршрутизации есть несколько слабостей. В общем случае безопасность анонимной сети прямо пропорциональна количеству узлов – участников сети. Улучшение равномерности статистического распределения узлов также является действенной мерой против многих типов атак. Уязвимости, связанные с использованием протокола TCP, отброшены.

3.2.1. Атака по времени

Если злоумышленник наблюдает за относительно слабо загруженным луковым маршрутизатором, он может соединять входящие/исходящие сообщения путём просмотра того, как близко по времени они были получены и переправлены [46, 47]. Однако это может быть преодолено путём буферизации нескольких сообщений и передачи их с использованием псевдослучайного временного алгоритма.

Атакующий может, например, заставить сервер отправлять браузеру данные с определёнными задержками. Это позволит обнаружить в зашифрованном трафике анонимной сети «шаблоны» задержек и, таким образом, с определённой вероятностью ответить на вопрос о принадлежности выходного трафика анонимной сети «подозреваемому» пользователю. Методы защиты от атаки включают внесение переменных задержек в характер информационного обмена, перемешивание и объединение сообщений, пересылку их блоками фиксированного размера.

3.2.2. Контроль большинства

Контролируя большинство маршрутизаторов на сервере, злоумышленник получает возможность установки факта общения между случайными пользователями [48]. Чем больше узлов контролируется, тем выше вероятность достоверно установить и получателя и отправителя. Однако, чем больше узлов в маршруте, тем меньше вероятность установить это достоверно. Например, контролируя 80% сети, злоумышленник может установить и отправителя и получателя, при условии что используется только три промежуточных маршрутизатора, с вероятностью 0,512. Ещё одним препятствием служит то, что длина цепи определяется случайным образом.

3.2.3. Захваченный сервер пользователей

Сервер, контролируемый злоумышленником, может подделывать списки пользователей. Подделывая списки, сервер может добиться того, что все выдаваемые промежуточные маршрутизаторы будут контролироваться одной группой. Однако, чтобы отправитель мог отправить сообщение, получатель тоже должен находиться в списке промежуточных маршрутизаторов. Таким образом, сервер должен знать заранее какому получателю отправитель хочет отправить

сообщение. Иначе отправки сообщения не произойдет. Эта атака является частным способом предыдущей.

3.2.4. Захваченный сервер подтверждений

Сервер, контролируемый злоумышленником, может подделывать хеши сообщений. Сервер достоверно знает, кто получил сообщение с нужным хешем, так как устанавливается прямое TCP соединение между получателем и сервером. Выдавая хеши по одному, до тех пор пока клиент не перестанет запрашивать подтверждение доставки, сервер может определить кто является отправителем. Это пресекается выполнением запроса подтверждения всегда одинаковым числом попыток.

3.2.5. Взлом шифрования

Для того чтобы прочитать сообщение необходимо либо взломать ключ шифра AES длиной 256 бит, либо взломать ключ шифра RSA длиной 2048 бит, так как используется схема гибридного шифрования. Для AES на данный момент эффективные методы взлома неизвестны, а, следовательно, требуется перебрать 2^{256} возможных комбинаций ключей. Для взлома RSA достаточно факторизовать модуль открытого ключа N. Наиболее подходящим для этого алгоритмом является общий метод решета числового поля [49]. Воспользовавшись оценкой Карла Померанса [50] можно вычислить необходимое число операций по формуле:

$$O\left(\exp\left(\sqrt[3]{\frac{64}{9}+O(1)}(\log N)^{\frac{1}{3}}(\log\log N)^{\frac{2}{3}}\right)\right), \quad (13)$$

где N — модуль RSA. При $N = 2^{2048}$ требуется выполнить 2^{122} операций.

3.2.6. Взлом протокола конфиденциальных вычислений.

Корректность и надёжность использования протокола конфиденциальных вычислений Яо обоснована в работе Линделла [29]. Однако, одна из сторон может обманывать другую. Стоит отметить, что сервер не узнает никакой дополнительной информации пытаясь обманывать клиента из-за мер принятых в пункте 3.2.4. Клиент же может только узнать присутствует ли некоторый хэш в хранилище сервера.

3.2.7. Взлом протокола забывчивой передачи.

На основании информации доступной по протоколу, сервер не сможет выяснить какое сообщение из двух необходимо клиенту, а также узнать сгенерированное им секретное число k [30]. Клиент же не может прочесть оба сообщения, не зная секретную экспоненту закрытого ключа RSA. Таким образом, для взлома данного протокола необходимо взломать криптосистему RSA (см. пункт 3.2.5).

3.3. Выводы

Проведен анализ эффективности предлагаемой сети обмена сообщениями по сравнению с другими анонимными сетями. По результатам анализа, выяснено, что разработанное приложение выигрывает по скорости доставки сообщения у сетей не использующих луковую маршрутизацию, но проигрывает использующим таковую. Однако, только сеть TorUDP реализует одностороннее соединение сообщений.

Выявлены и проанализированы основные угрозы нарушения анонимности участников. Включая: наблюдение за участниками и временными задержками; контроль участников и серверов; взлом алгоритмов и протоколов, используемых в программе. Определены и применены методы противодействия к данным атакам. Оценены ресурсы необходимые для взлома алгоритмов и протоколов.

Таким образом, разработка приложения успешно завершена.

ЗАКЛЮЧЕНИЕ

В результате проведенного исследования были выполнены работы, позволяющие получить решение поставленных задач. Рассмотрены устройство, достоинства и недостатки некоторых известных анонимных сетей, включая сети луковичной маршрутизации и сети с архитектурой P2P. Предложен новый метод построения сети обмена сообщениями, обеспечивающий анонимность пользователей, через одностороннюю связь между участниками с помощью протоколов луковой маршрутизации и UDP.

Данный метод позволяет не только скрыть адрес отправителя, но и достоверно подделать его, включая параметр времени жизни пакета. Использование предложенного метода открывает перспективу практическим реализациям защищенных программно-аппаратных систем электронного анонимного голосования, средств обеспечения и преодоления цензурирования, системам обмена конфиденциальной информации. Но для этого требуется доработка существующего или разработка нового сетевого драйвера.

Предложена схема гарантии доставки сообщений, использующая протоколы конфиденциального вычисления и забывчивой передачи. Рассмотрены алгоритмы данных протоколов. Обоснована их корректность и надежность.

Для демонстрации работы предложенного метода были написаны функциональные требования к программному обеспечению, реализующему данный метод. На основе данных требований было разработано приложение TorUDP. В работе описаны классы приложения, основные процедуры приложения, общие принципы работы.

Полученное приложение прошло сравнительный анализ с ранее описанными сетями. И хотя предложенный метод проигрывает по времени работы более популярным луковым сетям, таким как Tor и I2P, метод может обеспечить более надёжное сокрытие факта общения между отправителем и получателем сообщений. Выявлена модель угроз для участников. Разработаны и внедрены меры противодействия этим угрозам.

По итогам работы стоит отметить, что несмотря на полную работоспособность приложения, для полноценного раскрытия потенциала односторонних соединений необходимо разработать драйвер с возможностью свободного изменения адреса отправителя.

Другим направлением дальнейшей работы может быть модификация сети обмена сообщениями в сеть общего назначения. Это может быть достигнуто, например, с помощью туннелирования протокола TCP протоколом UDP или написанием программного интерфейса для взаимодействия сторонних приложений с разработанной сетью.

.

ЛИТЕРАТУРА

[1] Reed, M. Anonymous Connections and Onion Routing. / Michael G. Reed, Paul F. Syverson, David M. Goldschlag. – USA: Naval Research Laboratory, 1998. – 15 с.

[2] Syverson, P. Onion Routing access configurations. / P. Syverson, M. Reed, and D. Goldschlag. // DARPA Information Survivability Conference and Exposition (DISCEX 2000). – IEEE CS Press, 2000.

[3] Syverson, P. Towards an Analysis of Onion Routing Security. / P. Syverson, G. Tsudik, M. Reed, and C. Landwehr. // Designing Privacy Enhancing Technologies: Workshop on Design Issue in Anonymity and Unobservability. – 2000.

[4] Tor официальный сайт.– URL: https://www.torproject.org/ (дата обращения 30.05.2017).

[5] Dingledine, R. Tor: The second-generation onion router / R. Dingledine, N. Mathewson, P. Syverson // 13th USENIX Security Symposium. San Diego, California, USA (9-13 August 2004). – 2004.

[6] Dingledine, R. Deploying Low-Latency Anonymity: Design Challenges and Social Factors / Roger Dingledine, Nick Mathewson, Paul Syverson // IEEE Security & Privacy. – 2007

[7] RFC 1928. SOCKS Protocol Version 5. – URL: https://tools.ietf.org/html/rfc1928 (дата обращения 30.05.2017).

[8] I2P официальный сайт. – URL: https://geti2p.net/ (дата обращения 30.05.2017).

[9] Muller, J. Analysis of the I2P Network - Information Gathering and Attack Evaluations : дис. … bachelor. : защищена 16.06.16 / Jens Muller. – Bern University of Applied Sciences, 2016.

[10] I2P, документация по библиотеке Streaming lib. – URL: https://geti2p.net/ru/docs/api/streaming (дата обращения 30.05.2017).

[11] I2P, документация по протоколу Secure Semireliable UDP. – URL: https://geti2p.net/ru/docs/transport/ssu (дата обращения 30.05.2017).

[12] Egger, C. Practical Attacks Against the I2P Network / Christoph Egger, Johannes Schlumberger, Christopher Kruegel, Giovanni Vigna. // 16th International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2013). – 2013.

[13] BitMessage официальный сайт. – URL: https://bitmessage.org/ (дата обращения 30.05.2017).

[14] Jakobsson, M. Proofs of Work and Bread Pudding Protocols. / M. Jakobsson and A. Juels: // IFIP TC6 and TC11 Joint Working Conference on Communications and Multimedia Security (CMS ’99). – Kluwer, 1999.

[15] Freenet официальный сайт. – URL: https://www.freenetproject.org/ (дата обращения 30.05.2017).

[16] Freenet Messaging System официальный сайт. – URL: https://d6.gnutella2.info/freenet/USK@0npnMrqZNKRCRoGojZV93UNHCMN-

6UU3rRSAmP6jNLE,~BG-edFtdCC1cSH4O3BWdeIYa8Sw5DfyrSV-TKdO5ec,AQA-CAAE/fms/104/, (дата обращения 30.05.2017).

[17] Frost официальный сайт. – URL: http://jtcfrost.sourceforge.net/ (дата обращения 30.05.2017).

[18] RFC 793. Transmission control protocol. DARPA internet program. Protocol specification. – URL: https://tools.ietf.org/html/rfc793 (дата обращения 30.05.2017).

[19] RFC 768. User Datagram Protocol. – URL: https://tools.ietf.org/html/rfc768 (дата обращения 30.05.2017).

[20] Freedman, M. Tarzan: A peer-to-peer anonymizing network layer. / M. J. Freedman, R. Morris. // 9th ACM Conference on Computer and Communications Security (CCS 2002). – 2002.

[21] Rennhard, M. Practical anonymity for the masses with morphmix. / M. Rennhard, B. Plattner. // Springer-Verlag, LNCS. – 2004.

[22] Di-Crescenzo, G. Universal service-providers for database private information retrieval / Giovanni Di-Crescenzo, Yuval Ishai, Rafail Ostrovsky // 17th Annual ACM Symposium on Principles of Distributed Computing. Puerto Vallarta, Mexico (28 June – 2 July 1998). – 1998.

[23] Hazay, C. Efficient Secure Two-Party Protocols: Techniques and Constructions. / Carmit Hazay, Yehuda Lindell. – Springer, 2010. – ISBN 978-3642143021.

[24] Goldreich, O. General Cryptographic Protocols: The Very Basics / Oded Goldreich. – Cambridge University Press, 2013. – ISBN 978-1-61499-169-4.

[25] Yao, A. Protocols for Secure Computations / Andrew Chi-Chih Yao // 23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA (3-5 November 1982). – 1982. – с. 160 – 164.

[26] Goldreich, O. How to play any mental game or a completeness theorem for protocols with honest majority / Oded Goldreich, Silvio Micali, Avi Wigderson // 19th Annual ACM Conference on Theory of Computing. New York, USA (25-27 May 1987). – 1987.

