

Иван Зубофф

Разработка программы обмена мгновенными сообщениями

Часть бакалаврской работы, защищённой в Московском
энергетическом институте в 2015 году

Содержание

Обозначения и сокращения	3
Введение	4
1. Предварительный анализ поставленной задачи	7
2. Разработка алгоритмов	12
2.1. Организация сетевого взаимодействия	12
2.2. Организация криптографической схемы	17
2.3. Алгоритм криптографической защиты соединения	23
3. Программная реализация	29
3.1. Разработка графического интерфейса пользователя	29
3.2. Детали реализации	32
3.3. Работа с программой	34
3.4. Эксперименты	35
Заключение	40
Список использованных источников	43
Приложение А. Схема алгоритма программного экземпляра	46
Приложение Б. Листинг программы	48

Обозначения и сокращения

ЭП	Электронная подпись
Криптографическая хеш-функция	Функция, ставящая по определённому алгоритму в соответствие массиву данных произвольной длины массив данных фиксированной длины и обладающая свойствами необратимости, стойкости к коллизиям первого и второго рода [38, 39]
Хеш	Результат выполнения хеш-функции, на вход которой были поданы определённые данные
Нонс	Дополнительные данные для шифрования и расшифрования, предназначенные для защиты от атаки повторного воспроизведения [34, 40]
Лицензия <i>BSD</i> из 2-х пунктов	Лицензия программного обеспечения, содержащая отказ от гарантий и разрешение на использование, изменение и распространение исходного кода или двоичного кода программы с незначительными ограничениями [30, 41]

Введение

В широком смысле разработка выполняется в трёх областях: сетевого программирования, прикладного использования современной криптографии в программном обеспечении, построения графического интерфейса пользователя.

В узком смысле разработка выполняется в области клиент-серверного программирования между одним клиентом и одним сервером по протоколу *TCP*. Поскольку разработка ведётся под операционную систему *Ubuntu*, являющуюся *UNIX*-подобной, сетевое программирование будет реализоваться с использованием стандартных для всех *UNIX*-подобных систем средств, а именно с помощью сокетов Беркли [37]. Сокеты Беркли – это интерфейс программирования приложений для организации сетевого взаимодействия. Использование криптографических средств защиты информации заключается в использовании функций из сторонней криптографической библиотеки для осуществления асимметричного шифрования сообщений пользователей, подтверждения целостности и подлинности сообщений с помощью механизма электронной подписи, подтверждения соответствия открытых ключей шифрования и электронной подписи собеседников с помощью криптографической хеш-функции. Графический интерфейс пользователя будет разрабатываться для удобной работы с программой самых различных пользователей, в том числе не являющихся специалистами по вычислительной технике. Он должен основываться на применении кроссплатформенной библиотеки элементов интерфейса для обеспечения высокой мобильности по отношению к программным платформам.

В результате разработки будет получена программа для операционной системы *Ubuntu*, позволяющая двум пользователям обмениваться между собой текстовыми сообщениями по локальной или глобальной сети. При использовании один (любой) экземпляр программы подключается в качестве клиента к другому экземпляру (серверу), и с помощью установленного соединения пользователи

осуществляют двухсторонний обмен мгновенными сообщениями. Сообщения пользователей будут защищены криптографической схемой на основе вызова функций из сторонней криптографической библиотеки. Криптографическая схема будет включать в себя электронную подпись для аутентификации сообщений, шифрование с открытым ключом, а также подтверждение соответствия открытых ключей шифрования и электронной подписи собеседников с помощью криптографической хеш-функции.

Разработка ведётся для операционной системы *Ubuntu 14.04 LTS*, аппаратная платформа разработки — *x86*. За счёт использования функций стандартной библиотеки языка программирования *C* и высокой стандартизации различных дистрибутивов операционной системы *GNU/Linux*, данная разработка должна правильно работать также и на современных версиях других дистрибутивов этой операционной системы. Интерфейс сокетов Беркли является частью группы стандартов *POSIX* [42], поэтому программа должна работать на всех современных *UNIX*-подобных системах с минимальными изменениями. При использовании библиотеки *Winsock* [9] и незначительной правке исходного кода данная разработка должна правильно работать также и на операционных системах семейства *Windows*.

При переносе на другие аппаратные платформы, например *IA-64* или *ARM*, могут возникнуть небольшие проблемы с типами данных, но они не потребуют большого количества правок исходного кода. Таким образом, разработка будет очень мобильной по отношению к программным и аппаратным платформам.

Данная разработка может быть применена пользователями глобальной или локальной сети для личной и деловой переписки, а также разработчиками программного обеспечения в качестве основы для их проектов в областях сетевого программирования, применения криптографических средств защиты информации, построения графического интерфейса пользователя, что позволит им ускорить процесс разработки своих продуктов.

Из-за вскрывшихся в последние годы фактов бесконтрольного доступа спецслужб к конфиденциальной информации как при её перемещении по сети, так и при её хранении на серверах различных компаний, в мире значительно усилился интерес к использованию средств криптографической защиты информации. Учитывая огромный интерес к таким программам обмена мгновенными зашифрованными сообщениями, как *Telegram* [5] (число активных пользователей этого приложения составляло 50 миллионов по состоянию на декабрь 2014 года [6]) и *Cryptocat* [7], можно с уверенностью сказать: значительное число пользователей заинтересовано в том, чтобы иметь возможность приватно обмениваться сообщениями через Интернет.

Разрабатываемая в данной работе программа во многом аналогична *Telegram* и *Cryptocat*, так как предоставляет ту же возможность обмена по сети мгновенными сообщениями, защищёнными современными криптографическими средствами. Основными достоинствами данной разработки по сравнению с указанными выше аналогами является её простота и мобильность, позволяющая на её основе реализовать другие проекты, обеспечивающие обмен информацией по сети с использованием достижений прикладной криптографии — *Telegram* и *Cryptocat* заметно хуже подходят в качестве основы для новых разработок. Данный проект может также использоваться в качестве основы для построения графического интерфейса пользователя в стороннем проекте.

Данная работа будет содержать разделы, посвящённые предварительному анализу поставленной задачи, разработке алгоритмов, а также программной реализации.

1. ПРЕДВАРИТЕЛЬНЫЙ АНАЛИЗ ПОСТАВЛЕННОЙ ЗАДАЧИ

В широком смысле разработка выполняется в трёх областях: сетевого программирования, прикладного использования современной криптографии в программном обеспечении и построения графического интерфейса пользователя. Разрабатываемая программа, необходимая для защищённого обмена мгновенными сообщениями по локальной или глобальной сети, по очевидным причинам находится на стыке этих областей.

Среди схожих разработок в первую очередь стоит выделить программы *Telegram* и *Cryptocat*. Это программы ставят своей целью создание удобного и простого в использовании сервиса обмена мгновенными сообщениями с использованием средств криптографической защиты информации, как и проект, разрабатываемый в данной бакалаврской работе. Обе программы, названные выше, бесплатны и имеют открытый исходный код, выпущенный под лицензиями семейства *GNU GPL* (у проекта *Telegram* опубликована только клиентская часть), то есть являются доступными для изучения и изменения всеми желающими. На данный момент и *Telegram*, и *Cryptocat* предлагают своим пользователям удобный интерфейс, стойкую криптографическую защиту и мультиплатформенность, благодаря чему пользуются заслуженной популярностью у пользователей.

Клиентская сторона проекта *Telegram* реализована в виде приложений для целого ряда платформ: *Android*, *iOS*, *Windows Phone*, *Blackberry*, *Windows*, *Mac OS X*, *GNU/Linux*, а также в виде веб-приложения и расширения для браузера *Google Chrome*. Единого языка программирования у всех этих разработок нет, поскольку для каждой конкретной платформы авторы стараются разработать приложение с использованием наиболее подходящих для данной платформы средств.

Telegram использует собственный протокол *MTPROTO*, основанный на хорошо известных и стойких криптографических алгоритмах. При включении максимальных настроек приватности есть возможность отказаться от хранения

передаваемых сообщений в облаке *Telegram* и передавать их напрямую между устройствами собеседников. Возможность создать своё облако *Telegram* для работы через него, например, в корпоративной сети на данный момент отсутствует. Архитектура приложения — клиент-серверная: 2 или более людей подключаются к доверенному серверу, обмениваются через него ключами шифрования, после чего отправляют серверу зашифрованные сообщения, который тот перенаправляет нужным адресатам. Так как ключи генерируются на стороне клиентов, то в теории сообщения не могут быть прочитаны сервером или промежуточными узлами сети, если ключи не были подменены при обмене [14, 15].

Клиентская сторона проекта *Cryptocat* задумывалась как расширение для браузеров, и сейчас это расширение доступно для браузеров *Google Chrome*, *Mozilla Firefox*, *Opera* и *Apple Safari*; также разработаны приложения для *iOS* и *Mac OS X*. Расширения для браузеров написаны на языке программирования *Javascript*, приложения для *iOS* и *Mac OS X* — на языке программирования *Objective-C*.

Cryptocat использует надёжные криптографические протоколы *OTR* и *mpOTR*, работающие поверх *XMPP*-сервера, который в свою очередь защищён семейством криптографических протоколов представительского уровня *SSL/TLS*. Архитектура приложения — клиент-серверная: 2 или более людей подключаются к доверенному серверу, обмениваются через него ключами шифрования, после чего отправляют серверу зашифрованные сообщения, который тот перенаправляет нужным адресатам. Так как ключи генерируются на стороне клиентов, то в теории сообщения не могут быть прочитаны сервером или промежуточными узлами сети, если ключи не были подменены при обмене [11, 12]. Существуют сервера, предоставленные разработчиками *Cryptocat*, а также есть возможность запустить свой сервер по инструкциям от разработчиков [13], что является относительно несложной задачей для системного администратора.