[27] Ben-Or, M. Completeness theorems for non-cryptographic fault-tolerant distributed computation / Michael Ben-Or, Shafi Goldwasser, Avi Wigderson // 20th Annual ACM Conference on Theory of Computing. Chicago, USA (2-4 May 1988). – 1988.

[28] Chaum, D. Multiparty unconditionally secure protocols / David Chaum, Claude Crépeau, Ivan Damgard // 20th Annual ACM Conference on Theory of Computing. Chicago, USA (2-4 May 1988). – 1988.

[29] Lindell, Y. A proof of security of Yao's protocol for two-party computation. / Y. Lindell, B. Pinkas. // Journal of Cryptology, 22(2). – 2009.

[30] Rabin, M. How to exchange secrets by oblivious transfer. / Michael O. Rabin. // Technical Report TR-81, Aiken Computation Laboratory, Harvard University. – 1981.

[31] Ishai, Y. Extending oblivious transfers efficiently. / Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. // Crypto '03. LNCS 2729. – Springer-Verlag, 2003.

[32] Naor, M. Oblivious transfer and polynomial evaluation. / M. Naor, B. Pinkas. // 31st Symposium on Theory of Computer Science (STOC). – 1999.

[33] Fairplay, официальный сайт. – URL: http://www.cs.huji.ac.il/project/Fairplay/ (дата обращения 30.05.2017).

[34] Malkhi, D. Fairplay — a secure two-party computation system / Dahlia Malkhi, Noam Nisan, Benny Pinkas, Yaron Sella // 13th USENIX Security Symposium. San Diego, California, USA (9-13 August 2004). – 2004.

[35] VIFF официальный сайт. – URL: https://viff.dk/, свободный – Дата обращения 30.05.2017.

[36] Geisler, M. Cryptographic protocols: Theory and implementation : дис. … Ph.D. : защищена 26.02.10 / Martin Geisler. – Aarhus University, 2010.

[37] Cramer, R. Share Conversion, Pseudorandom Secret-Sharing and Applications to Secure Computation / Ronald Cramer, Ivan Damgård, Yuval Ishai // Second Theory of Cryptography Conference. Cambridge, Massachusetts, USA (10-12 February 2005). – 2005.

[38] Бобыкин, Д.В. Применение протокола конфиденциального вычисления для извлечения информации из базы данных [Текст] / Бобыкин Д.В., Елсаков С.М. // ЮЖНО-УРАЛЬСКАЯ МОЛОДЕЖНАЯ ШКОЛА ПО МАТЕМАТИЧЕСКОМУ МОДЕЛИРОВАНИЮ сборник трудов III всероссийской научно-практической конференции. – 2016. – с. 17-20.

[39] RFC 1321. The MD5 Message-Digest Algorithm. – URL: http://www.ietf.org/rfc/rfc1321 (дата обращения 30.05.2017).

[40] RFC 3447. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography. Specifications Version 2.1. – URL: http://www.ietf.org/rfc/rfc3447 (дата обращения 30.05.2017).

[41] Документация Java Platform Standard Edition 7. – URL: http://docs.oracle.-com/javase/7/docs/ (дата обращения 30.05.2017).

[42] FIPS 197. Advanced Encryption Standard (AES). – URL: http://nvlpubs.nist.-gov/nistpubs/FIPS/NIST.FIPS.197.pdf (дата обращения 30.05.2017).

[43] Goldreich, O. Foundations of Cryptography: Vol. 2 Basic Applications. – Cambridge University Press, 2004.

[44] OnionShare официальный сайт. – URL: https://onionshare.org/ (дата обращения 30.05.2017).

[45] qtI2P-Messenger официальный сайт. – URL: http://echelon.i2p.rocks/qti2pmessenger/I2P-Messenger_0.2.24_Beta_src.zip, (дата обращения 30.05.2017).

[46] Ferguson, N. Practical Cryptography. / Niels Ferguson, Bruce Schneier. – John Wiley & Sons, 2003. – ISBN 0-471-22357-3.

[47] Raymond, J. Traffic Analysis: Protocols, Attacks, Design Issues, and Open Problems. // Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability. –2000.

[48] Feigenbaum, J. Probabilistic Analysis of Onion Routing in a Black-box Model. / Joan Feigenbaum, Aaron Johnson, Paul Syverson // WPES'07: Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society. — ACM Press, 2007.

[49] Lenstra, A. The Development of the Number Field Sieve / A. K. Lenstra, H. W. Lenstra. – Springer, 1993. – ISBN 9783540570134

[50] Pomerance, C. A Tale of Two Sieves. / Carl Pomerance // Notices of the American Mathematical Society. 43 (12). – 1996.

ПРИЛОЖЕНИЕ 1. ОПИСАНИЕ ПРОГРАММЫ

П.1.1. ОБЩИЕ СВЕДЕНИЯ О ПРОГРАММЕ

Название программного продукта – «TorUDP».

Программа написана на языке программирования Java. Для использования программы необходима операционная система в которой установлена Java Virtual Machine версии не ниже 7.

П.1.2. ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

Программа предназначена для обмена сообщениями между пользователями.

Программа обладает следующими возможностями.

1. Обмен сообщениями между участниками.
2. Запуск сервера, хранящего информацию о подключаемых пользователях и гарантирующего доставку сообщений.

П.1.3. ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ

Взаимодействие между классами подсистемы показано на рис. П1.1.

Запуск программы и выбор режима работы происходит в классе Main. Классы Client, ServerAck и ServerUsers выполняются в отдельных потоках.

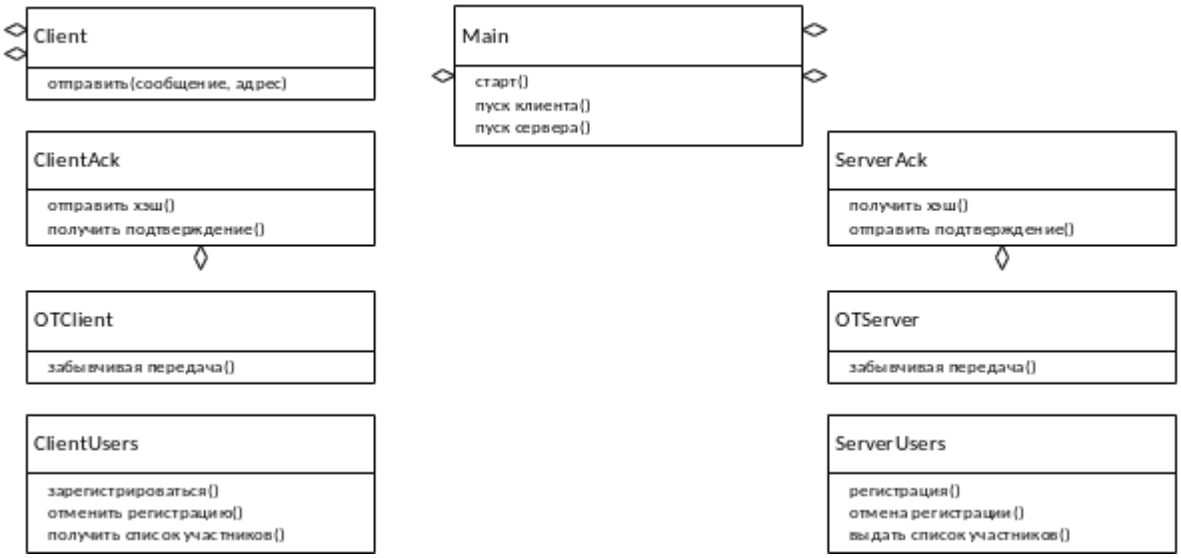

Рис. П.1.1. Взаимодействие классов подсистемы

П.1.4. ИСПОЛЬЗУЕМЫЕ ТЕХНИЧЕСКИЕ СРЕДСТВА

Для работы программы рекомендуется персональный компьютер со следующей конфигурацией:

– процессор с тактовой частотой 1,6 ГГц;

– объем оперативной памяти 1024 Мб;

– дисковое пространство 18 Мб.

На компьютере должна быть установлена операционная система и Java Virtual Machine.

П.1.5. ВЫЗОВ И ЗАГРУЗКА

В консоли перейти в нужную директорию и запустить основной класс программы. Например, командой «java Main».

П.1.6. ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ

Входными данными являются действия пользователя (ввод команд).

Выходными данными являются сообщения присланные пользователю.

ПРИЛОЖЕНИЕ 2. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

П.2.1. ОБЩИЕ СВЕДЕНИЯ О ПРОГРАММЕ

Название программного продукта – «TorUDP».

Программа написана на языке программирования Java. Для использования программы необходима операционная система в которой установлена Java Virtual Machine версии не ниже 7.

Для запуска и работы программы требуются следующие аппаратные и программные компоненты:

1. IBM PC-совместимый компьютер;
2. процессор с тактовой частотой 1,6 ГГц;
3. объем оперативной памяти 1024 Мб;
4. дисковое пространство 18 Мб.

П.2.2. СТРУКТУРА ПРОГРАММЫ

Взаимодействие между классами подсистемы показано на рис. П1.1.

Запуск программы и выбор режима работы происходит в классе Main. Классы Client, ServerAck и ServerUsers выполняются в отдельных потоках.

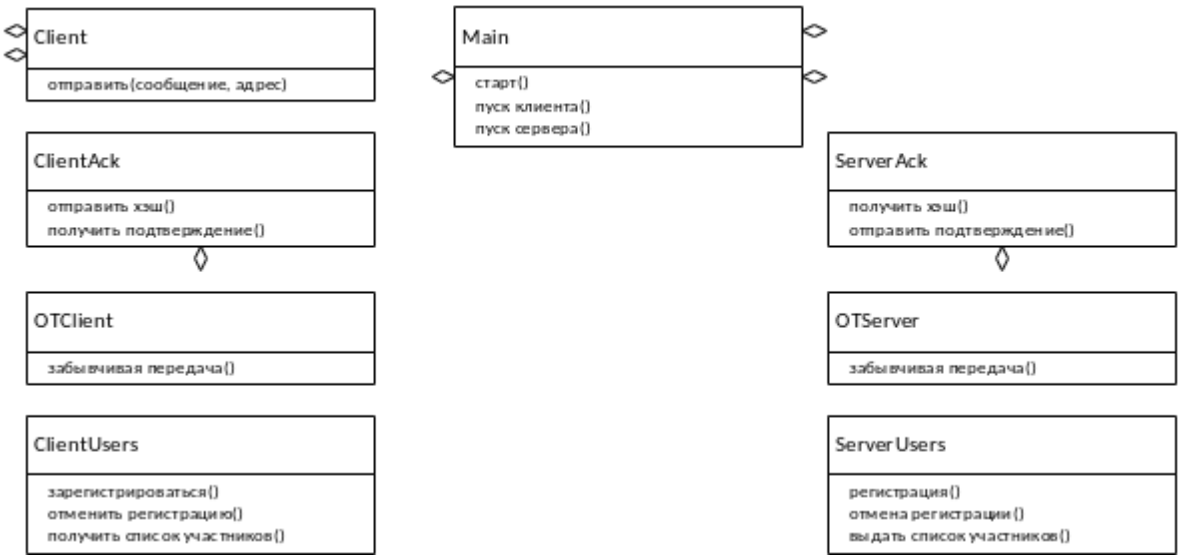

Рис. П.2.1. Взаимодействие классов подсистемы

П.2.3. НАСТРОЙКА ПРОГРАММЫ

Необходимо проверить, что имеющееся программное и аппаратное обеспечение соответствует требованиям, описанным в разделе П2.1.

Дополнительных действий по настройке программы не требуется.

П.2.4. ВЫПОЛНЕНИЕ ПРОГРАММЫ

Для запуска программы требуется запустить файл «Main.class» на исполнение в среде Java Virtual Machine.

П.2.5. ПРОВЕРКА ПРОГРАММЫ

Программа проверена на сети из трёх компьютеров. На рис. П2.2 отображено состояние окна консоли при отправке сообщения. На рис. П2.3 отображено состояние окна консоли при получении сообщения. На рис. П2.4 отображено состояние окна консоли при получении подтверждения о доставке.