Разрабатываемая в данной работе программа во многом аналогична

Telegram и *Cryptocat*, так как предоставляет ту же возможность обмена по сети мгновенными сообщениями, защищёнными современными криптографическими средствами. Основными достоинствами данной разработки по сравнению с указанными выше аналогами является её простота и мобильность, позволяющие на её основе реализовать другие проекты, обеспечивающие обмен информацией по сети с использованием достижений прикладной криптографии. Данный проект может также использоваться в качестве основы для построения графического интерфейса пользователя в стороннем проекте. *Telegram* и *Cryptocat* заметно хуже подходят в качестве основы для новых разработок (особенно это касается *Telegram*, не предоставляющего исходные коды серверной части).

В качестве платформы выбрана операционная система *GNU/Linux* как широко распространённая, активно развивающаяся и очень перспективная последовательница операционной системы *UNIX*. В качестве более конкретной платформы был выбран дистрибутив *Ubuntu* [21] — один из самых распространённых дистрибутивов *Linux* с хорошей поддержкой русского языка и регулярными обновлениями [22].

Разработка будет состоять из нескольких не слишком больших по объёму модулей, а основным объектом операций в ней будут строки, поэтому рационально будет применить процедурный компилируемый язык программирования *C* и парадигму структурного программирования. Программирование на данном языке обеспечивает высокую производительность и хорошую читабельность кода, широкий инструментарий для разработки и отладки программы. Стандартным и наиболее популярным компилятором под выбранную платформу является *GCC* [23], поэтому именно он выбран для компиляции программы, разрабатываемой в данной бакалаврской работе.

Поскольку разработка ведётся под операционную систему *Ubuntu*, являющуюся *UNIX*-подобной, сетевое программирование рационально будет реализовать с использованием стандартных для всех *UNIX*-подобных систем

средств, а именно с помощью сокетов Беркли. Этот интерфейс программирования приложений является частью группы стандартов *POSIX*, а значит, обеспечит высокую мобильность разрабатываемой программы — она должна будет правильно работать не только на операционной системе *Ubuntu* и не только на других современных дистрибутивах операционной системы *GNU/Linux*, но также и на всех современных *UNIX*-подобных операционных системах. При использовании библиотеки *Winsock* данную разработку можно будет портировать и на *Windows*. Большим плюсом использования этой технологии является также её широкая освещённость в статьях и литературе, в том числе на русском языке, поэтому проблем с недостатком справочного материала не возникнет.

Современная криптография является сложной математической наукой, а программная реализация её разработок является вдвойне непростой задачей: нужно реализовать в максимально безопасном и быстром коде довольно сложные алгоритмы, хорошо понимая суть их работы [16, 17, 18]. Кодирование криптографических алгоритмов на достойном уровне является достаточно подходящей по сложности задачей для бакалаврской работы, даже не говоря о каком-то сетевом программировании и построении графического интерфейса пользователя помимо этого, поэтому в процессе проектирования было принято решение воспользоваться сторонней криптографической библиотекой. Во-первых, это позволит сосредоточиться на прикладных задачах реализации криптографической схемы более высокого уровня, сетевого программирования и построения графического интерфейса пользователя вместо повторного кодирования известных алгоритмов, а во-вторых, позволит избежать множества ошибок, сводящих на нет все попытки обеспечить конфиденциальность, целостность, аутентификацию и неотказуемость передаваемой по сети информации [18, 28].

Важным свойством любой программной разработки является удобство её использования. Это свойство просто критично для программ общего пользования, не ориентированных на профессиональных специалистов по вычислительной

технике. По этой причине было решено разработать для данной программы графический интерфейс пользователя. В качестве библиотеки элементов интерфейса была выбрана широко распространённая и кроссплатформенная библиотека *GTK+* [24], использование которой обеспечит высокую мобильность по отношению к программным платформам. Данная библиотека хорошо документирована, поэтому проблем с недостатком справочного материала не возникнет. Графический интерфейс пользователя будет состоять из нескольких десятков объектов *GTK+* — окон, кнопок, полей ввода и вывода информации. Программное создание такого количества элементов интерфейса с заданием их свойств будет неоправданно громоздким и неудобным, так что было решено использовать *Glade* [35], редактор графического интерфейса для *GTK+*. Этот редактор позволяет в режиме конструктора создавать окна и сохранять их как *XML*-документы [43], в самом же коде прикладной программы достаточно реализовать лишь импорт окон и логику их функционирования.

Поскольку программа претендует на обеспечение защищённого обмена сообщениями между пользователями, при разработке необходимо постараться избегать внесения в код различных уязвимостей. Становится понятно, почему это особенно важно, если вспомнить: при работе программы устанавливается соединение, по которому в программу может быть отправлено любое содержимое. Это означает, что в некоторых участках программы гипотетически возможны, например, удалённые переполнения буфера или кучи, компрометирующие безопасность всего компьютера пользователя, а не только его переписки. С целью избежания использования небезопасных функций при программировании в данной работе используется заголовочный файл *GCC Poison* [25], блокирующий компиляцию уязвимого кода.

2. РАЗРАБОТКА АЛГОРИТМОВ

2.1. Организация сетевого взаимодействия

Особенностью сокетов Беркли, применяемых в данной работе, является их ориентированность на клиент-серверную архитектуру при установлении соединения. Изначальная архитектура разрабатываемой системы, предполагавшая общение нескольких клиентов через сервер, была отброшена в пользу более простой в использовании архитектуры, в которой клиент и сервер являются собеседниками. Такая архитектура позволяет очень просто связаться двум пользователям по локальной и глобальной сети: нет нужды отдельно запускать отдельную программу-сервер, а затем подключаться к нему с двух сторон клиентами, достаточно запустить сервер на первой стороне, а со второй стороны подключиться к этому серверу клиентом. В будущем можно внедрить и возможность соединения по изначальному проекту архитектуры, предоставив доверенный центральный сервер с возможностью его смены на любой другой, как это сделано в *Cryptocat*. Такой подход является исключительно удобным для многих пользователей, поскольку не во всех случаях можно открыть слушающий порт для ожидания подключения клиента. К тому же, в этом случае для соединения можно знать лишь логин, а не *IP*-адрес собеседника.

Большинство сетевых взаимодействий сегодня опираются на протокол *IPv4* сетевого уровня эталонной модели *OSI*, поэтому реализация работы программы с опорой на него была первоочередной задачей при разработке. Однако уже в конце восьмидесятых годов прошлого века возникла необходимость в более совершенном протоколе сетевого уровня, и им стал *IPv6*, утверждённый в качестве международного стандарта в 1996 году. Для того, чтобы разрабатываемая в данной бакалаврской работе программа не потеряла своей актуальности в ближайшей перспективе, в неё была внедрена поддержка и этого протокола

сетевого уровня. Переключение между режимами работы по двум этим протоколам осуществляется в окне настроек соединения.

Скорость обмена сообщениями между пользователями не является сверхкритичной — даже если каждое сообщение будет идти по сети одну секунду, пользователи могут даже не заметить этого. В отличие от скорости, для данной разработки очень важна целостность сообщений и обеспечение надёжного канала связи между пользователями, поэтому при выборе между протоколами транспортного уровня *TCP* и *UDP* решено было остановиться на *TCP*. Конечно, проверку на ошибки и проверку доставки можно реализовать в своём прикладном коде, опирающемся на *UDP*, но в данном случае это было бы неэффективным решением поставленной задачи, так как *TCP* представляет отлаженный и эффективный функционал для тех же целей.

Указание номера *TCP*-порта сервера как для клиента, так и для сервера является опциональным, по умолчанию используется не закреплённый ни за одним приложением на июнь 2015 года согласно списку *IANA* [26] *TCP*-порт 37000. *IANA* (*Internet Assigned Numbers Authority*) — некоммерческая организация, занимающаяся в том числе ведением списка официально назначенных отдельным приложениям портов. Порт 37000 попадает в диапазон «1024 — 49151», порты из которого регистрируются за определёнными приложениями, но могут с лёгкостью использоваться и другими приложениями («зарегистрированные» или «пользовательские» порты). Порты в диапазоне «0 — 1023» («общеизвестные» или «системные») могут использоваться только при наличии привелегий суперпользователя; порты в диапазоне «49152 — 65535» («динамические») обычно не открываются напрямую приложениями, а открываются по инициативе операционной системы. Именно поэтому от назначения порта сервера по умолчанию из этих двух диапазонов было решено отказаться (тем не менее, в окне настроек соединения можно указать любой порт в диапазоне «1024 — 65535»).

Номер *TCP*-порта клиента не имеет какого-либо значения, поэтому его автоматически выбирает ОС в процессе работы приложения, что обеспечивает

удобство использования для обеих сторон соединения.

При разработке этой части бакалаврской работы активно использовалась литература по теме: [1, 2, 3, 4].

На рис. 1 приведена схема алгоритма работы сервера, на рис. 2 – схема алгоритма работы клиента. Ниже приведены упрощённые последовательности системных вызовов для организации сетевого взаимодействия. Используемые переменные: *servinfo* — информация о сервере, *listensock* — слушающий сокет на сервере, ожидающий подключения клиента, *sock* — сокет для общения клиента и сервера.

Сервер:

1. *servinfo = getaddrinfo()* — получение информации о текущем хосте, так как она будет нужна в дальнейшем.

2. *listensock = socket(servinfo)* — открытие сокета, который будет ожидать подключения клиента.

3. *setsockopt(listensock)* — разрешение повторного использования порта, на котором сервер будет ожидать подключения (добавлено для удобства использования).

4. *bind(listensock, servinfo)* — привязка слушающего сокета к желаемому порту.

5. *freeaddrinfo(servinfo)* — очистка информации о текущем хосте, так как она больше не нужна для работы программы.

6. *listen(listensock)* — ожидание подключения клиента.