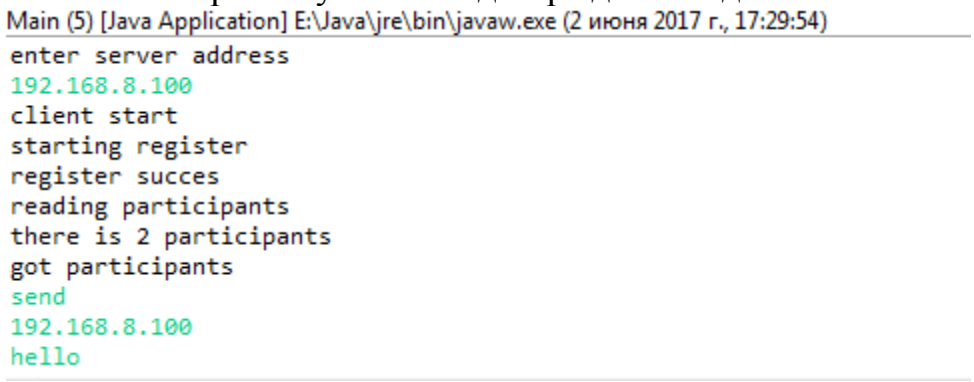

Рис. П.2.2. Отправка сообщения.

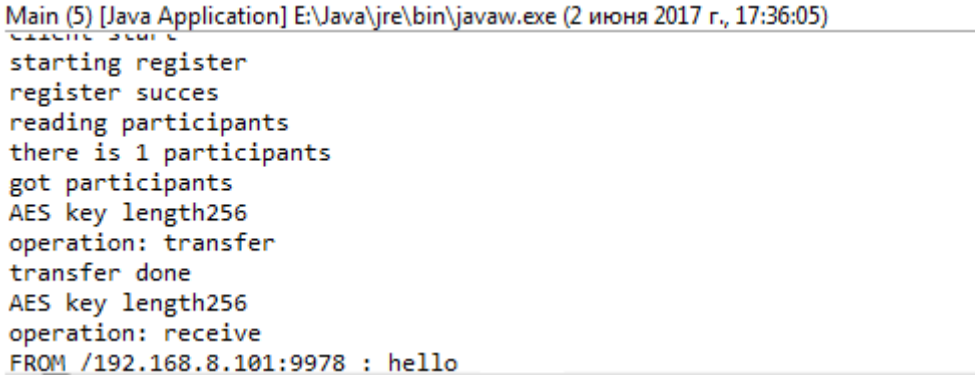

Рис. П.2.3. Получение сообщения.

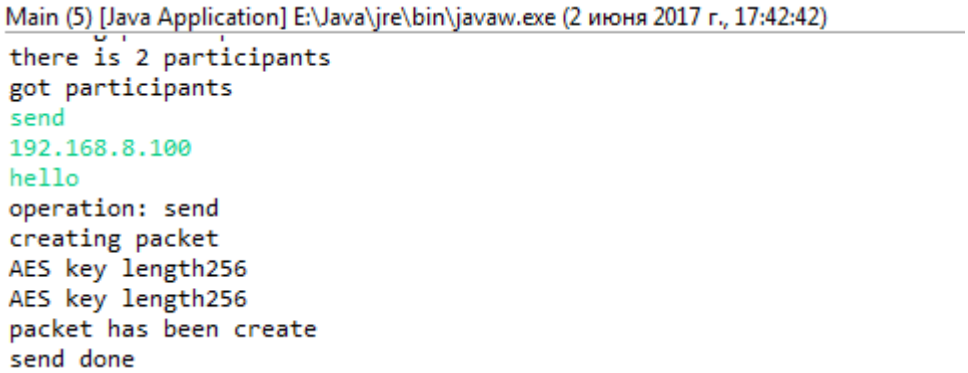

Рис. П.2.4. Получение подтверждения о принятии сообщения.

П.2.6. ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ

Входными данными являются действия пользователя (ввод команд).
Выходными данными являются сообщения присланные пользователю.

ПРИЛОЖЕНИЕ 3. ТЕКСТ ПРОГРАММЫ

П.3.1. ФАЙЛ «Main.java»

```
import java.io.IOException;
import java.net.InetAddress;
import java.net.SocketException;
import java.net.UnknownHostException;
import java.util.Scanner;

import Client.Client;
import Server.ServerAck;
import Server.ServerUsers;

public class Main {
	static ServerUsers su;
	static ServerAck sa;
	static Client c;

	public static void main(String[] args) {

		int port = Config.Constants.portOfClient;

		System.out.println("Select mode");
		Scanner sc = new Scanner(System.in);
		while(sc.hasNext())
		{
			String line = sc.nextLine();
			if(line.equals("client") )
			{
				clientMode(port);
				break;
			}
			if(line.equals("server") )
			{
				serverMode();
				break;
			}
		}
		sc.close();
	}

	public static void serverMode()
	{
		try
		{
			sa = new ServerAck();
			su = new ServerUsers();
		}
		catch (IOException e1)
		{
			e1.printStackTrace();
			System.out.println("critical failure: problem with socket");
			return ;
		}
```

```
		sa.start();
		su.start();

		Scanner sc = new Scanner(System.in);
		while(sc.hasNext())
		{
			String line = sc.nextLine();
			if(line.equals("exit") )
			{
				try
				{
					sa.close();
					su.close();
				}
				catch (IOException e)
				{
					e.printStackTrace();
					System.out.println("critical failure: problem with closing");
					sc.close();
					return ;
				}
				break;
			}
			else
				System.out.println("uncorrect command");
		}
		sc.close();
}

public static void clientMode(int port)
{
		System.out.println("enter server address");
		Scanner sc = new Scanner(System.in);
		String servaddr = sc.nextLine();
		InetAddress serv;
		try
		{
			serv = InetAddress.getByName(servaddr);
		}
		catch (UnknownHostException e1)
		{
			e1.printStackTrace();
			sc.close();
			return ;
		}

		Client c;
		try
		{
			c = new Client(serv, port);
		}
		catch (UnknownHostException | SocketException e)
		{
			e.printStackTrace();
			System.out.println("critical error: server not found");
			sc.close();
			return ;
		}
		c.start();
```

```
                while(sc.hasNext() && c.isAlive())
                {
                        String line = sc.nextLine();
                        if(line.equals("exit") )
                        {
                                try
                                {
                                        c.close();
                                }
                                catch (IOException e)
                                {
                                        e.printStackTrace();
                                        sc.close();
                                        System.out.println("critical failure: problem with closing");
                                        return ;
                                }
                                break;
                        }
                        else if(line.equals("send"))
                        {
                                String ip = sc.nextLine();
                                String mes = sc.nextLine();
                                try
                                {
                                        c.sendMessage(ip, mes);
                                }
                                catch (IOException e)
                                {
                                        e.printStackTrace();
                                        System.out.println("problem with sending");
                                }
                        }
                        else
                                System.out.println("uncorrect command");
                }
                sc.close();
        }

}
```

П.3.2. ФАЙЛ «TestGate.java»

```
import java.io.IOException;
import java.math.BigInteger;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.security.NoSuchAlgorithmException;
import java.security.spec.InvalidKeySpecException;
import java.util.Random;
import java.util.Scanner;

import Client.AckClient;
import Gate.Error;
import Server.ServerAck;

public class TestGate
{
```

```
public static void main(String[] args)
{
	System.out.println("Select mode");
	Scanner sc = new Scanner(System.in);
	while(sc.hasNext())
	{
		String line = sc.nextLine();
		if(line.equals("client") )
		{
			clientMode();
			break;
		}
		if(line.equals("server") )
		{
			serverMode();
			break;
		}
	}
	sc.close();
}

public static void clientMode()
{
	InetAddress serverAddress;
	Scanner sc = new Scanner(System.in);
	try
	{
		String line = sc.nextLine();
		serverAddress = InetAddress.getByName(line);
		sc.close();
	}
	catch (UnknownHostException e)
	{
		sc.close();
		e.printStackTrace();
		return ;
	}
	AckClient ack = new AckClient(serverAddress);

	BigInteger buff;
	byte[] hash;

	System.out.println("preset hash");
	for(int i=0; i<Config.Constants.numberOfValues; ++i)
	{
		buff = new BigInteger(Config.Constants.numberOfBits - 1, new Random());
		hash = buff.toByteArray();
		try
		{
			ack.setAcknowledgement(hash);
		}
		catch (IOException e)
		{
			e.printStackTrace();
			return ;
		}
	}
	System.out.println("preset ok");
```

```
		try {
			hash = Util.CryptoService.hash("123");
		} catch (NoSuchAlgorithmException e1) {
			// TODO Auto-generated catch block
			e1.printStackTrace();
			return ;
		}

		try
		{
			ack.setAcknowledgement(hash);
		}
		catch (IOException e)
		{
			e.printStackTrace();
			return ;
		}

		boolean flag = false;
		try
		{
			flag = ack.getAcknowledgement(hash);
		}
		catch (ClassNotFoundException | NoSuchAlgorithmException | InvalidKeySpecException |
IOException | Error e)
		{
			e.printStackTrace();
			return ;
		}

		System.out.println("client procedure ok");
		if(flag)
			System.out.println("client all ok");
	}

	public static void serverMode()
	{
		ServerAck ack;
		try {
			ack = new ServerAck();
		} catch (IOException e) {
			e.printStackTrace();
			return ;
		}

		ack.start();

		Scanner sc = new Scanner(System.in);
		while(sc.hasNext())
		{
			String line = sc.nextLine();
			if(line.equals("exit") )
			{
				try
				{
					ack.close();
				}
				catch (IOException e)
				{
```

```
                                        e.printStackTrace();
                                        System.out.println("critical failure: problem with closing");
                                        sc.close();
                                        return ;
                                }
                                break;
                        }
                        else
                                System.out.println("uncorrect command");
                }
                sc.close();

                System.out.println("server procedure ok");
        }
}
```

П.3.3. ФАЙЛ «TestOT.java»

```
import java.io.IOException;
import java.math.BigInteger;
import java.security.NoSuchAlgorithmException;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import java.security.spec.InvalidKeySpecException;

import Util.CryptoService;

public class TestOT {

        public static void main(String[] args)
        {
                System.out.println("OT test");
                RSAPrivateKey pkey;
                try {
                        pkey = (RSAPrivateKey) Util.CryptoService.getPrivateKeyFromFile("priv.key");
                } catch (ClassNotFoundException | NoSuchAlgorithmException | InvalidKeySpecException |
IOException e1) {
                        // TODO Auto-generated catch block
                        e1.printStackTrace();
                        return ;
                }
                RSAPublicKey key;
                try {
                        key = (RSAPublicKey) Util.CryptoService.getPublicKeyFromFile("pub.key");
                } catch (ClassNotFoundException | NoSuchAlgorithmException | InvalidKeySpecException |
IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                        return ;
                }

                if( !key.getModulus().equals(pkey.getModulus()) )
                        System.out.println("Wrong modulus");
                BigInteger test = new BigInteger("100");
                BigInteger e = key.getPublicExponent();
                BigInteger d = pkey.getPrivateExponent();
                BigInteger modulus = key.getModulus();
                test = test.modPow(e, modulus).modPow(d, modulus);
                byte[] biginttest = test.toByteArray();
```

```
		if(biginttest.length != 1 && biginttest[0] != 100)
			System.out.println("Wrong exponent");

		byte[] mess = {100, 10};
		byte[] x = new byte[2];
		x[0] = 10;
		x[1] = 25;
		byte[] tmp = new byte[Config.Constants.keyLength + 1];
		byte[] v = CryptoService.encryptOT(x, key, tmp);
		byte[] olol = CryptoService.decryptOT(v, x, pkey);


		BigInteger var = new BigInteger(mess);
		BigInteger var2 = new BigInteger(olol);
		mess = var.add(var2).toByteArray();


		byte[] res = CryptoService.finalizeOT(mess, tmp, key);
		if(res.length == 2 && res[0] == 100 && res[1] == 10)
			System.out.println("OT ok");
		else
			System.out.println("Problem with OT");
	}

}
```

П.3.4. ФАЙЛ «Client/Client.java»

```
package Client;
import java.io.*;
import java.net.*;
import java.security.NoSuchAlgorithmException;
import java.security.spec.InvalidKeySpecException;
import java.util.Random;

import Gate.Error;
import Util.Address;
import Util.StreamMediator;
import Util.UserInfo;

public class Client extends Thread {
	int mode;
	Address clientAddress;
	InetAddress serverAddress;
	InetAddress ipAddress;
	String message;
	DatagramSocket clientSocket;
	UsersClient us;
	AckClient ack;


	public Client(InetAddress server, int p) throws UnknownHostException, SocketException
	{
		mode=0;
		message = "";
		clientSocket = new DatagramSocket(p);
		ipAddress = null;
		clientAddress = new Address(InetAddress.getLocalHost().getAddress(), p);
		serverAddress = server;
		us = new UsersClient(serverAddress, clientAddress);
		ack = new AckClient(serverAddress);
	}
```

```
	private void receivePacket(byte[] input, boolean hasAddress) throws IOException, NoSuchAlgorithmEx-
ception
	{
		String address = "anonymous";
		if(hasAddress)
		{
			byte[] addr = new byte[4];
			System.arraycopy(input, 0, addr, 0, 4);

			InetAddress a = null;
			a = InetAddress.getByAddress(addr);
			address = a.toString();

			System.arraycopy(input, 4, addr, 0, 4);
			int pp = StreamMediator.intFromBytes(addr);
			address += ":";
			address += Integer.toString(pp);

			byte[] tmp = new byte[input.length-8];
			System.arraycopy(input, 8, tmp, 0, input.length-8);
			input = tmp;
		}

		String message = new String(input);
		message.trim();
		System.out.println("FROM " + address + " : " + message);
		try(FileWriter writer = new FileWriter("message.txt", false))
	{
	   writer.write("FROM " + address + " : " + message + "\r\n");
	   writer.flush();
	}
	catch(IOException ex)
			{
	   System.out.println(ex.getMessage());
	}
		AckClient ack = new AckClient(serverAddress);
		ack.setAcknowledgement(Util.CryptoService.hash(message));
	   }

	   private void transferPacket(byte[] mess) throws UnknownHostException
	   {
		byte[] addr = new byte[4];
		System.arraycopy(mess, 0, addr, 0, 4);
		InetAddress a = null;
		a = InetAddress.getByAddress(addr);
		System.arraycopy(mess, 4, addr, 0, 4);
	      int p = StreamMediator.intFromBytes(addr);

	      byte[] tmp = new byte[mess.length-8];
	      System.arraycopy(mess, 8, tmp, 0, mess.length-8);

		DatagramPacket sendPacket = new DatagramPacket(tmp, tmp.length, a, p);
		try {
			clientSocket.send(sendPacket);
		} catch (IOException e) {
			e.printStackTrace();
		}
	   }
```

```
	private DatagramPacket createPacket() throws FileNotFoundException, ClassNotFoundException,
IOException, NoSuchAlgorithmException, InvalidKeySpecException//11 full
	{
		System.out.println("creating packet");
		byte[] sendData = new byte[message.length()];
		sendData = message.getBytes();

		byte[] tmp = new byte[sendData.length + 9];
		byte[] opcode = new byte[1];
		opcode[0] = 2;
		System.arraycopy(opcode, 0, tmp, 0, 1);
		System.arraycopy(clientAddress.getIP().getAddress(), 0, tmp, 1, 4);
		byte[] pp = StreamMediator.intToBytes(clientAddress.getPort());
		System.arraycopy(pp, 0, tmp, 5, 4);
		System.arraycopy(sendData, 0, tmp, 9, sendData.length);
		sendData = tmp;

		Random rand = new Random();
		int rounds = 1;
		UserInfo[] tableOfUsers = us.getTableOfUsers();
		for(int i=0; i<tableOfUsers.length; ++i)
		{
			if(tableOfUsers[i].getAddress().getIP().equals(ipAddress))
			{
				sendData = Util.CryptoService.encrypt(sendData, Util.CryptoService.getPub-
licKeyFromFile( tableOfUsers[i].getKeyFileName() ));
				break;
			}
		}
		if(tableOfUsers .length > 2)
			rounds = rand.nextInt(tableOfUsers.length);
		if(rounds > 5)
			rounds = 5;
		for(int i=0; i < rounds; ++i)
		{
			int ind = rand.nextInt(tableOfUsers.length);


if(tableOfUsers[ind].getAddress().equals(clientAddress))//?????????????????????????????????????????????
??????????????????????????????????????????????
			{
				continue;
			}
			tmp = new byte[sendData.length + 9];
			opcode[0] = 1;
			System.arraycopy(opcode, 0, tmp, 0, 1);
			System.arraycopy(ipAddress.getAddress(), 0, tmp, 1, 4);
			pp =
StreamMediator.intToBytes(tableOfUsers[ind].getAddress().getPort());//&&&&&&&&&&&&&&&&&&&&&&&&&&&
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&7
			System.arraycopy(pp, 0, tmp, 5, 4);
			System.arraycopy(sendData, 0, tmp, 9, sendData.length);
			sendData = Util.CryptoService.encrypt(tmp,
Util.CryptoService.getPublicKeyFromFile(tableOfUsers[ind].getKeyFileName()) );
			ipAddress = tableOfUsers[ind].getAddress().getIP();
		}
		return new DatagramPacket(sendData, sendData.length, ipAddress, clientAddress.getPort());
	}
```

```
	private void sendPacket() throws IOException, ClassNotFoundException, NoSuchAlgorithmException,
InvalidKeySpecException, InterruptedException
	{
		byte[] check = Util.CryptoService.hash(message);
		//do
		for(int i=0; i<3; ++i)
		{
			DatagramPacket sendPacket = createPacket();//11
			System.out.println("packet has been create");
			clientSocket.send(sendPacket);
			Thread.sleep(1000);
			try
			{
				if(ack.getAcknowledgement(check))
					break;
			}
			catch (Error e)
			{
				e.printStackTrace();
			}
		}
		//while( !ack.getAcknowledgement(check) );
		//ack.getAcknowledgement(check);
	}

	public void run()
	{
		Util.CryptoService.generateKey(Config.Constants.clientPrivateKeyFileName, Config.Constants.-
clientPublicKeyFileName);
		System.out.println("client start");
		try {
			us.registerService();
		} catch (IOException e) {
			e.printStackTrace();
			System.out.println("critical failure: problem with register");
			return ;
		}
		System.out.println("register succes");
		try {
			us.getParticipants();
		} catch (IOException e) {
			e.printStackTrace();
			System.out.println("critical failure: can't get participants");
			return;
		}
		System.out.println("got participants");
		boolean end = false;
		while(!end)
		{
			while(mode == 0)
			{
				byte[] receiveData = new byte[Config.Constants.maxLengthData];
				DatagramPacket receivePacket = new DatagramPacket(receiveData, receive-
Data.length);
				try
				{
					clientSocket.receive(receivePacket);
				}
				catch (IOException e)
				{
```

```
                                        e.printStackTrace();
                                        System.out.println("failure: can't read message");
                                        break;
                                }
                                //System.out.println(receivePacket.getAddress().toString());
                                if(receivePacket.getAddress().equals(InetAddress.getLoopbackAddress()) )
                                        break;

                                byte[] tmp = {-1};
                                try
                                {
                                        tmp = Util.CryptoService.decrypt(receivePacket.getData(), Util.Cryp-
toService.getPrivateKeyFromFile(Config.Constants.clientPrivateKeyFileName));
                                }
                                catch (ClassNotFoundException | NoSuchAlgorithmException | Invalid-
KeySpecException | IOException e1)
                                {
                                        // TODO Auto-generated catch block
                                        e1.printStackTrace();
                                        continue;
                                }
                                byte operand = tmp[0];
                                if( operand == -1 )
                                {
                                        continue;
                                }
                                //System.out.println(operand);
                                byte[] mess = java.util.Arrays.copyOfRange(tmp, 1, tmp.length-1);
                                if( (operand&1) == 0)//receive
                                {
                                        System.out.println("operation: receive");
                                        try
                                        {
                                                receivePacket(mess, (operand&2) == 2);
                                        }
                                        catch (IOException | NoSuchAlgorithmException e)
                                        {
                                                e.printStackTrace();
                                                System.out.println("failure: can't read message");
                                        }
                                        System.out.println("receive done");
                                        break;
                                }
                                if( (operand&1) == 1)//transfer
                                {
                                        System.out.println("operation: transfer");
                                        try
                                        {
                                                transferPacket(mess);
                                        }
                                        catch (UnknownHostException e)
                                        {
                                                e.printStackTrace();
                                        }
                                        System.out.println("transfer done");
                                        break;
                                }

                                System.out.println("failure: unknown operation");
                        }
                        if(mode == 1)//send
```

```
                {
                        mode = 0;
                        System.out.println("operation: send");
                        try
                        {
                                sendPacket();
                        }
                        catch (ClassNotFoundException | NoSuchAlgorithmException | Invalid-
KeySpecException | IOException
                                                        | InterruptedException e)
                        {
                                e.printStackTrace();
                                continue;
                        }
                        System.out.println("send done");
                }
                if(mode == 2)//exit
                {
                        try
                        {
                                us.unregisterService();
                        }
                        catch (IOException e)
                        {
                                e.printStackTrace();
                        }
                        end=true;
                        System.out.println("unregister done");
                }
        }
        clientSocket.close();
        System.out.println("client stop");
    }

    public void sendMessage(String  ip, String message) throws IOException
    {
        ipAddress = InetAddress.getByName(ip);
        this.message = message;
        mode = 1;
        byte[] tmp = new byte[1];
        tmp[0] = 16;
        DatagramPacket dp = new DatagramPacket(tmp, 1, InetAddress.getLoopbackAddress(), clien-
tAddress.getPort());
        clientSocket.send(dp);
    }

    public void close() throws IOException
    {
        System.out.println("send closing packet");
        mode = 2;
        byte[] tmp = new byte[1];
        tmp[0] = 16;
        DatagramPacket dp = new DatagramPacket(tmp, 1, InetAddress.getLoopbackAddress(), clien-
tAddress.getPort());
        clientSocket.send(dp);
        System.out.println("send success");
    }
}
```