7. *sock = accept(listensock)* — принятие подключения клиента с получением нового сокета для общения с ним.

8. *close(listensock)* — так как мы ожидали только одно подключение на открытый ранее порт, то после успешного установления соединения с клиентом

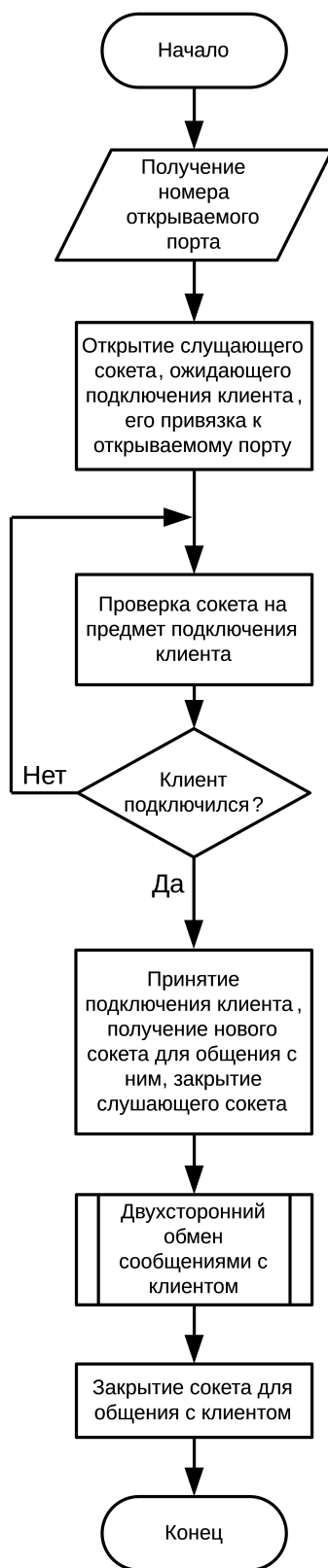


Рис. 1. Схема алгоритма работы сервера

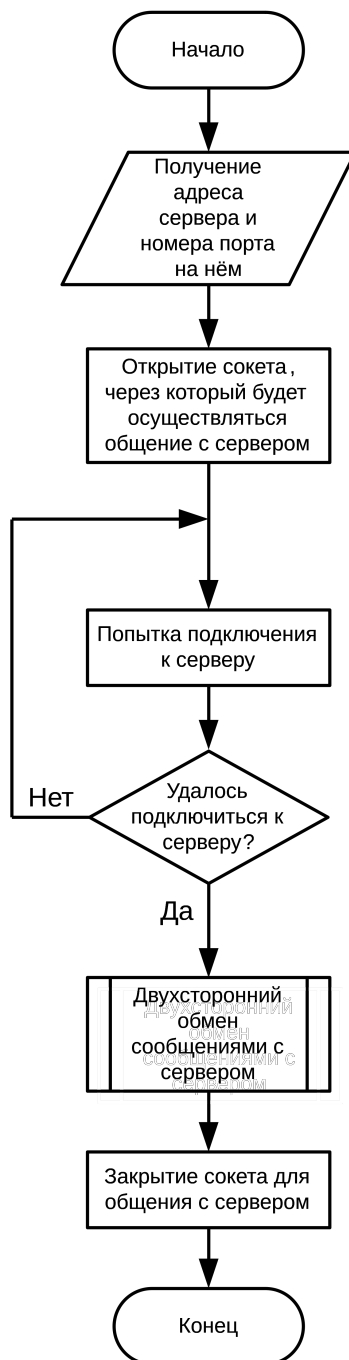


Рис. 2. Схема алгоритма работы клиента

первый сокет необходимо закрыть.

9. *send(sock) + recieve(sock)* — обмен сообщениями с клиентом.

10. *close(sock)* — закрытие сокета для общения с клиентом после завершения общения.

Клиент:

1. *servinfo = getaddrinfo()* — получение информации о сервере, так как она будет нужна в дальнейшем.

2. *sock = socket(servinfo)* — открытие сокета, через который будет осуществляться общение с сервером.

3. *connect(sock, servinfo)* — подключение к серверу.

4. *freeaddrinfo(servinfo)* — очистка информации о сервере, так как она больше не нужна для работы программы.

5. *send(sock) + recieve(sock)* — обмен сообщениями с сервером.

6. *close(sock)* — закрытие сокета для общения с сервером после завершения общения.

2.2. Организация криптографической схемы

Криптография — это наука о методах обеспечения конфиденциальности (невозможности прочтения информации посторонним), целостности (невозможности незаметного изменения информации), аутентификации (проверки подлинности авторства или иных свойств объекта) и неотказуемости (невозможности отказа от авторства) данных [28]. Все эти возможности, безусловно, являются крайне желательными и ценными при разработке приложения для обмена мгновенными сообщениями, поэтому в данную бакалаврскую работу было решено внедрить средства криптографической защиты информации, обеспечивающие все 4 её свойства, перечисленные выше. По результатам предварительного анализа задания было решено использовать

функции из сторонней криптографической библиотеки.

Среди различных криптографических библиотек выбрана библиотека *TweetNaCl* [8], основными причинами выбора которой стали безопасность и надёжность, портбельность, лёгкость в изучении и использовании, малый размер в исходном коде и в скомпилированном виде, достаточно высокая скорость работы, а также полнота предоставляемых ею функций. Библиотека написана на языке C, что делает её использование в разрабатываемой программе простым и удобным. Эта написанная специалистами-криптографами миниатюрная библиотека предоставляет прикладному программисту 25 криптографически стойких функций, позволяющих реализовать симметричное и асимметричное шифрование, электронную подпись, хеширование и некоторые вспомогательные процедуры [16, 18]. Она легка в изучении и использовании, что выгодно отличает её от широко используемой криптографической библиотеки *OpenSSL* [10]. Последняя предлагает программисту набор из более чем тысячи функций [20]. Проект *TweetNaCl* является ветвью развития другого криптографического проекта *NaCl* [29] тех же авторов, поэтому значительная часть документации к *NaCl* цитируется в данной бакалаврской работе.

Для обеспечения конфиденциальности, целостности и аутентификации используется функция *crypto_box_curve25519xsalsa20poly1305*, обеспечивающая аутентифицированное асимметричное шифрование данных путём использования эллиптической кривой *Curve25519*, поточного шифра *XSalsa20* и имитовставки *Poly1305*. Для обеспечения целостности, аутентификации и неотказуемости используется функция *crypto_sign_ed25519*, обеспечивающая электронную подпись данных путём использования системы электронной подписи на основе открытых ключей *Ed25519*. Для проверки соответствия публичных ключей собеседников используется функция *crypto_hash_sha512*, обеспечивающая хеширование данных путём использования криптографической хеш-функции *SHA-512* семейства *SHA-2* [16].

Читателю может показаться, что использование функций из сторонней

криптографической библиотеки решает все проблемы защиты информации без каких-либо усилий со стороны программиста, однако далее будет показано, что это далеко не так. Дело в том, что криптографические функции нужно ещё правильно и аккуратно применить в прикладной программе, иначе вся предоставляемая ими защита легко обходится.

В ходе выполнения данной бакалаврской работы был разработан криптографический протокол, обеспечивающий защиту от нескольких типов атак. Сначала пользователи генерируют две пары постоянных (долговременных) ключей: публичный и секретный ключи электронной подписи, публичный и секретный ключи шифрования, после чего обмениваются постоянными публичными ключами между собой. Обмен может происходить напрямую через сеть по незащищённому каналу, но существует также возможность записать сгенерированные постоянные секретные и публичные ключи на диск, обмениваться постоянными публичными ключами с собеседником по защищённому каналу (например, с помощью обмена USB-флеш-накопителями при личной встрече), после чего загрузить все необходимые ключи с диска. После обмена постоянными ключами по сети либо загрузки их с диска собеседники могут сверить хеши постоянных публичных ключей (своих и собеседника) для того, чтобы удостовериться в их соответствии, что особенно важно при обмене постоянными ключами по сети.

Возможность передачи постоянных публичных ключей по защищённому каналу, а также возможность сверки хешей постоянных публичных ключей собеседниками необходимы в силу незащищённости асимметричной криптографии от атаки типа «человек посередине» [31]. Допустим, хакер Меллори имеет полный доступ к коммутатору на пути передачи данных между Бобом и Алисой. Полный доступ означает, что Меллори помимо прослушивания канала (которое стойкая криптография делает практически бесполезным) может модифицировать и/или подделывать пакеты между Бобом и Алисой. В этом случае

она может осуществить названную выше атаку, представившись Бобу Алисой, а Алисе — Бобом сначала при обмене ключами (она выдаёт им свои ключи вместо ключей собеседника), а затем и при последующем обмене сообщениями. Теперь при отправке сообщения кем-либо из сторон она расшифровывает его имеющимися у неё ключами, шифрует другими ключами и передаёт адресату. Помимо чтения сообщений она может также без проблем изменять, удалять любые сообщения или даже подделывать сообщения целиком. Таким образом, при использовании полностью правильно работающих и стойких криптографических функций без применения более сложных методов защиты (например, описанных и реализованных в данной работе) вся забота о приватности практически обесценивается. Другим способом борьбы с этой атакой является построение сети доверия наподобие реализованной в программе *PGP* [32], однако это является уже куда более сложной задачей, поэтому было решено остановиться на двух более простых и столь же безопасных решениях, несколько проигрывающих в удобстве использования.