П.3.5. ФАЙЛ «Client/AckClient.java»

```
package Client;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.Socket;
import java.security.NoSuchAlgorithmException;
import java.security.interfaces.RSAPublicKey;
import java.security.spec.InvalidKeySpecException;
import java.util.Arrays;
import java.util.HashMap;

import Gate.Error;
import Gate.Gate;
import Util.StreamMediator;

public class AckClient
{
	InetAddress serverAddress;

	public AckClient(InetAddress server)
	{
		serverAddress = server;
	}

	public void setAcknowledgement(byte[] ack) throws IOException
	{
		Socket socket = new Socket(serverAddress, Config.Constants.portOfAck);
      byte[] contents = new byte[Config.Constants.maxLengthData];

      OutputStream os = socket.getOutputStream();
      contents[0] = 0;
      System.arraycopy(ack, 0, contents, 1, ack.length);
      os.write(contents, 0, 1 + ack.length);
      socket.shutdownOutput();
      socket.close();
	}

	public boolean getAcknowledgement(byte[] check) throws IOException, Error, ClassNotFoundExcep-
tion, NoSuchAlgorithmException, InvalidKeySpecException//7
	{
		System.out.println("try to get ack");
		/*++count;
		if(count > 3)
		{
			count = 0;
			getParticipants();//2
			return false;
		}*/

		Socket socket = new Socket(serverAddress, Config.Constants.portOfAck);
      OutputStream os = socket.getOutputStream();
      InputStream is = socket.getInputStream();
```

```
        os.write(1);

        OTClient ot = new OTClient();
        ot.obliviousTransfer(is, os, check);
        System.out.println("start fill the etalons");
        byte[][] etalons = new byte[2][Config.Constants.numberOfStoredBytes];
        etalons[0] = Util.StreamMediator.bytesFromInputStream(is);
        etalons[1] = Util.StreamMediator.bytesFromInputStream(is);

        System.out.println("start fill the gate");
        Util.StreamMediator.fileFromInputStream(is, "client_gate.bin");

        is.close();
        os.close();
        socket.close();
        System.out.println("end fill the gate");
                return evalueteGate(etalons, check, ot.getKeys());//8
            }

        private boolean evalueteGate(byte[][] etalons, byte[] check, byte[][] tableOfObliviousKeys) throws IOEx-
ception, Error, ClassNotFoundException, NoSuchAlgorithmException, InvalidKeySpecException
        {
                System.out.println("try to evaluete gate");
                HashMap<Integer, Gate> gc = new HashMap<Integer, Gate>();

                int id=0, id1, id2;
                byte[][][] table = new byte[2][2][Config.Constants.keyLength + 1];
                FileInputStream in = new FileInputStream(new File("client_gate.bin"));
                long start = System.nanoTime();
                int count=0;
                while (in.available() != 0)
                {
                        System.out.println("filling number " + count++);
                        id = StreamMediator.intFromFileInputStream(in);
                        id1 = StreamMediator.intFromFileInputStream(in);
                        id2 = StreamMediator.intFromFileInputStream(in);
                        for(int i=0; i<2; ++i)
                        {
                                for(int j=0; j<2; ++j)
                                {
                                        table[i][j] = StreamMediator.bytesFromInputStream(in);
                                }
                        }
                        if(id < Config.Constants.numberOfInputGates)
                        {
                                int i = id / 8;
                                int j = id % 8;
                                boolean b = ( check[i]&(1<<j) ) == (1<<j);
                                RSAPublicKey key = (RSAPublicKey)
Util.CryptoService.getPublicKeyFromFile(Config.Constants.serverPublicKeyFileName);//"pub.key");
                                if(b)
                                {
                                        table[0][0] = Util.CryptoService.finalizeOT(table[1][0], tableOfOblivi-
ousKeys[id], key);
                                }
                                else
                                {
                                        table[0][0] = Util.CryptoService.finalizeOT(table[0][0], tableOfOblivi-
ousKeys[id], key);
```

```
                        }
                }
                Gate first=null, second=null;
                if(id1 != -1)
                {
                        first = (Gate) gc.get(id1);
                }
                if(id2 != -1)
                {
                        second = (Gate) gc.get(id2);
                }
                Gate tmp = new Gate(id, table, first, second);
                gc.put(id, tmp);
        }
        in.close();
        long end = System.nanoTime();
        long time = end - start;
        System.out.println("recreating gate time:" + time/1e9);

        Gate gate = (Gate) gc.get(id);
        if(gate == null)
        {
                throw new Error();
        }

        /*byte[][] etalons = new byte[2][Config.Constants.numberOfStoredBytes];
        in = new FileInputStream(new File("etalons.bin"));
        in.read(etalons[0], 0, Config.Constants.numberOfStoredBytes);
        in.read(etalons[1], 0, Config.Constants.numberOfStoredBytes);
        in.close();*/

        start = System.nanoTime();
        byte[] res;
        res = gate.calc();
        end = System.nanoTime();
        time = end-start;
        System.out.println("calculation time:" + time/1e9);

        System.out.println("etalon0:" + etalons[0].length);
        System.out.println(etalons[0][0]);
        System.out.println(etalons[0][etalons[0].length-1]);
        System.out.println("etalon1:" + etalons[1].length);
        System.out.println(etalons[1][0]);
        System.out.println(etalons[1][etalons[1].length-1]);
        System.out.println("res:" + res.length);
        System.out.println(res[0]);
        System.out.println(res[res.length-1]);

        if( Arrays.equals(res, etalons[0]) )
                return false;
        else if( Arrays.equals(res, etalons[1]) )
                return true;
        else
                throw new Error();
    }
}
```

П.3.6. ФАЙЛ «Client/OTClient.java»

```
package Client;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.security.NoSuchAlgorithmException;
import java.security.interfaces.RSAPublicKey;
import java.security.spec.InvalidKeySpecException;

public class OTClient
{
	byte[][] tableOfObliviousKeys;

	public OTClient()
	{
		tableOfObliviousKeys = new byte[Config.Constants.numberOfBits][Config.Constants.keyLength
+ 1];
	}

	private void atomaryOT(InputStream in, OutputStream out, RSAPublicKey key, boolean b, byte[] save)
throws IOException
	{
		byte[] x0,x1;
		System.out.println("reading x");
		x0 = Util.StreamMediator.bytesFromInputStream(in);
		x1 = Util.StreamMediator.bytesFromInputStream(in);
		System.out.println("x read");

		byte[] v;

		System.out.println("encrypt v");
		if(b)
		{
			v = Util.CryptoService.encryptOT(x1, key, save);
		}
		else
		{
			v = Util.CryptoService.encryptOT(x0, key, save);
		}
		Util.StreamMediator.bytesToOutputStream(v, out);
		out.flush();
		System.out.println("v send");
	}

	public void obliviousTransfer(InputStream in, OutputStream out, byte[] check) throws IOException,
ClassNotFoundException, NoSuchAlgorithmException, InvalidKeySpecException
	{
		System.out.println("OT work need to be done");

		RSAPublicKey key = (RSAPublicKey) Util.CryptoService.getPublicKeyFromFile(Config.Con-
stants.serverPublicKeyFileName);
		for(int i=0; i < Config.Constants.numberOfInputGates; ++i)
		{
			int posByte = i / 8;
			int posBit = i % 8;
			System.out.println(i);
```

```
                        boolean b = ( check[posByte]&(1<<posBit) ) == (1<<posBit);
                        atomaryOT(in, out, key,  b, tableOfObliviousKeys[i]);
                }
        }

        public byte[][] getKeys()
        {
                return tableOfObliviousKeys;
        }
}
```

П.3.7. ФАЙЛ «Client/UsersClient.java»

```
package Client;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.UUID;

import Util.Address;
import Util.StreamMediator;
import Util.UserInfo;

public class UsersClient
{
        InetAddress serverAddress;//outer
        Address clientAddress;//outer
        UUID sysID;
        UserInfo[] tableOfUsers;

        public UsersClient(InetAddress server, Address client) throws UnknownHostException
        {
                clientAddress = client;
                serverAddress = server;
                tableOfUsers = null;
                sysID = new UUID(0, 0);
        }

        public void registerService() throws IOException
        {
                System.out.println("starting register");
                Socket socket = new Socket(serverAddress, Config.Constants.portOfUsers);

      OutputStream os = socket.getOutputStream();
      os.write(1);
      //StreamMediator.bytesToOutputStream(clientAddress.getIP().getAddress(), os);
      StreamMediator.intToOutputStream(clientAddress.getPort(), os);
      StreamMediator.fileToOutputStream(os, Config.Constants.clientPublicKeyFileName);
      socket.shutdownOutput();

      InputStream is = socket.getInputStream();
      long lower = StreamMediator.longFromInputStream(is);
      long upper = StreamMediator.longFromInputStream(is);
      sysID = new UUID(upper, lower);
```

```
socket.close();
    }

    public void unregisterService() throws IOException
    {
            System.out.println("staring unregister");
            Socket socket = new Socket(serverAddress, Config.Constants.portOfUsers);

OutputStream os = socket.getOutputStream();
os.write(0);

StreamMediator.longToOutputStream(sysID.getLeastSignificantBits(), os);
StreamMediator.longToOutputStream(sysID.getMostSignificantBits(), os);
socket.shutdownOutput();

if(socket.getInputStream().read() == 1)
{
    System.out.println("unregister went ok");
}
else
{
    System.out.println("unregister went not ok");
}
socket.close();
    }

    public void getParticipants() throws IOException
    {
            System.out.println("reading participants");
            Socket socket = new Socket(serverAddress, Config.Constants.portOfUsers);
byte[] contents = new byte[Config.Constants.maxLengthData];

OutputStream os = socket.getOutputStream();
contents[0] = 2;
os.write(contents, 0, 1);
socket.shutdownOutput();

InputStream is = socket.getInputStream();
int num = StreamMediator.intFromInputStream(is);
System.out.println("there is " + num + " participants");
tableOfUsers = new UserInfo[num];
for(int i=0; i<num; ++i)
{
    byte[] ip = StreamMediator.bytesFromInputStream(is);
    int port = StreamMediator.intFromInputStream(is);
    Address addr = new Address(ip, port);
    String key = "users/" + addr.getIP().toString() + ".key";
    StreamMediator.fileFromInputStream(is, key);
    tableOfUsers[i] = new UserInfo(addr, key);
}
socket.close();
            //fill the tables from server
    }

    public UserInfo[] getTableOfUsers()
    {
            return tableOfUsers.clone();
    }
```

```
	public UUID getSysId()
	{
		return sysID;
	}
}
```

П.3.8. ФАЙЛ «Server/ServerAck.java»

```
package Server;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.security.NoSuchAlgorithmException;
import java.security.spec.InvalidKeySpecException;

import Gate.Error;
import Gate.GateConstructor;
import Util.Queue;
import Util.StreamMediator;

public class ServerAck extends Thread {
	int mode;
	Socket serverSocket;
	ServerSocket welcomeSocket;
	Queue q;
	GateConstructor gc;


	public ServerAck() throws IOException
	{
		mode = 0;
		serverSocket = null;
		welcomeSocket = new ServerSocket(Config.Constants.portOfAck);
		q = new Queue();
		gc = null;
	}


	private void createGate() throws Error
	{
		System.out.println("creating gate");
		byte[][] tmp = q.get();


		long start = System.nanoTime();
		gc = new GateConstructor(tmp);
		long end = System.nanoTime();
		long time = end-start;
		System.out.println("creation time:" + time/1e9);
	}


	private void giveEtalons(OutputStream out) throws IOException
	{
		System.out.println("giving etalons");
		byte[][] eta = gc.getEtalon();
		Util.StreamMediator.bytesToOutputStream(eta[0], out);
		Util.StreamMediator.bytesToOutputStream(eta[1], out);
```

```
	}

	private void giveGate(InputStream in, OutputStream out) throws IOException
	{
		System.out.println("sending gate");
		gc.getGate().print();

		StreamMediator.fileToOutputStream(out, "gate.bin");
	   System.out.println("send ok");
	}

	private void set(InputStream in, OutputStream out) throws IOException
	{
		System.out.println("setting hash");
		byte[] tmp = new byte[Config.Constants.hashLength];
		in.read(tmp, 0, Config.Constants.hashLength);
		q.push(tmp);
		System.out.println("set done");
	}

	private void sendGate(InputStream in, OutputStream out) throws Error, IOException, ClassNotFoundEx-
ception, NoSuchAlgorithmException, InvalidKeySpecException
	{
		createGate();
		OTServer ot = new OTServer( gc.collectOTinput() );
		ot.obliviousTransfer(in, out);

		for(int i=0; i < Config.Constants.numberOfInputGates; ++i)
		{
			gc.getInputGate(i).set(true, ot.getArr(i, 1));
			gc.getInputGate(i).set(false, ot.getArr(i, 0));
		}

		giveEtalons(out);
		giveGate(in, out);
		out.close();
		System.out.println("all ok");
	}

	public void run()
	{
		Util.CryptoService.generateKey(Config.Constants.serverPrivateKeyFileName, Config.Con-
stants.serverPublicKeyFileName);
		System.out.println("server of acknowledgement start");
		boolean end = false;
		while(!end)
		{
			InputStream inFromClient;
			OutputStream outToClient;
			try {
				serverSocket = welcomeSocket.accept();
				inFromClient = serverSocket.getInputStream();
				outToClient = serverSocket.getOutputStream();
			} catch (IOException e1) {
				e1.printStackTrace();
				System.out.println("critical failure: problem with socket");
				continue;
			}
```

```
while(mode == 0)
{
	int operand = -1;
	try {
		operand = inFromClient.read();
	} catch (IOException e) {
		e.printStackTrace();
		System.out.println("failure: can't read socket");
		break;
	}
	//System.out.println(operand);
	if(operand == 0)//set
	{
		System.out.println("operation: set");
		try
		{
			set(inFromClient, outToClient);
		}
		catch (IOException e)
		{
			e.printStackTrace();
		}
		break;
	}

	if(operand == 1)//give gate
	{
		System.out.println("operation: get");
		try
		{
			sendGate(inFromClient, outToClient);
		}
		catch (Error | IOException e)
		{
			e.printStackTrace();
		}
		catch (ClassNotFoundException e)
		{
			e.printStackTrace();
		} catch (NoSuchAlgorithmException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (InvalidKeySpecException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		break;
	}

	System.out.println("failure: unknown operation");
}
try {
	serverSocket.close();
} catch (IOException e) {
	e.printStackTrace();
	System.out.println("failure: problem with closing socket");
	return ;
}
if(mode == 1)//exit
```

```
                {
                        end=true;
                }
        }
        try {
                serverSocket.close();
        } catch (IOException e) {
                e.printStackTrace();
                System.out.println("failure: problem with closing socket");
                return ;
        }
        System.out.println("server of acknowledgement stop");
    }

    public void close() throws IOException
    {
        System.out.println("sending close packet");
        mode = 1;
        InetAddress local = InetAddress.getLocalHost();
        Socket socket = new Socket(local, Config.Constants.portOfAck);
        socket.getOutputStream().write(10);
        socket.close();
        System.out.println("sending done");
    }
}
```