Итак, если при сверке хеши постоянных публичных ключей на обеих сторонах окажутся идентичными, то вероятность атаки «человек посередине» ничтожно мала; если хеши различны, а передаваемые между собеседниками данные не проходят проверок на противоположной стороне, то произошла какая-то ошибка при работе с ключами; если хеши различны, а сообщения собеседников успешно проходят проверки на противоположной стороне, то происходит атака типа «человек посередине». Хеши выводятся в шестнадцатеричном коде, в удобном для человека виде, что позволяет сверить их как визуально (например, передав значение хеша через доверенный ресурс), так и аудиально (например, продиктовав значение хеша по телефону). В данной бакалаврской работе используется хеш-значение от буфера, в котором сначала записан бóльший публичный ключ электронной подписи из своего и собеседника, затем меньший, после этого бóльший публичный ключ шифрования из своего и собеседника, затем меньший, за счёт чего хеши на обеих сторонах совпадают. Такой подход не

зависит от сетевой роли «клиент» или «сервер», поэтому он будет работать и при переходе к архитектуре с центральным сервером, передающим данные между всеми клиентами.

При разработке криптографической схемы в данной бакалаврской работе было решено, что не стоит полагаться на один и тот же набор постоянных ключей — гораздо лучшим решением будет использование упомянутых ранее постоянных ключей при установлении соединения, а после этого сразу же обмениваться сеансовыми (временными) ключами. За счёт использования ранее согласованных неподменённых постоянных ключей Мэлори не сможет ни прочесть, ни подменить сеансовые публичные ключи, а обмен постоянными ключами защищается описанными выше методами: передачей по защищённому каналу и/или сверкой хешей собеседниками. Главный выигрыш от введения сеансовых ключей в том, что мы получаем очень важное и ценное свойство разработанного в данной работе криптографического протокола под названием «совершенная прямая секретность» [33]. Даже если в руках Мэлори окажутся все постоянные ключи обеих сторон и сохранённый ранее зашифрованный трафик, то расшифровать его она не сможет: он шифровался временными сеансовыми ключами, нигде не сохраняющимися после беседы. Ещё одним плюсом является то, что Мэлори не сможет собрать большое количество зашифрованных одними и теми же ключами сообщений (теоретически это может привести к компрометации используемых ключей). Это связано с тем, что постоянными ключами шифруются только сообщения с сеансовыми публичными ключами, то есть лишь 2 сообщения за одну беседу, а сеансовые ключи для каждой беседы будут являться уникальными. На первый взгляд может показаться, что для усиления безопасности стоит пойти ещё дальше и генерировать новые временные ключи не для сеанса связи, а для каждого отдельного сообщения, однако этот подход имеет один существенный минус. Дэниел Юлиус Бернштейн, ведущий разработчик криптографических библиотек *NaCl* и *TweetNaCl*, категорически не согласен с таким подходом по

причине избыточной нагрузки на системный генератор псевдослучайных чисел. Свои аргументы в пользу экономного использования псевдослучайных чисел в криптографии он описал в статье «*Entropy Attacks!*» [19]. При разработке программы в данной бакалаврской работе применялся именно такой подход.

Важно заметить, что после обмена сеансовыми публичными ключами собеседникам может быть полезно сверить хеши от сеансовых публичных ключей. Причина в том, что даже если злоумышленник прочитает или подменит на диске все постоянные ключи обоих собеседников либо их часть, для чтения их переписки ему всё равно придётся совершать атаку «человек посередине». Сверка хешей позволяет легко обнаружить её проведение и разорвать связь.

Для защиты от атак типа «повторное воспроизведение» [34] и от утаивания части сообщений отправителя от получателя применяются особым образом генерируемые нонсы (дополнительные данные для шифрования и расшифрования [40]). Если генерировать нонс для каждого сообщения случайным образом, то помимо избыточной нагрузки на системный генератор псевдослучайных чисел, а также на канал связи между собеседниками (эти нонсы придётся передавать собеседнику для расшифровки сообщения), возникает и более серьёзный недостаток. Меллори может незаметно утаивать от Алисы любые сообщения Боба — даже если до получателя будет доходить, например, каждое второе сообщение отправителя, получатель ничего об этом не узнает. Более того, для Мэлори нет никаких проблем в том, чтобы сохранить нонс и зашифрованное сообщение Боба, после чего без дешифровки послать данный набор в этом сеансе связи Алисе 100 раз, и 100 раз Алиса примет эти дубликаты как правильные сообщения, ведь они правильно зашифрованы и подписаны. Таким образом, у Меллори появляется просто реализуемая возможность манипуляции сообщениями собеседников. При генерации нонса из хеша от предыдущего зашифрованного отправителем сообщения получатель не сможет расшифровать ни один подобный дубликат, ведь при расшифровке он с вероятностью, близкой к единице, будет использовать не

тот нонс, с которым было зашифровано сообщение. Для шифровки своего первого сообщения и его расшифровки на противоположной стороне каждому собеседнику всё-таки придётся сгенерировать и передать один случайный нонс.

Такой подход позволяет избавиться от излишней нагрузки на системный генератор псевдослучайных чисел и на канал связи между собеседниками, избежать успешного проведения атаки повторного воспроизведения и легко обнаруживать попытки её реализации (одному из собеседников приходит нерасшифруемое сообщение якобы от другого, которое тот не отправлял). Кроме того, при блокировании передачи хотя бы одного сообщения одному из собеседников злоумышленником атака также будет обнаружена, так как после этого получатель будет безуспешно пытаться расшифровать все последующие сообщения не с теми нонсами, с которыми они были зашифрованы. Первые N сообщений каждого отправителя по-прежнему можно незаметно утаить от получателя, что планируется исправить в будущих версиях.

2.3. Алгоритм криптографической защиты соединения

Ниже приведён разработанный в данной бакалаврской работе криптографический алгоритм. По этому алгоритму работает каждая сторона соединения, в отличие от алгоритмов организации сетевого взаимодействия. Графически алгоритм изображён в составе схемы алгоритма программного экземпляра на рис. 3. Используемые переменные:

Mt_{sp} , Mt_{ss} — постоянные публичный и секретный ключи электронной подписи данной стороны;

Mt_{cp} , Mt_{cs} — постоянные публичный и секретный ключи шифрования данной стороны;

Mx_{sp} , Mx_{cp} — постоянные публичные ключи электронной подписи и шифрования собеседника;

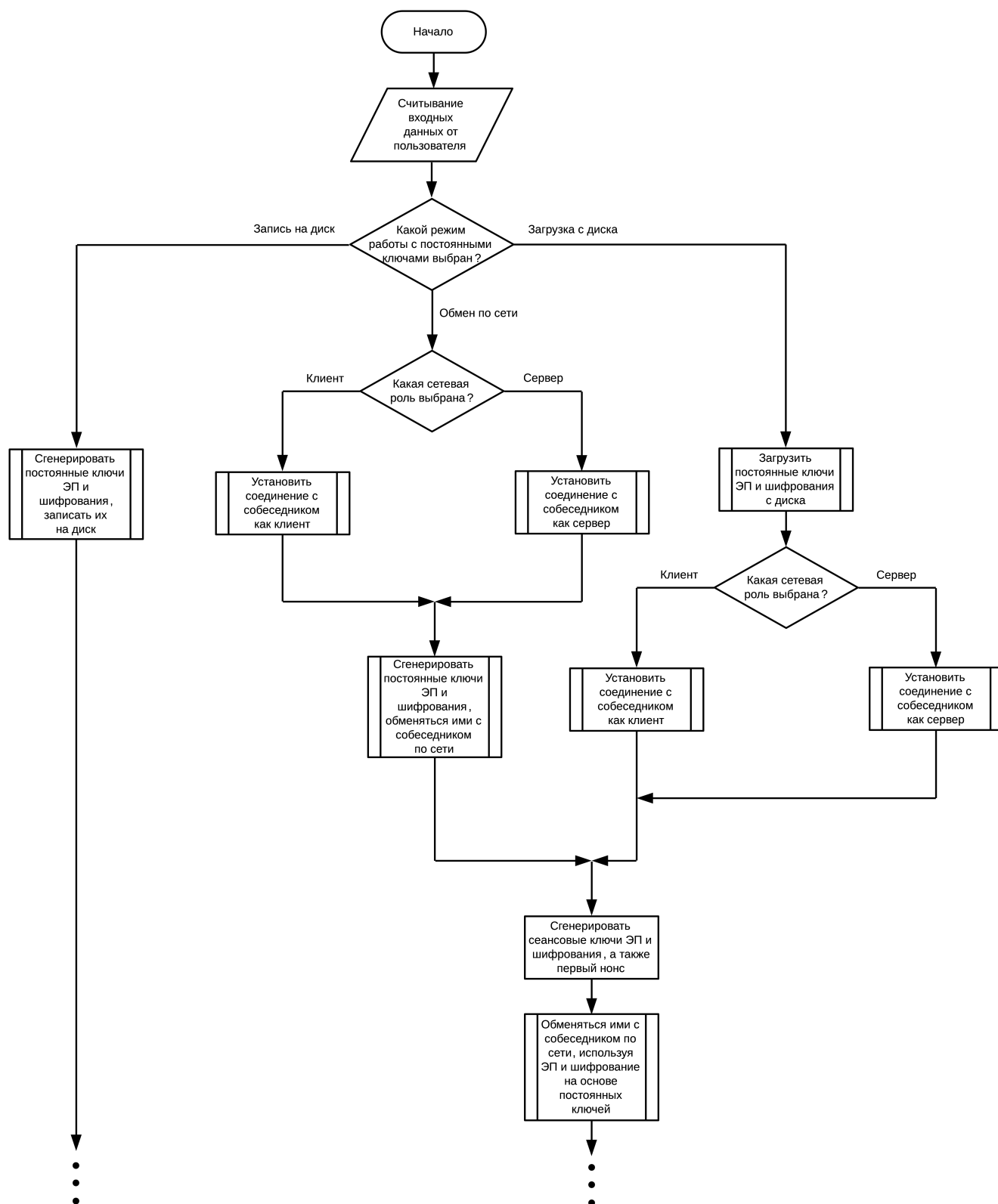


Рис. 3. Схема алгоритма программного экземпляра (начало)

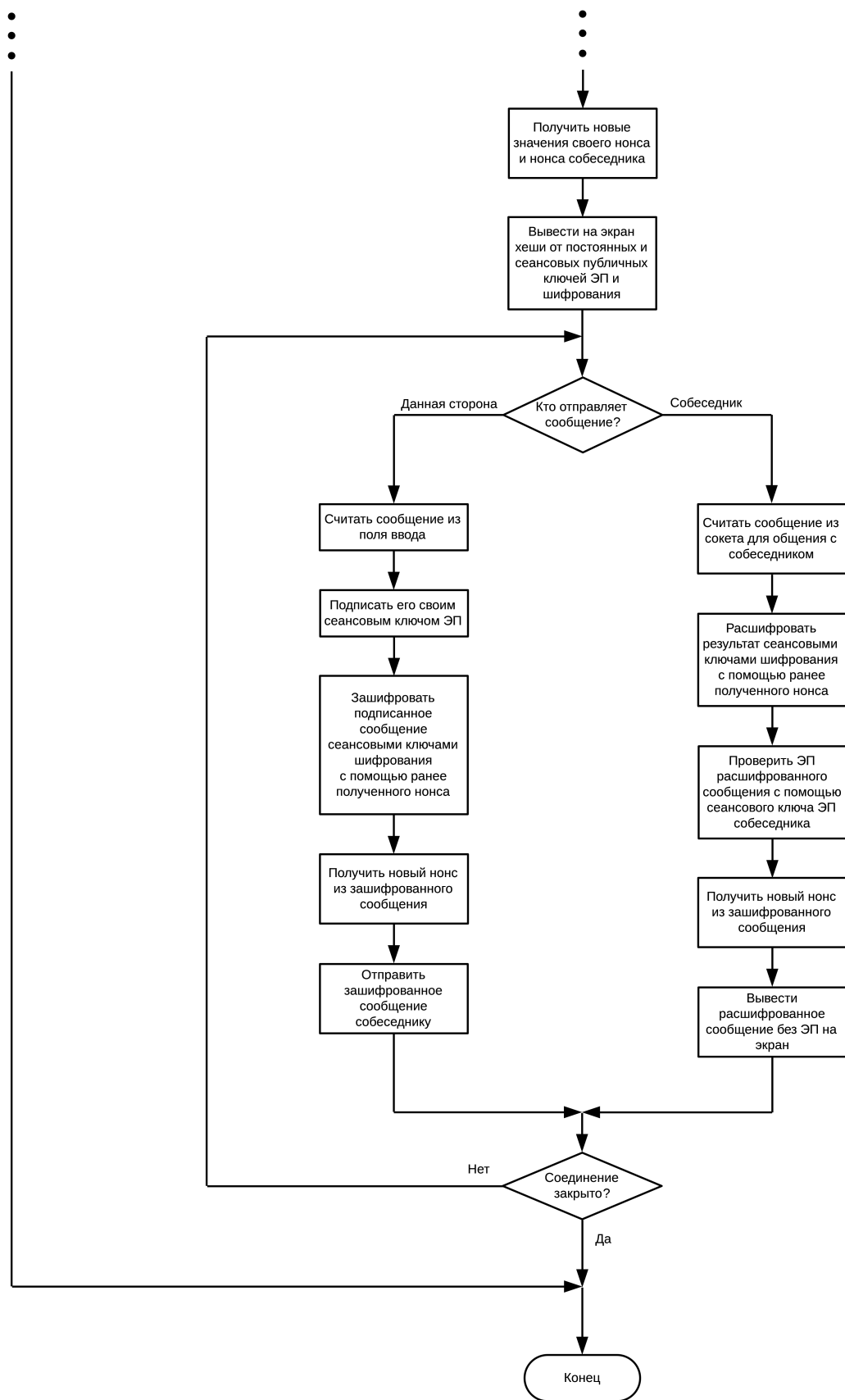


Рис. 3. Схема алгоритма программного экземпляра (окончание)

m_{sp} , m_{ss} — сеансовые публичный и секретный ключи электронной подписи данной стороны;

m_{cp} , m_{cs} — сеансовые публичный и секретный ключи шифрования данной стороны;

x_{sp} , x_{cp} — сеансовые публичные ключи электронной подписи и шифрования собеседника,

m_n , x_n , x_{n_tmp} — нонсы: данной стороны, собеседника, неподтверждённый собеседника.

1. Обмен постоянными ключами

1.1. Генерация постоянных ключей электронной подписи (Mm_{sp} , Mm_{ss}) и шифрования (Mm_{cp} , Mm_{cs}) данной стороны.

1.2. Отправка собеседнику напрямую через данное приложение или через защищённый канал постоянных публичных ключей электронной подписи и шифрования данной стороны (Mm_{sp} , Mm_{cp}).

1.3. Получение от собеседника напрямую через данное приложение или через защищённый канал его постоянных публичных ключей электронной подписи и шифрования (Mx_{sp} , Mx_{cp}).

1.4*. (опционально) Сверка собеседниками по другому каналу (не через данное приложение) хешей от постоянных публичных ключей электронной подписи и шифрования (Mm_{sp} , Mm_{cp} , Mx_{sp} , Mx_{cp}). Если они не совпали, то разрыв соединения, иначе продолжение работы.

2. Обмен сеансовыми ключами

2.1. Генерация сеансовых ключей электронной подписи (m_{sp} , m_{ss}) и шифрования (m_{cp} , m_{cs}) данной стороны, псевдослучайного нонса данной стороны m_n . Формирование сообщения с сеансовыми публичными ключами электронной подписи и шифрования данной стороны (m_{sp} , m_{cp}).

2.2. Подпись этого сообщения постоянным секретным ключом электронной

подписи данной стороны Mm_{ss} . Шифрование получившегося сообщения с использованием нонса данной стороны m_n и постоянных ключей шифрования: секретного данной стороны и открытого собеседника (Mm_{cs} и Mx_{cp}).

2.3. Добавление к результирующему сообщению нонса данной стороны в виде открытого текста. Вычисление хеша от получившегося сообщения и получение из него нового значения нонса данной стороны m_n . Отправка получившегося сообщения собеседнику.

2.4. Получение от собеседника аналогичного сообщения. Вычисление хеша от этого сообщения и получение из него значения неподтверждённого нонса собеседника x_n_{tmp} , считывание из сообщения нонса собеседника x_n .

2.5. Попытка расшифровать зашифрованную часть сообщения собеседника с использованием нонса собеседника x_n и постоянных ключей шифрования: секретного данной стороны и открытого собеседника (Mm_{cs} и Mx_{cp}). Если расшифровка не удалась, то разрыв соединения, иначе продолжение работы.

2.6. Попытка проверить электронную подпись расшифрованного сообщения собеседника с использованием постоянного публичного ключа электронной подписи собеседника Mx_{sp} . Если при этой проверке произошла ошибка, то разрыв соединения, иначе продолжение работы.

2.7. Считывание из расшифрованного сообщения без электронной подписи сеансовых публичных ключей электронной подписи и шифрования (x_{sp} и x_{cp}), присвоение нонсу следующего сообщения собеседника x_n значения ранее посчитанного неподтверждённого нонса собеседника x_n_{tmp} .

2.8*. (опционально) Сверка собеседниками по другому каналу (не через данное приложение) хешей от сеансовых публичных ключей электронной подписи и шифрования (m_{sp} , m_{cp} , x_{sp} , x_{cp}). Если они не совпали, то разрыв соединения, иначе продолжение работы.

3. Отправка сообщения

3.1. Подпись текстового сообщения сеансовым секретным ключом

электронной подписи данной стороны m_{ss} . Шифрование получившегося сообщения с использованием нонса данной стороны m_n и сеансовых ключей шифрования: секретного данной стороны и открытого собеседника (m_{cs} и x_{cp}). Вычисление хеша от результирующего сообщения и получение из него нового значения нонса данной стороны m_n . Отправка получившегося сообщения собеседнику.

4. Приём сообщения

4.1. Вычисление хеша от принятого сообщения и получение из него значения неподтверждённого нонса собеседника x_{n_tmp} . Попытка расшифровать зашифрованную часть сообщения собеседника с использованием нонса, посчитанного ранее (x_n), и сеансовых ключей шифрования: секретного данной стороны и открытого собеседника (m_{cs} и x_{cp}). Если расшифровка не удалась, то выдача на экран информации о произошедшей ошибке, иначе продолжение работы.

4.2. Попытка проверить электронную подпись расшифрованного сообщения собеседника с использованием сеансового публичного ключа электронной подписи собеседника x_{sp} . Если при этой проверке произошла ошибка, то выдача на экран информации о произошедшей ошибке, иначе присвоение нонсу следующего сообщения собеседника x_n значения ранее посчитанного неподтверждённого нонса собеседника x_{n_tmp} и вывод текстового сообщения на экран.

3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

3.1. Разработка графического интерфейса пользователя

По результатам предварительного анализа поставленной задачи было решено разработать для данной программы максимально понятный графический интерфейс пользователя на основе широко распространённой кроссплатформенной библиотеки *GTK+* с применением редактора графического интерфейса *Glade*. Этот редактор позволяет в режиме конструктора создавать окна и сохранять их как *XML*-документы; в самом же коде прикладной программы достаточно реализовать лишь импорт окон и логику их функционирования. При проектировании пользовательского интерфейса было решено создать три окна: окно настройки соединения, окно беседы и окно «О программе».