П.3.9. ФАЙЛ «Server/OTServer.java»

```
package Server;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.math.BigInteger;
import java.security.NoSuchAlgorithmException;
import java.security.interfaces.RSAPrivateKey;
import java.security.spec.InvalidKeySpecException;
import java.util.Random;

public class OTServer
{
    byte[][][] arr;//bitIndex of table g(numberOfBits), bitState of table g(0,1), length of bitValue(numberOfS-
toredBytes)

    public OTServer(byte[][][] OTinput)
    {
        arr = OTinput.clone();
        //bitIndex of table g(numberOfBits), bitState of table g(0,1), length of bitValue(numberOfStored-
Bytes)
    }

    private void atomaryOT(InputStream in, OutputStream out, byte[][] g, RSAPrivateKey key) throws IOEx-
ception
    {
        byte[][] res = g;
        //random x0 x1
        BigInteger x0 = new BigInteger(Config.Constants.numberOfStoredBytes * 8, new Random());
        BigInteger x1 = new BigInteger(Config.Constants.numberOfStoredBytes * 8, new Random());
```

```
                System.out.println("Sending x");
                Util.StreamMediator.bytesToOutputStream(x0.toByteArray(), out);
                Util.StreamMediator.bytesToOutputStream(x1.toByteArray(), out);

                System.out.println("x send");
                System.out.println("reading v");
                byte[] v = Util.StreamMediator.bytesFromInputStream(in);
                System.out.println("read v");
                byte[] k0 = Util.CryptoService.decryptOT(v, x0.toByteArray(), key);
                byte[] k1 = Util.CryptoService.decryptOT(v, x1.toByteArray(), key);
                System.out.println("get k");

                //res[0] = res[0] + k0;
                BigInteger var = new BigInteger(res[0]);
                BigInteger var2 = new BigInteger(k0);
                res[0] = var.add(var2).toByteArray();
                //res[1] = res[1] + k1;
                var = new BigInteger(res[1]);
                var2 = new BigInteger(k1);
                res[1] = var.add(var2).toByteArray();

                System.out.println("get answer");
                g = res.clone();
        }

        public void obliviousTransfer(InputStream in, OutputStream out) throws IOException, ClassNot-
FoundException, NoSuchAlgorithmException, InvalidKeySpecException
        {
                System.out.println("OT work need to be done");

                RSAPrivateKey key = (RSAPrivateKey) Util.CryptoService.getPrivateKeyFromFile(Config.Con-
stants.serverPrivateKeyFileName);

                for(int i=0; i < Config.Constants.numberOfInputGates; ++i)
                {
                        System.out.println(i);
                        atomaryOT(in, out, arr[i], key);
                }
        }

        public byte[] getArr(int bitIndex, int bitState)
        {
                return arr[bitIndex][bitState];
        }

}
```

П.3.10. ФАЙЛ «Server/ServerUsers.java»

```
package Server;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.IOException;
import java.net.*;
import java.util.UUID;
```

```
import Util.Address;
import Util.StreamMediator;
import Util.UserInfo;

import java.util.HashMap;


public class ServerUsers extends Thread {
	int mode;
	HashMap<UUID, UserInfo> table;
	Socket serverSocket;
	ServerSocket welcomeSocket;

	public ServerUsers() throws IOException
	{
		table = new HashMap<UUID, UserInfo >();
		welcomeSocket = new ServerSocket(Config.Constants.portOfUsers);
		mode = 0;
	}

	private void unregister(InputStream in, OutputStream out) throws IOException
	{
		System.out.println("starting unregister");

		long lower = StreamMediator.longFromInputStream(in);
		long upper = StreamMediator.longFromInputStream(in);
		UUID check = new UUID(upper, lower);

		if( table.containsKey(check) )
		{
			table.remove(check);
			out.write(1);
			System.out.println("succes");
		}
		else
		{
			out.write(0);
			System.out.println("fail");
		}
		out.flush();
	}

	private void register(InputStream in, OutputStream out) throws IOException
	{
		System.out.println("starting register");
		//byte[] addr = Util.StreamMediator.bytesFromInputStream(in);
		int port = StreamMediator.intFromInputStream(in);
		Address user = new Address(/*addr*/ serverSocket.getInetAddress().getAddress(), port);
		String key = "users/" + user.getIP().toString() + ".key";
		Util.StreamMediator.fileFromInputStream(in, key);
		System.out.println("registering " + user.getIP().toString() + ":" + user.getPort());



		UUID check = UUID.randomUUID();
		while(table.containsKey(check))
			check = UUID.randomUUID();
		System.out.println(check.toString());
```

```
		table.put(check, new UserInfo(user, key));
		StreamMediator.longToOutputStream(check.getLeastSignificantBits(), out);
		StreamMediator.longToOutputStream(check.getMostSignificantBits(), out);
		System.out.println("succes");

		out.flush();
	}

	private void writeParticipants(OutputStream out) throws IOException
	{
		System.out.println("starting write participants");
		StreamMediator.intToOutputStream(table.size(), out);
		for(UserInfo i : table.values() )
		{
			StreamMediator.bytesToOutputStream(i.getAddress().getIP().getAddress(), out);
			StreamMediator.intToOutputStream(i.getAddress().getPort(), out);
			StreamMediator.fileToOutputStream(out, i.getKeyFileName());
		}
		out.flush();
		System.out.println("succes");
	}

	public void run()
	{
		System.out.println("server of users start");
		boolean end = false;
		while(!end)
		{
			InputStream inFromClient;
			OutputStream outToClient;
			try {
				serverSocket = welcomeSocket.accept();
				inFromClient = serverSocket.getInputStream();
				outToClient = serverSocket.getOutputStream();
			} catch (IOException e1) {
				e1.printStackTrace();
				System.out.println("critical failure: problem with socket");
				continue;
			}

			while(mode == 0)
			{
				int operand = -1;
				try {
					operand = inFromClient.read();
				} catch (IOException e) {
					e.printStackTrace();
					System.out.println("failure: can't read socket");
					break;
				}
				//System.out.println(operand);
				if(operand == 0)//unregister
				{
					try {
						unregister(inFromClient, outToClient);
					} catch (IOException e) {
						e.printStackTrace();
						System.out.println("failure: problem with unregister");
					}
					break;
```

```
                                }

                                if(operand == 1)//register
                                {
                                        try {
                                                register(inFromClient, outToClient);
                                        } catch (IOException e) {
                                                e.printStackTrace();
                                                System.out.println("failure: problem with register");
                                        }
                                        break;
                                }

                                if(operand == 2)//get participants
                                {
                                        try {
                                                writeParticipants(outToClient);
                                        } catch (IOException e) {
                                                e.printStackTrace();
                                                System.out.println("failure: problem with writing participants");
                                        }
                                        break;
                                }

                                System.out.println("failure: unknown operation");
                        }
                        if(mode == 1)//exit
                        {
                                end=true;
                        }
                }
                try {
                        serverSocket.close();
                } catch (IOException e) {
                        e.printStackTrace();
                        System.out.println("failure: problem with closing socket");
                        return ;
                }
                System.out.println("server of users stop");
        }

        public void close() throws IOException
        {
                mode = 1;
                InetAddress local = InetAddress.getLocalHost();
                Socket socket = new Socket(local, Config.Constants.portOfUsers);
                socket.getOutputStream().write(10);
                socket.close();
        }
}
```

П.3.11. ФАЙЛ «Util/Address.java»

```
package Util;
import java.io.IOException;
import java.io.InputStream;
import java.net.InetAddress;
import java.net.UnknownHostException;
```

```
public class Address {
	InetAddress ip;
	int port;

	public Address(byte[] addr, int p)
	{
		try {
			ip = InetAddress.getByAddress(addr);
		} catch (UnknownHostException e) {
			e.printStackTrace();
			return ;
		}
		port = p;
	}

	public InetAddress getIP()
	{
		return ip;
	}

	public int getPort()
	{
		return port;
	}

	public boolean equals(Object b)
	{
		if (this == b)
	    return true;
	  if (b == null)
	    return false;
	  if (getClass() != b.getClass())
	    return false;
	  Address tmp = (Address) b;
		return this.ip.equals(tmp.ip) && this.port == tmp.port;
	}

	public int hashCode()
	{
		return 31*ip.hashCode() + port;
	}

	public static Address fill(InputStream in) throws IOException
	{
		byte[] addr = new byte[4];
		in.read(addr);
		int port = StreamMediator.intFromInputStream(in);
		return new Address(addr,port);
	}
}
```

П.3.12. ФАЙЛ «Util/ByteArrayWrapper.java»

```
package Util;

public class ByteArrayWrapper
	{
```

```
		byte[] inner;
		public ByteArrayWrapper()
		{
			inner = new byte[Config.Constants.numberOfStoredBytes];
		}

		public void set(byte[] val)
		{
			inner = val.clone();
		}

		public byte[] get()
		{
			return inner.clone();
		}

		public boolean equals(Object b)
		{
			if (this == b)
		    return true;
		  if (b == null)
		    return false;
			return inner.equals(b);
		}
	}
```