В первом окне, окне настройки соединения, пользователь может задать все настройки, предусмотренные в программе: задать сетевую роль, режим работы с ключами, сетевой протокол, ввести имя собеседника (опционально, по умолчанию используется имя «Собеседник»), ввести адрес сервера (при работе в режиме клиента), при необходимости изменить значение порта по умолчанию. Значение адреса сервера по умолчанию равно 127.0.0.1 для того, чтобы пользователю было удобно практиковаться в работе с программой на своём локальном компьютере перед её использованием для серьёзных задач, а также для удобства тестирования программы. Окно показано на рис. 4.

Во втором окне, окне беседы, пользователь видит в верхней половине служебные сообщения и сообщения его переписки с собеседником, а в нижней половине вводит новые сообщения для собеседника. Нажатие кнопки «Отправить» для отправки сообщения не обязательно — вместо этого можно нажать на кнопку *Enter*. Подобный интерфейс используется во многих других программах обмена мгновенными сообщениями, так что он должен быть

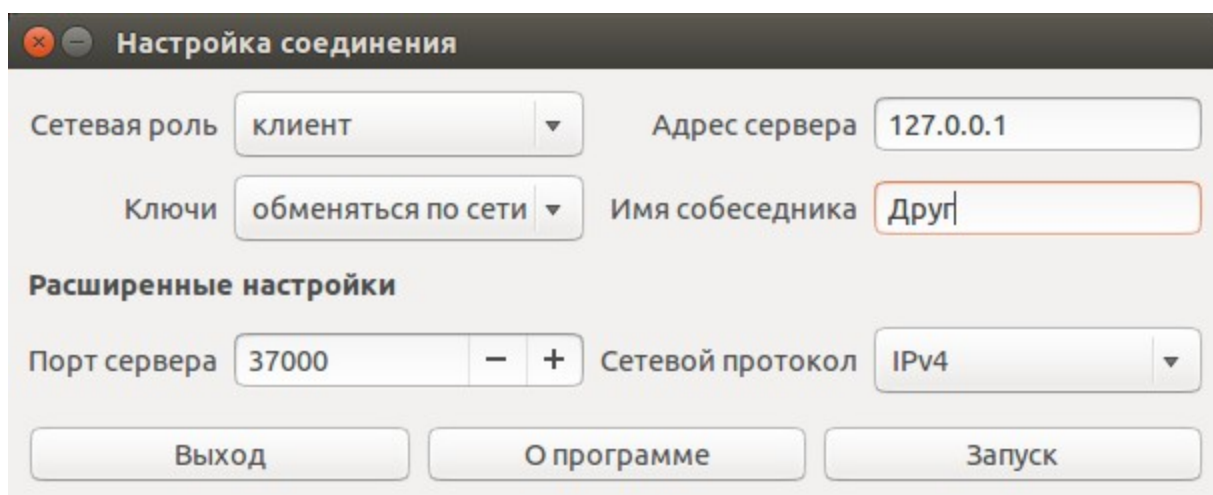


Рис. 4. Окно настройки соединения

интуитивно понятен пользователю. Вверху этого окна есть меню, позволяющее сохранить постоянные ключи этой беседы на диск, закрыть соединение, выйти из программы, посмотреть информацию о программе. Окно показано на рис. 5.

Третье окно, окно «О программе», выводит на экран сведения о названии и версии программы, её предназначении, адресе её официального репозитория в сети Интернет, авторе и лицензии. Окно показано на рис. 6.

Общий подход к выводу служебной информации в программе был выбран таким: вывод подробной информации о текущих выполняемых операциях и о возникающих ошибках осуществляется в консоль, но при реальной необходимости сообщения выводятся и в графическом интерфейсе. Последний вариант используется, когда происходят критические ошибки — например, не удалось осуществить обмен сеансовыми ключами по сети, или произошло другое важное для пользователя событие — например, закрытие соединения собеседником. Таким образом, можно сказать, что у программы есть два режима вывода — нормальный, информации из которого более чем достаточно при стабильной работе, и отладочный, необходимый для получения подробной информации о возникающих ошибках. Вывод отладочной информации в консоль показан на рис. 7.

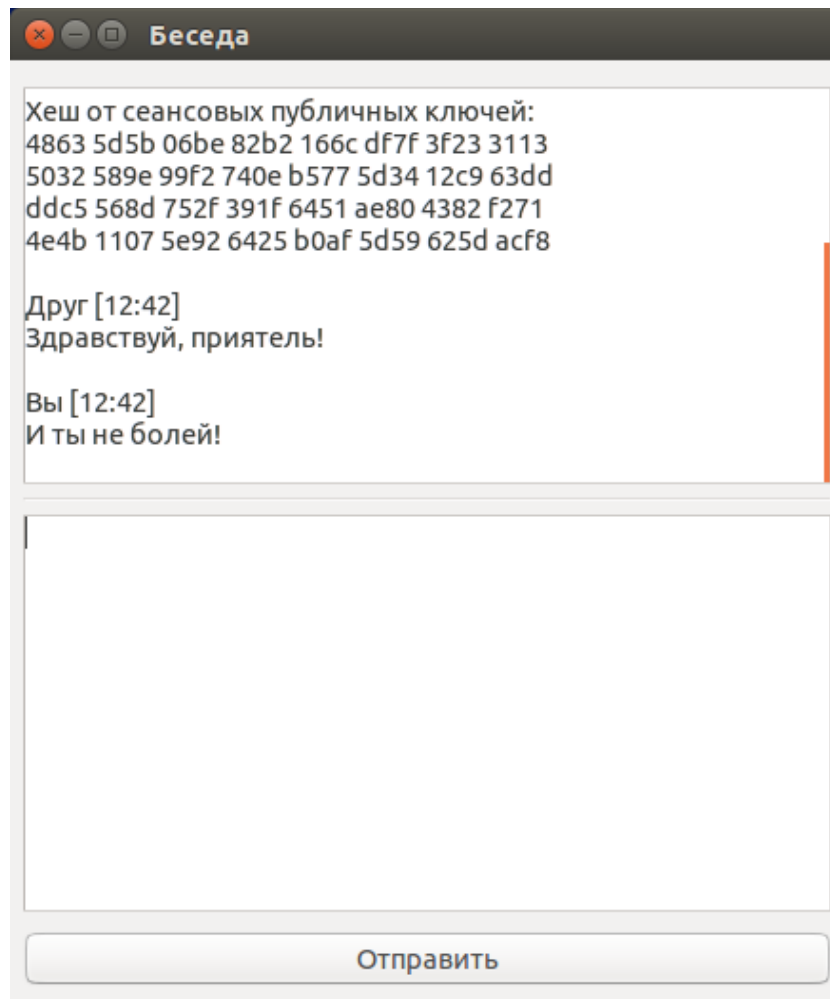


Рис. 5. Окно беседы

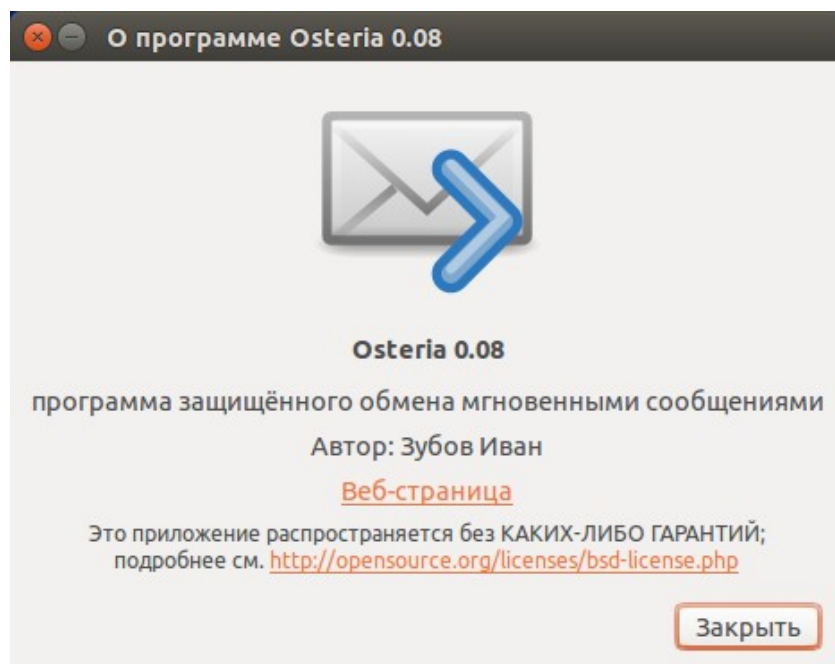
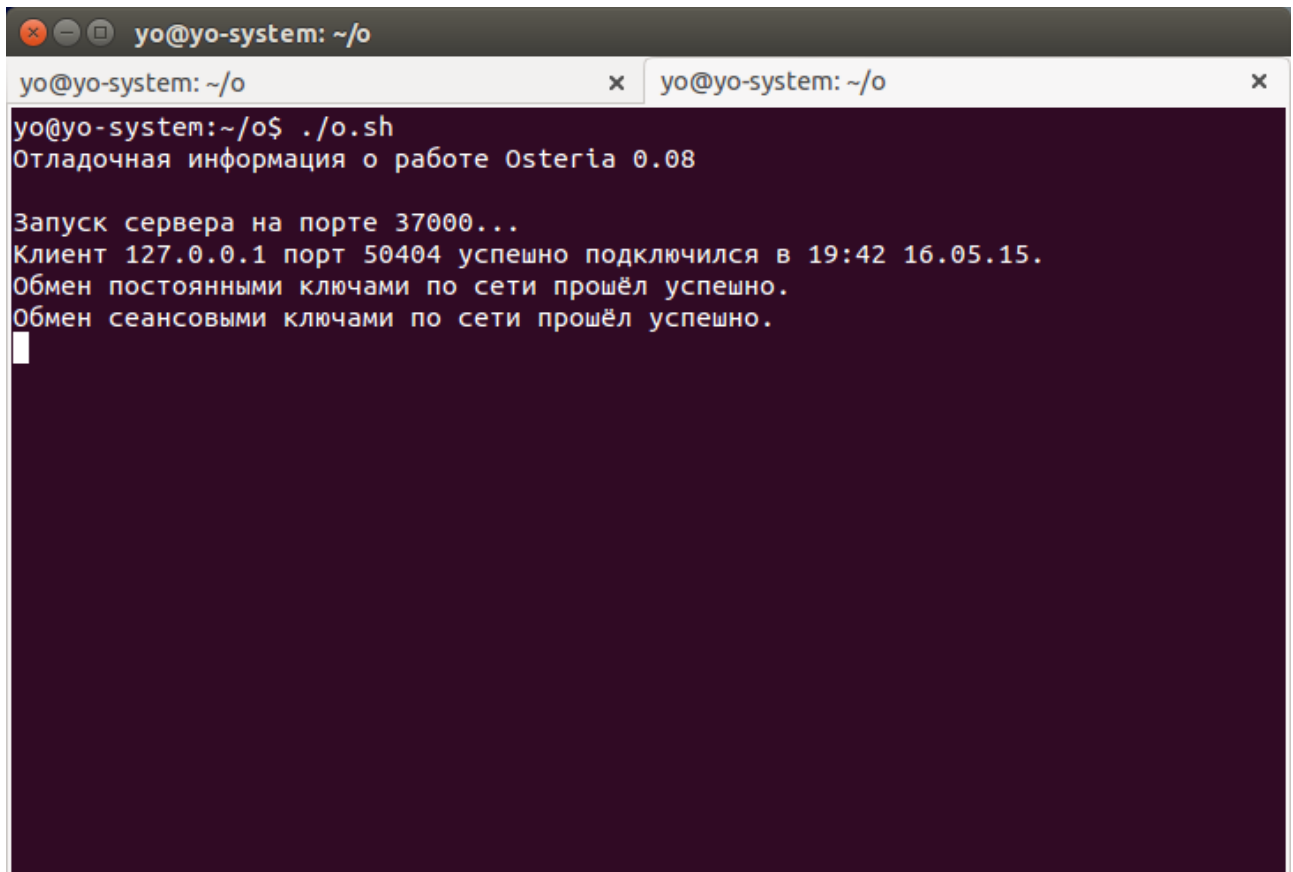


Рис. 6. Окно «О программе»



```
yo@yo-system: ~/o
yo@yo-system: ~/o$ ./o.sh
Отладочная информация о работе Osteria 0.08

Запуск сервера на порте 37000...
Клиент 127.0.0.1 порт 50404 успешно подключился в 19:42 16.05.15.
Обмен постоянными ключами по сети прошёл успешно.
Обмен сеансовыми ключами по сети прошёл успешно.
```

Рис. 7. Вывод отладочной информации в консоль

3.2. Детали реализации

Код разработанной программы приведён в приложении Б.

При написании кода выполняемые программой команды прокомментированы там, где необходимо, переменные названы осмысленно, код свёрстан в удобном для восприятия стиле. Программа разбита на несколько файлов с исходными кодами различных функций для удобства навигации и редактирования. Модуль *main* содержит логику работы графического интерфейса пользователя, модуль *crypto* — криптографический функционал программы, модуль *stuff* — прочий вспомогательный функционал, модуль *tweetnacl* — криптографическую библиотеку *TweetNaCl* последней на июнь 2015 года версии

20140427, модуль *poison* — заголовочный файл для обеспечения безопасной разработки программы.

В процессе своей работы программа выдаёт в графический интерфейс пользователя сообщения о важных возникающих ошибках и о важных текущих событиях, что призвано сделать комфортной работу пользователя с программой. В консоль выводятся более подробные сообщения о возникающих ошибках и о текущих событиях, что заметно упрощает отладку программы. При обработке данных они всегда проверяются на правильность, что делается с двумя целями: во-первых, непредсказуемая и неправильная работа программы причиняет неудобства пользователю, а во-вторых, ошибки вроде переполнения буфера представляют собой угрозу информационной безопасности всего компьютера, так как могут эксплуатироваться злоумышленниками. Во избежание излишней траты системных ресурсов и расширения поверхности атаки все запрошенные программой ресурсы (например, сокеты и структуры со служебной информацией) освобождаются в тот момент, когда они перестают быть нужными для работы программы.

После завершения работы над данной программой её исходные коды были опубликованы на хостинге *Github* [44] под лицензией *BSD* из 2-х пунктов [30, 41]. Открытая публикация программы в виде исходных кодов важна для того, чтобы другие разработчики могли использовать её в качестве основы для своих проектов, использующих что-либо из перечисленного: сетевое взаимодействие, криптографические средства защиты информации, графический интерфейс пользователя. Для того чтобы исходными кодами программы и самой программой могли пользоваться люди по всему миру, интерфейс и все файлы с исходными кодами были переведены на английский язык после первичной публикации на русском языке. Именно лицензия *BSD* из 2-х пунктов была выбрана среди множества лицензий для проектов с открытым исходным кодом за её понятность и за возможность использования кода в проприетарном программном обеспечении, что очень важно для подобных несложных проектов с примерами реализации

часто встречающихся задач.

3.3. Работа с программой

Сначала пользователи договариваются между собой о том, каким образом они будут осуществлять обмен постоянными ключами электронной подписи и шифрования. Обмен может происходить напрямую через сеть по незащищённому каналу, но существует также возможность записать сгенерированные постоянные секретные и публичные ключи на диск, обменяться постоянными публичными ключами с собеседником по защищённому каналу (например, с помощью обмена USB-флеш-накопителями при личной встрече), после чего загрузить все необходимые ключи с диска.

Для создания соединения один (любой) экземпляр программы активирует режим сервера, открывая определённый порт по сетевому протоколу *IPv4* или *IPv6* для подключения к нему. При общении через глобальную сеть Интернет очень важно проверить, чтобы этот порт был доступен для подключения извне. Для тестирования установления соединения с ним через *IPv4* по сети Интернет можно использовать удобное веб-приложение [27]. Для просмотра локальных *IP*-адресов (как *IPv4*, так и *IPv6*) нужно запустить в консоли команду «*ifconfig*», для просмотра внешних — «*wget -qO- [адрес]*» или «*curl [адрес]*», где адрес — «*v4.ident.me*» для *IPv4* или «*v6.ident.me*» для *IPv6*.

Другой экземпляр программы активирует режим клиента, который подключается к первому экземпляру, серверу. Для создания соединения клиент должен знать локальный (в случае соединения по локальной сети) или глобальный (в случае соединения по глобальной сети) *IP*-адрес сервера, используемый протокол сетевого уровня (по умолчанию *IPv4*), а также номер открытого порта (по умолчанию 37000).

После установления соединения, а также обмена постоянными ключами по сети либо загрузки их с диска собеседники могут сверить хеши от постоянных публичных ключей (своих и собеседника) для того, чтобы удостовериться в их

соответствии, что особенно важно при обмене постоянными ключами по сети. Обмен сеансовыми публичными ключами всегда происходит по сети и опирается на постоянные ключи. Собеседники могут сверить хеши от сеансовых публичных ключей (своих и собеседника) для того, чтобы удостовериться в их соответствии. Хеши выводятся в шестнадцатеричном коде, в удобном для человека виде, что позволяет сверить их как визуально (например, передав значение хеша через доверенный ресурс), так и аудиально (например, продиктовав значение хеша по телефону). Если происходил обмен постоянными ключами по сети, после сверки хешей их можно сохранить с помощью соответствующего пункта меню окна беседы.

Затем пользователи осуществляют двухсторонний обмен мгновенными сообщениями по защищённому каналу. Для разрыва соединения любой из собеседников завершает программу каким-либо способом либо выбирает в меню окна беседы пункт «Заккрыть соединение».

3.4. Эксперименты

В процессе написания программы она постоянно тестировалась на том же компьютере, на котором она писалась, для проверки работоспособности изменений. Кроме этого, после серьёзных изменений она тестировалась в домашней локальной сети из компьютера и ноутбука, соединённых через WiFi-роутер.

С тестированием работы в глобальной сети Интернет мне помогал мой одноклассник Даниленко Егор. При первой попытке нам не удалось установить соединения между нашими компьютерами — сервер безрезультатно ожидал подключения, а клиент не мог к нему подключиться, прерываясь из-за истечения таймаута. Так было и когда Егор запускал клиент, а я — сервер, так и наоборот. Наши компьютеры были подключены к Интернет через WiFi-роутеры, и проблема

была вызвана именно этим: роутеры просто блокировали попытку программы открыть слушающий серверный сокет, доступный для подключения из глобальной сети Интернет. Когда я подключил компьютер к Интернет напрямую и запустил на нём сервер, клиенту на компьютере Егора удалось соединиться с ним и успешно обменяться сообщениями.

Позже, когда я разрешил в настройках своего роутера открытие слушающего сокета на порту 37000 (осуществил проброску портов), клиенту на компьютере Егора удалось подключиться к моему компьютеру, после чего мы обменялись несколькими сообщениями. Затем работу программы при соединении через Интернет несколько раз со мной успешно тестировал Елманов Андрей из группы А-09-11.

Для того, чтобы убедиться в шифровании передаваемой информации, был использован анализатор трафика *Wireshark* [36]. Просмотр передаваемой по сети информации показал, что данные действительно зашифрованы (прочитать их было невозможно, данные походили на случайные за исключением длин сообщений), причём сообщения с одинаковой информацией, переданные в одном сеансе связи, имели совершенно разный вид за счёт использования разных нонсов при шифровании. Информация, которую можно было сгенерировать на приёмной стороне, генерировалась именно там без её передачи по сети (это касается нулевых байт в начале каждого зашифрованного сообщения, а также всех нонсов кроме первого с каждой стороны). Данные этого эксперимента, полученные с помощью самой программы и анализатора трафика приведены в табл. 1. В эксперименте был выбран режим обмена постоянными ключами по сети с использованием сетевого протокола *IPv4*.