П.3.13. ФАЙЛ «Util/CryptoService.java»

```
package Util;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.math.BigInteger;
import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;
import java.util.Random;

import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;

import Gate.Error;


public class CryptoService
{
```

```
        public static void generateKey(String Private_Key_File, String Public_Key_File)
        {
           try
           {
                final KeyPairGenerator keyGen = KeyPairGenerator.getInstance(Config.Constants.cipherAlgo-
rithm);
                keyGen.initialize(2048);
                final KeyPair key = keyGen.generateKeyPair();
                RSAPublicKey key2 = (RSAPublicKey) key.getPublic();
                RSAPrivateKey key3 = (RSAPrivateKey) key.getPrivate();

                /*if( key2.getModulus().equals(key3.getModulus()) )
                        {
                                BigInteger test = new BigInteger("6");
                                test.modPow(key2.getPublicExponent(), key2.getModulus()).modPow(key3.get-
PrivateExponent(), key3.getModulus());
                                System.out.println(test.toString());
                        }
                        else
                        {
                                System.out.println("modulus error");
                        }*/

                File privateKeyFile = new File(Private_Key_File);
                File publicKeyFile = new File(Public_Key_File);

                // Create files to store public and private key
                if (privateKeyFile.getParentFile() != null)
                {
                        privateKeyFile.getParentFile().mkdirs();
                }
                privateKeyFile.createNewFile();

                if (publicKeyFile.getParentFile() != null)
                {
                        publicKeyFile.getParentFile().mkdirs();
                }
                publicKeyFile.createNewFile();

                // Saving the Public key in a file
                X509EncodedKeySpec x509EncodedKeySpec = new X509EncodedKeySpec( key2.getEn-
coded());
              FileOutputStream fos = new FileOutputStream(publicKeyFile);
              fos.write(x509EncodedKeySpec.getEncoded());
              fos.close();


                // Saving the Private key in a file
              PKCS8EncodedKeySpec pkcs8EncodedKeySpec = new PKCS8EncodedKeySpec(key3.getEn-
coded());
              fos = new FileOutputStream(privateKeyFile);
              fos.write(pkcs8EncodedKeySpec.getEncoded());
              fos.close();
           }
           catch (Exception e)
           {
                e.printStackTrace();
           }

        }
```

```
	public static byte[] encrypt(byte[] input, PublicKey key)
	{
		//return input;

	   byte[] cipherText = null;
	   try
	   {
		byte[] randKey = new byte[Config.Constants.innerKeyLength];
		Random rand = new Random();
		rand.nextBytes(randKey);

		final Cipher cipher = Cipher.getInstance(Config.Constants.cipherAlgorithm);

		cipher.init(Cipher.ENCRYPT_MODE, key);
		byte[] tmp = cipher.doFinal(randKey);
		byte[] tmpLength = Util.StreamMediator.intToBytes(tmp.length);
		System.out.println("AES key length" + tmp.length);

		final Cipher innerCipher = Cipher.getInstance(Config.Constants.innerCipherAlgorithm);
		SecretKeySpec innerKey = new SecretKeySpec(randKey, Config.Constants.innerCipherAlgo-
rithm.substring(0, 3));
		IvParameterSpec iv = new IvParameterSpec("0102030405060708".getBytes());
		innerCipher.init(Cipher.ENCRYPT_MODE, innerKey, iv);

		int len = input.length;
		len += 16 - (len%16);
		byte[] message = new byte[len];
		System.arraycopy(input, 0, message, 0, input.length);
		byte[] tmp2 = innerCipher.doFinal(message);

		cipherText = new byte[4 + tmp.length + tmp2.length];
		System.arraycopy(tmpLength, 0, cipherText, 0, 4);
		System.arraycopy(tmp, 0, cipherText, 4, tmp.length);
		System.arraycopy(tmp2, 0, cipherText, 4+tmp.length, tmp2.length);
	   }
	   catch (Exception e)
	   {
		e.printStackTrace();
	   }
	   return cipherText;

	}

	public static byte[] decrypt(byte[] text, PrivateKey key)
	{
		//return text;

	   byte[] dectyptedText = null;
	   try
	   {
		byte[] tmpLength = new byte[4];
		System.arraycopy(text, 0, tmpLength, 0, 4);
		byte[] cipherKey = new byte[Util.StreamMediator.intFromBytes(tmpLength)];
		System.out.println("AES key length" + Util.StreamMediator.intFromBytes(tmpLength));
		System.arraycopy(text, 4, cipherKey, 0, cipherKey.length);
```

```
                final Cipher cipher = Cipher.getInstance(Config.Constants.cipherAlgorithm);
                cipher.init(Cipher.DECRYPT_MODE, key);
                byte[] plainKey = cipher.doFinal(cipherKey);
                SecretKeySpec innerKey = new SecretKeySpec(plainKey, Config.Constants.innerCipherAlgo-
rithm.substring(0, 3));

                final Cipher innerCipher = Cipher.getInstance(Config.Constants.innerCipherAlgorithm);
                IvParameterSpec iv = new IvParameterSpec("0102030405060708".getBytes());
                innerCipher.init(Cipher.DECRYPT_MODE, innerKey, iv);
                int len = text.length -4 -cipherKey.length;
                len += 16 - (len%16);
                byte[] message = new byte[len];
                System.arraycopy(text, 4+cipherKey.length, message, 0, text.length -4 -cipherKey.length);
                dectyptedText = innerCipher.doFinal(message);
            }
            catch (Exception ex)
            {
                ex.printStackTrace();
            }

            return dectyptedText;

        }

        public static byte[] hash(String str) throws NoSuchAlgorithmException
        {
                byte[] res = null;
                MessageDigest md;
                md = MessageDigest.getInstance(Config.Constants.hashAlgorithm);

                md.reset();
                md.update(str.getBytes());
                res = md.digest();

                return res;
        }

        public static byte[] encryptOT(byte[] x, RSAPublicKey key, byte[] save)
        {
                BigInteger res = new BigInteger(x);
                //random k
                BigInteger k = new BigInteger(Config.Constants.numberOfStoredBytes * 8, new Random());
                BigInteger e = key.getPublicExponent();



                //save k to tableOfObliviousKeys
                byte[] tmp = k.toByteArray();
                System.arraycopy( tmp, 0, save, save.length - tmp.length, tmp.length );
                BigInteger modulus = key.getModulus();
                //k = k ^ e
                k = k.modPow(e, modulus);
                //res = x + k
                res = res.add(k);
                res = res.mod(key.getModulus());
                return res.toByteArray();
        }
```

```
	public static byte[] decryptOT(byte[] v, byte[] x, RSAPrivateKey key)
	{
		BigInteger res = new BigInteger(v);
		BigInteger tmp = new BigInteger(x);
		//res = v - x;
		res = res.subtract(tmp);
		//res = res^d;
		res = res.modPow(key.getPrivateExponent(), key.getModulus());
		return res.toByteArray();
	}

	public static byte[] finalizeOT(byte[] m, byte[] k, RSAPublicKey key)
	{
		BigInteger res = new BigInteger(m);
		BigInteger tmp = new BigInteger(k);
		res = res.subtract(tmp);

		BigInteger modulus = key.getModulus();
		res = res.mod(modulus);
		if( res.compareTo( modulus.divide(new BigInteger("2")) ) > 0)
		{
			res = res.subtract(modulus);
		}
		return res.toByteArray();
	}

	public static PrivateKey getPrivateKeyFromFile(String path) throws FileNotFoundException, IOException, ClassNotFoundException, NoSuchAlgorithmException, InvalidKeySpecException
	{
		File filePrivateKey = new File(path);
		FileInputStream fis = new FileInputStream(path );
	byte[] encodedPrivateKey = new byte[(int) filePrivateKey.length()];
	fis.read(encodedPrivateKey);
	fis.close();

	KeyFactory keyFactory = KeyFactory.getInstance(Config.Constants.cipherAlgorithm);
	PKCS8EncodedKeySpec privateKeySpec = new PKCS8EncodedKeySpec(encodedPrivateKey);
	return keyFactory.generatePrivate(privateKeySpec);
	}

	public static PublicKey getPublicKeyFromFile(String path) throws FileNotFoundException, IOException, ClassNotFoundException, NoSuchAlgorithmException, InvalidKeySpecException
	{
		File filePublicKey = new File(path);
	FileInputStream fis = new FileInputStream(path);
	byte[] encodedPublicKey = new byte[(int) filePublicKey.length()];
	fis.read(encodedPublicKey);
	fis.close();

	KeyFactory keyFactory = KeyFactory.getInstance(Config.Constants.cipherAlgorithm);
	X509EncodedKeySpec publicKeySpec = new X509EncodedKeySpec(encodedPublicKey);
	return keyFactory.generatePublic(publicKeySpec);
	}
}
```

П.3.14. ФАЙЛ «Util/Queue.java»

```
package Util;

public class Queue {
	public ByteArrayWrapper[] array;
	int head;
	int tail;

	public Queue()
	{
		array = new ByteArrayWrapper[Config.Constants.numberOfValues];
		for(int i=0; i<Config.Constants.numberOfValues;++i)
			array[i] = new ByteArrayWrapper();
		head = tail = 0;
	}

	public boolean check(byte[] hash)
	{
		for(int i=0; i<Config.Constants.numberOfValues; ++i)
		{
			if(array[i] != null && array[i].equals(hash))
				return true;
		}
		return false;
	}

	public void push(byte[] hash)
	{
		if(check(hash))
			return ;
		array[tail] = new ByteArrayWrapper();
		array[tail].set(hash);
		++tail;
		if(tail >= Config.Constants.numberOfValues)
			tail = 0;
		if(tail == head)
		{
			++head;
		}
	}

	public byte[][] get()
	{
		byte[][] res = new byte[Config.Constants.numberOfValues][Config.Constants.hashLength];
		for(int i=0; i<Config.Constants.numberOfValues; ++i)
		{
			res[i] = array[i].get();
		}
		return res;
	}
}
```

П.3.15. ФАЙЛ «Util/StreamMediator.java»

```
package Util;
import java.io.BufferedInputStream;
```

```
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.nio.ByteBuffer;

public class StreamMediator {
	public static int intFromBytes(byte[] in)
	{
		ByteBuffer buffer = ByteBuffer.allocate(Integer.BYTES);
	   buffer.put(in);
	   buffer.flip();//need flip
	   return buffer.getInt();
	}

	public static byte[] intToBytes(int num)
	{
		ByteBuffer buffer = ByteBuffer.allocate(Integer.BYTES);
	   buffer.putInt(num);
	   return buffer.array().clone();
	}

	public static long longFromInputStream(InputStream in) throws IOException
	{
		byte[] buf = new byte[8];
		in.read(buf, 0, 8);
		ByteBuffer buffer = ByteBuffer.allocate(Long.BYTES);
	   buffer.put(buf);
	   buffer.flip();//need flip
	   return buffer.getLong();
	}

	public static int intFromInputStream(InputStream in) throws IOException
	{
		byte[] buf = new byte[4];
		in.read(buf, 0, 4);
		ByteBuffer buffer = ByteBuffer.allocate(Integer.BYTES);
	   buffer.put(buf);
	   buffer.flip();//need flip
	   return buffer.getInt();
	}

	public static void intToOutputStream(int num, OutputStream out) throws IOException
	{
		ByteBuffer buffer = ByteBuffer.allocate(Integer.BYTES);
	   buffer.putInt(num);
	   out.write(buffer.array());
	}

	public static void intToFileOutputStream(int num, FileOutputStream out) throws IOException
	{
		ByteBuffer buffer = ByteBuffer.allocate(Integer.BYTES);
	   buffer.putInt(num);
	   out.write(buffer.array());
	}
```

```
    public static void longToOutputStream(long num, OutputStream out) throws IOException
    {
       ByteBuffer buffer = ByteBuffer.allocate(Long.BYTES);
       buffer.putLong(num);
       out.write(buffer.array());
    }

    public static int intFromFileInputStream(FileInputStream in) throws IOException
    {
            byte[] buf = new byte[4];
            in.read(buf, 0, 4);
            ByteBuffer buffer = ByteBuffer.allocate(Integer.BYTES);
       buffer.put(buf);
       buffer.flip();//need flip
       return buffer.getInt();
    }

    public static void fileToOutputStream(OutputStream out, String filePath) throws IOException
    {
            File myFile = new File(filePath);
            int len = (int) myFile.length();
            intToOutputStream(len, out);
            byte[] mybytearray = new byte[len];
       BufferedInputStream bis = new BufferedInputStream(new FileInputStream(myFile));
       bis.read(mybytearray, 0, mybytearray.length);
       out.write(mybytearray, 0, mybytearray.length);
       out.flush();
       bis.close();
    }

    public static void fileFromInputStream(InputStream is, String name) throws IOException
    {
            int len = intFromInputStream(is);
            int bufLen = Math.min(len, Config.Constants.maxLengthData);
            byte[] contents = new byte[bufLen];
            FileOutputStream fos = new FileOutputStream(name);
BufferedOutputStream bos = new BufferedOutputStream(fos);

//No of bytes read in one read() call
int bytesRead = 0;
int allBytesRead = 0;

while(allBytesRead < len)
{
    bytesRead = is.read(contents);
    allBytesRead += bytesRead;
   bos.write(contents, 0, bytesRead);
}

bos.flush();
bos.close();
fos.close();
    }

    public static void bytesToOutputStream(byte[] arr, OutputStream out) throws IOException
    {
            intToOutputStream(arr.length, out);
            out.write(arr, 0, arr.length);
    }
```

```
	public static byte[] bytesFromInputStream(InputStream in) throws IOException
	{
		int len = intFromInputStream(in);
		byte[] res = new byte[len];
		in.read(res, 0, len);
		return res.clone();
	}
}
```

П.3.16. ФАЙЛ «Util/UserInfo.java»

```
package Util;

public class UserInfo
{
	Address userAddress;
	String keyFileName;

	public UserInfo(Address addr, String key)
	{
		userAddress = addr;
		keyFileName = key;
	}

	public Address getAddress()
	{
		return userAddress;
	}

	public String getKeyFileName()
	{
		return keyFileName;
	}
}
```

П.3.17. ФАЙЛ «Config/Constants.java»

```
package Config;

public class Constants {
	public static final String hashAlgorithm = "MD5";
	public static final int hashLength = 16;
	public static final int numberOfBits = hashLength * 8;
	public static final int numberOfInputGates = numberOfBits;

	public static final String serverGateFileName = "gate.bin";
	public static final String clientGateFileName = "client_gate.bin";
	public static final int numberOfBytes = 10;//length of garbled strings
	public static final int numberOfStoredBytes = Config.Constants.numberOfBytes + 1;

	public static final int numberOfValues = 10000;
	public static final int portOfUsers = 9981;
	public static final int portOfAck = 9982;
	public static final int portOfClient = 9978;
	public static final int maxLengthData = 524288;//65536;
```

```
	public static final String cipherAlgorithm = "RSA";
	public static final int keyLength = 256;
	public static final String serverPublicKeyFileName = "server/pub.key";
	public static final String serverPrivateKeyFileName = "server/priv.key";
	public static final String clientPublicKeyFileName = "client/pub.key";
	public static final String clientPrivateKeyFileName = "client/priv.key";
	public static final String innerCipherAlgorithm = "AES/CBC/NoPadding";
	public static final int innerKeyLength = 16;
}
```

П.3.18. ФАЙЛ «Gate/Gate.java»

```
package Gate;
import java.util.Random;

import Util.StreamMediator;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class Gate {
	protected Gate a;
	protected Gate b;
	private byte[] c0;
	private byte[] c1;
	private boolean p;
	protected byte[][][] g;
	private Gate pred;
	private byte[] value;
	private static int countId = 0;
	private int id;
	private boolean printed;


	public Gate(Gate first, Gate second)
	{
		id = countId++;
		printed = false;
		c0=grableStr();
		c1=grableStr();
		while( c0.equals(c1) )
			c1=grableStr();
		Random rand = new Random();
		p = rand.nextBoolean();
		if(p)
		{
			c0[Config.Constants.numberOfStoredBytes - 1] = 1;
			c1[Config.Constants.numberOfStoredBytes - 1] = 0;
		}
		else
		{
			c0[Config.Constants.numberOfStoredBytes - 1] = 0;
			c1[Config.Constants.numberOfStoredBytes - 1] = 1;
		}
		a = first;
		if(a != null)
```

```
                a.pred = this;
        b = second;
        if(b != null)
                b.pred = this;

        g = new byte[2][2][Config.Constants.numberOfStoredBytes];
        for(int i=0; i<2; ++i)
                for(int j=0; j<2; ++j)
                        g[i][j]=grableStr();

        value=null;
        pred=null;
}

public Gate(int id, byte[][][] inp, Gate first, Gate second)
{
        this.id = id;
        printed = false;
        c0=c1=null;
        p=false;

        a = first;
        if(a != null)
                a.pred = this;
        b = second;
        if(b != null)
                b.pred = this;

        g = new byte[2][2][Config.Constants.numberOfStoredBytes];
        g[0][0]=inp[0][0].clone();
        g[0][1]=inp[0][1].clone();
        g[1][0]=inp[1][0].clone();
        g[1][1]=inp[1][1].clone();

        value=null;
        pred=null;
}

public byte[] calc() throws Error
{
        Gate tmp = this;
        byte[] tmp1 = null;
        byte[] tmp2 = null;
        while (tmp != null)
        {
                tmp1=null;
                tmp2=null;

                if(tmp.a != null)
                {
                        if(tmp.a.value != null)
                        {
                                tmp1=tmp.a.value;
                        }
                        else
                        {
                                tmp=tmp.a;
                                continue;
```

```
                }
            }

            if(tmp.b != null)
            {
                if(tmp.b.value != null)
                {
                    tmp2=tmp.b.value;
                }
                else
                {
                    tmp=tmp.b;
                    continue;
                }
            }

            if(tmp1 != null)
            {
                int i = 0;
                if(tmp1[tmp1.length- 1] == 1)
                    i = 1;
                int j = 0;
                if(tmp2 != null && tmp2[tmp2.length - 1] == 1)
                {
                    j = 1;
                }
                byte[] key = calcKey(tmp1,tmp2);
                tmp.value = xorBytes(tmp.g[i][j],key);
            }
            else
            {
                tmp.value = tmp.g[0][0];
            }

            if(tmp.value != null)
            {
                tmp=tmp.pred;
            }
        }

        return this.value;
}

private void resetPrinted()
{
        Gate tmp = this;
        while(tmp != null)
        {
            tmp.printed = false;
            if(tmp.a != null && tmp.a.printed)
            {
                tmp = tmp.a;
                continue;
            }
            if(tmp.b != null && tmp.b.printed)
            {
                tmp = tmp.b;
                continue;
            }
            tmp = tmp.pred;
```

```
		}
	}

	public void print() throws IOException
	{
		resetPrinted();
		FileOutputStream out = new FileOutputStream(new File(Config.Constants.serverGateFile-
Name));
		Gate point = this;

		while(point != null)
		{
			//point.printed = false;
			if(point.a != null && !point.a.printed)
			{
				point = point.a;
				continue;
			}
			if(point.b != null && !point.b.printed)
			{
				point = point.b;
				continue;
			}
			StreamMediator.intToFileOutputStream(point.id, out);
			if(point.a != null)
				StreamMediator.intToFileOutputStream(point.a.id, out);
			else
				StreamMediator.intToFileOutputStream(-1, out);
			if(point.b != null)
				StreamMediator.intToFileOutputStream(point.b.id, out);
			else
				StreamMediator.intToFileOutputStream(-1, out);
			for(int i=0; i<2; ++i)
			{
				for(int j=0; j<2; ++j)
				{
					StreamMediator.intToFileOutputStream(point.g[i][j].length, out);
					out.write(point.g[i][j], 0, point.g[i][j].length);
				}
			}
			point.printed = true;
			point = point.pred;
		}
	}

	protected byte[] get0()
	{
		return c0;
	}

	protected byte[] get1()
	{
		return c1;
	}

	protected boolean getP()
	{
		return p;
	}
```

```
private byte[] grableStr()
{
	byte[] tmp = new byte[Config.Constants.numberOfStoredBytes];
	Random rand = new Random();
	for(int i=0; i < Config.Constants.numberOfStoredBytes; ++i)
	{
		for(int j=0; j<8; ++j)
		{
			if(rand.nextBoolean())
			{
				tmp[i] += 1<<j;
			}
		}
	}
	return tmp;
}

private byte[] xorBytes(byte[] a, byte[] b)
{
	if(a == null && b == null)
		return null;
	if(a == null)
		return b;
	if(b == null)
		return a;
	byte[] res = new byte[a.length];
	for(int i=0; i<a.length; ++i)
	{

		res[i] = (byte) (a[i] ^ b[i]);
	}
	return res;
}

private byte[] generateKey(int index) throws Error
{
	byte[] res = null;
	MessageDigest md;
	try {
		md = MessageDigest.getInstance(Config.Constants.hashAlgorithm);
	} catch (NoSuchAlgorithmException e) {
		e.printStackTrace();
		throw new Error();
	}
	byte[] tmp;
	md.reset();

	if(a != null)
	{
		if(index >= 2)
			tmp = a.get1();
		else
			tmp = a.get0();
		md.update(tmp);
		res = md.digest();

		if(b != null)
		{
```

```
                        if( (index & 1) == 1)
                                tmp = b.get1();
                        else
                                tmp = b.get0();
                        md.update(tmp);
                        res = xorBytes(res, md.digest());
                }
        }

        return res;
}

private byte[] calcKey(byte[] a, byte[] b) throws Error
{
        byte[] res = null;
        MessageDigest md;
        try {
                md = MessageDigest.getInstance(Config.Constants.hashAlgorithm);
        } catch (NoSuchAlgorithmException e) {
                e.printStackTrace();
                throw new Error();
        }
        md.reset();

        if(a != null)
        {
                md.update(a);
                res = md.digest();

                if(b != null)
                {
                        md.update(b);
                        res = xorBytes(res, md.digest());
                }
        }

        return res;
}

protected void encrypt() throws Error
{
        for(int i=0; i<2; ++i)
                for(int j=0; j<2; ++j)
                {
                        byte[] key;
                        key = generateKey(i*2+j);
                        if(key == null)
                                continue;
                        g[i][j] = xorBytes(g[i][j], key);
                        //keys[i][j] = key;
                }

        if(a != null && a.getP())
        {
                byte[] tmp = g[0][0];
                g[0][0] = g[1][0];
                g[1][0] = tmp;
```

```
                tmp = g[0][1];
                g[0][1] = g[1][1];
                g[1][1] = tmp;
            }

            if(b != null && b.getP())
            {
                byte[] tmp = g[0][1];
                g[0][1] = g[0][0];
                g[0][0] = tmp;

                tmp = g[1][1];
                g[1][1] = g[1][0];
                g[1][0] = tmp;
            }
        }

        public byte[][] getOutput()
        {
            byte[][] code = new byte[2][Config.Constants.numberOfStoredBytes];
            code[0]=get0();
            code[1]=get1();
            return code;
        }
}
```

П.3.19. ФАЙЛ «Gate/GateAnd.java»

```
package Gate;
public class GateAnd extends Gate {
        public GateAnd(Gate first, Gate second) throws Error
        {
            super(first, second);
            g[0][0]=this.get0();
            g[0][1]=this.get0();
            g[1][0]=this.get0();
            g[1][1]=this.get1();

            encrypt();
        }
}
```

П.3.20. ФАЙЛ «Gate/GateInput.java»

```
package Gate;

public class GateInput extends Gate {

        public GateInput()
        {
            super(null,null);
            for(int i=0; i<2; ++i)
                for(int j=0; j<2; ++j)
                    g[i][j]=this.get0();
        }
```

```
	public void set(boolean pos, byte[] value)
	{
		if(pos)
		{
			g[1][0] = value;
			g[1][1] = value;
		}
		else
		{
			g[0][0] = value;
			g[0][1] = value;
		}
	}
}
```

П.3.21. ФАЙЛ «Gate/GateNop.java»

```
package Gate;
public class GateNop extends Gate {
	public GateNop(Gate first) throws Error
	{
		super(first, null);
		g[0][0]=this.get0();
		g[0][1]=this.get0();
		g[1][0]=this.get1();
		g[1][1]=this.get1();

		encrypt();
	}
}
```

П.3.22. ФАЙЛ «Gate/GateNot.java»

```
package Gate;
public class GateNot extends Gate {
	public GateNot(Gate first) throws Error
	{
		super(first, null);
		g[0][0]=this.get1();
		g[0][1]=this.get1();
		g[1][0]=this.get0();
		g[1][1]=this.get0();

		encrypt();
	}
}
```

П.3.23. ФАЙЛ «Gate/GateOr.java»

```
package Gate;
public class GateOr extends Gate {
	public GateOr(Gate first, Gate second) throws Error
	{
		super(first, second);
		g[0][0]=this.get0();
		g[0][1]=this.get1();
```

```
                g[1][0]=this.get1();
                g[1][1]=this.get1();

                encrypt();
        }
}
```

П.3.24. ФАЙЛ «Gate/GateConstructor.java»

```
package Gate;

public final class GateConstructor {
        private Gate func;
        private GateInput[] inp;
   private byte[][] res;

        public GateConstructor(byte[][] p) throws Error
        {
                Gate first = null;
                Gate second = null;
                inp = new GateInput[Config.Constants.numberOfInputGates];
                for(int i=0; i<Config.Constants.numberOfInputGates; ++i)
                {
                        inp[i] = new GateInput();
                }
                if(p.length > 0)
                        first = buildElement(p[0]);
                for(int i=1; i<p.length; ++i)
                {
                        second = buildElement(p[i]);
                        Gate tmp = new GateOr(first,second);
                        first = tmp;
                }
                func = first;
                res=func.getOutput();
        }

        private Gate buildElement(byte[] num) throws Error
        {
                Gate first = null;
                Gate second = null;
                Gate tmp = null;
                if((num[0] & 1) == 0)
                {
                        tmp = new GateNot(getInputGate(0));
                }
                else
                {
                        tmp = new GateNop(getInputGate(0));
                }
                for(int j=0; j<num.length; ++j)
                {
                        for(int i=0; i<8; ++i)
                        {
                                if( i == 0 && j == 0)
                                        continue;
                                if((num[j] & (1<<i)) == 0)
                                {
                                        second = new GateNot(getInputGate(j*8+i));
```

```
                                        }
                                        else
                                        {
                                                second = new GateNop(getInputGate(j*8+i));
                                        }
                                        first = tmp;
                                        tmp = new GateAnd(first, second);
                                }
                        }
                        return tmp;
                }

                public Gate getGate()
                {
                        return func;
                }

                public GateInput getInputGate(int num)
                {
                        if(num >= 0 && num < inp.length)
                                return inp[num];
                        return null;
                }

                public byte[][][] collectOTinput()
                {
                        byte[][][] res = new byte[Config.Constants.numberOfBits][2][Config.Constants.numberOfStored-
Bytes];
                        //bitIndex of table g(numberOfBits), bitState of table g(0,1), length of bitValue(numberOfStored-
Bytes)
                        for(int i=0; i<Config.Constants.numberOfInputGates; ++i)
                                res[i] = getInputGate(i).getOutput();
                        return res;
                }

                public byte[][] getEtalon()
                {
                        return res;
                }
}
```