Таблица 1

Данные, полученные в результате эксперимента

Источник данных, их объём	Содержание данных	Данные в шестнадцатеричном коде
передача от клиента серверу, 64 байта	постоянные открытые ключи электронной подписи и шифрования клиента в открытом виде	18 0f 4d 78 17 97 4e 84 b4 80 c4 65 20 0a 3d 24 27 3a ae 7c 6e e7 8b ee f7 9c e9 0c 30 70 1f 9e 34 5f 9c 48 61 f3 9c d2 17 7f 5c 23 d6 78 ef ef 8a 89 d0 02 db 81 c6 b3 19 2d 8f c7 f5 46 79 47
передача от сервера клиенту, 64 байта	постоянные открытые ключи электронной подписи и шифрования сервера в открытом виде	fd 32 40 0a 91 81 75 f2 37 39 c0 5f 69 a1 26 32 f1 d5 07 0e ad cb f3 3e 0a 0b 9f 2d 54 01 6b 7a 6f 29 e7 ec 76 d9 2b 1c 50 24 47 53 a2 4e 2e 94 2c 41 05 72 a4 78 54 13 30 c3 62 41 f3 59 58 43
вычисление на клиенте, 64 байта	хеши от постоянных открытых ключей электронной подписи и шифрования обеих сторон	8f 64 74 02 b5 88 e1 99 c7 02 df 36 2a 69 52 b0 e6 cd 49 01 f3 5d ad f4 16 d5 bc ca 6a 96 2e bb dc 43 f8 7a 24 96 74 ac 98 12 1c d2 e1 f4 0c 2c 91 70 d0 5b 9d 1e 6d db 27 9b a9 62 25 8c 26 9f
вычисление на сервере, 64 байта	то же	8f 64 74 02 b5 88 e1 99 c7 02 df 36 2a 69 52 b0 e6 cd 49 01 f3 5d ad f4 16 d5 bc ca 6a 96 2e bb dc 43 f8 7a 24 96 74 ac 98 12 1c d2 e1 f4 0c 2c 91 70 d0 5b 9d 1e 6d db 27 9b a9 62 25 8c 26 9f
передача от клиента серверу, 24 байта	начальный нонс клиента	e0 5b c7 77 65 2c 5d ce da 1f 19 ba b0 fd a3 6e ad 11 b5 b2 84 e0 b3 41
передача от клиента серверу, 144 байта	сеансовые открытые ключи электронной подписи и шифрования клиента, подписанные и зашифрованные с помощью постоянных ключей и начального нонса клиента	c7 fa 45 ea 2e 16 89 de 25 5f f3 6e b1 be aa 3c 08 82 b4 e2 d7 5c 81 c7 d6 ef 62 69 52 ea 6e 18 79 c5 12 89 ae 39 4d 8f 96 5e 50 5a 48 bb 7b 66 c5 85 35 e4 dd 41 45 34 65 c7 98 cd c3 cb 2b 51 67 fb 22 2f 7f 44 d9 33 56 89 f5 7d 48 3d 0d f0 cc 7c 1d 6b e1 89 86 4b f9 1e ea 23 b0 db 5b b7 e7 e2 80 fb e6 29 12 a8 50 2c 1d 38 e0 6f 2a f0 17 44 71 79 30 85 21 18 47 60 6e b4 47 20 00 f6 eb d5 a2 18 88 64 2c ef 01 3d ee 54 06 46 96 f8
передача от сервера клиенту, 24 байта	начальный нонс сервера	f3 a3 cf 24 e6 d2 a3 a1 c7 60 b3 27 1f 86 fb f7 a1 b0 9f 71 4a b3 89 25

Источник данных, их объём	Содержание данных	Данные в шестнадцатеричном коде
передача от сервера клиенту, 144 байта	сеансовые открытые ключи электронной подписи и шифрования сервера, подписанные и зашифрованные с помощью постоянных ключей и начального нонса сервера	43 c8 5e 15 9c 6d 06 5e be b0 bf aa 44 0e bb f8 2b d8 db db 0c e4 92 ce 8f 22 ed 9d c0 1f 8c bf 55 da c7 ce 8f e2 66 33 7f dc a3 6c 2a 82 6d 4f ce 91 c0 94 20 71 f6 aa a3 02 2b be 2f 51 08 07 c0 3a f9 80 1a b0 61 44 3c d4 d5 96 53 4b ff e1 70 09 a2 ac 6e f9 d0 61 0d c4 41 f3 0b f4 3c 3b db 6a af 02 02 5f 0a 97 16 eb c7 03 38 9c 9a 4d 7c fe ab 84 5d 24 42 2b 26 16 89 99 e6 03 14 5b 42 10 bc 34 4f 22 71 d1 3a d3 92 cf 92 2c c4 5f
вычисление на клиенте, 64 байта	хеши от сеансовых открытых ключей электронной подписи и шифрования обеих сторон	57 b8 ab c0 b8 64 68 b0 ea 92 2e 7b a4 84 1e fb 17 e3 28 e7 b2 ca 36 ed 88 09 98 bd eb bf 77 2c 69 5c b4 45 8e 3d d3 02 25 17 81 f2 a2 88 e4 95 1c ce e9 71 d2 58 4d 82 7e 04 ef 17 a0 37 54 ae
вычисление на сервере, 64 байта	то же	57 b8 ab c0 b8 64 68 b0 ea 92 2e 7b a4 84 1e fb 17 e3 28 e7 b2 ca 36 ed 88 09 98 bd eb bf 77 2c 69 5c b4 45 8e 3d d3 02 25 17 81 f2 a2 88 e4 95 1c ce e9 71 d2 58 4d 82 7e 04 ef 17 a0 37 54 ae
передача от сервера клиенту, 2 байта	длина следующего подписанного и зашифрованного сообщения сервера — 84 байта	00 54
передача от сервера клиенту, 84 байта	пользовательское сообщение «zzz», подписанное и зашифрованное с помощью сеансовых ключей и текущего нонса	b2 a3 24 95 34 5b 95 52 0f 28 54 71 98 84 82 df 12 87 c7 ae 30 8f d3 dd 26 86 12 f5 00 00 b8 22 02 51 a2 83 3b 3e 1b b8 15 7c 12 3a d5 79 69 e7 28 a9 8c 9a fa 90 fa b4 20 80 73 58 f1 89 f2 e1 e1 65 59 8b 9c 95 4f dd dc 20 25 38 b4 c0 0e 11 d8 a5 40 fe
передача от сервера клиенту, 2 байта	длина следующего подписанного и зашифрованного сообщения — 84 байта	00 54
передача от сервера клиенту, 84 байта	пользовательское сообщение «zzz», подписанное и зашифрованное с помощью сеансовых ключей и текущего нонса	c0 2d a7 c1 ca 93 ad 8d 5b 1e e7 46 c9 1d 62 de 3c 6a ad 9b d2 38 e4 69 1c de 10 83 99 3e a8 86 b4 d4 9d 17 09 c3 f0 91 ea 64 15 8a 34 21 dd 31 4b ca 73 fa aa 88 89 23 21 05 1d 50 b1 68 78 23 ad c4 96 80 16 99 e6 20 9c da a2 87 07 86 31 4f 3b ad b2 cf

Продолжение таблицы 1

Источник данных, их объём	Содержание данных	Данные в шестнадцатеричном коде
передача от клиента серверу, 2 байта	длина следующего подписанного и зашифрованного сообщения — 84 байта	00 54
передача от клиента серверу, 84 байта	пользовательское сообщение «zzz», подписанное и зашифрованное с помощью сеансовых ключей и текущего нонса	a8 fd dc 6f 4a f4 65 e4 81 97 18 11 69 b3 e1 71 11 d3 e2 94 a8 b3 f9 9d 83 a6 c6 51 34 a9 a5 a0 3c 79 59 9c 00 f9 6e 25 81 66 e9 09 29 23 4d 51 b2 33 43 bd f4 dc 4a 5a d7 6f e7 a9 d9 96 f9 a7 88 5a ae 2d eb 53 77 6c 91 df 60 c3 1b 0a f7 f2 08 d5 9a a9
передача от клиента серверу, 2 байта	длина следующего подписанного и зашифрованного сообщения — 84 байта	00 54
передача от клиента серверу, 84 байта	пользовательское сообщение «zzz», подписанное и зашифрованное с помощью сеансовых ключей и текущего нонса	aa ca c8 c0 b0 5e 16 f7 44 37 f7 38 c5 40 59 2f d7 96 0d d8 a1 98 ec a8 36 99 79 27 6f 94 55 e6 e8 ec 65 9a 29 2e 21 76 bf e6 22 72 36 88 bd 01 55 ed b2 6e a1 00 a2 40 d7 27 49 21 09 70 47 11 93 1f 1a b0 1d 99 9d fa 19 60 1d 9d 8e 1f 96 eb eb 45 61 91
передача от клиента серверу, 2 байта	длина следующего подписанного и зашифрованного сообщения — 106 байт	00 6a
передача от клиента серверу, 106 байт	пользовательское сообщение «Привет, АБВГД!», подписанное и зашифрованное с помощью сеансовых ключей и текущего нонса	3f 61 de 25 87 56 b0 f0 07 39 8c 4d f6 51 b9 43 40 67 40 92 df 83 c0 97 6b fc 38 ae ce ff ec 8a 20 ab b3 a4 2f 18 d9 ac e3 d3 06 5a dc 8d 72 19 40 0e 17 e8 63 ae 8f fc 9b c0 c0 c5 0d ed 9c ca 4a 4a e7 2e 04 3a f0 a9 f9 c6 b1 c4 14 85 ad 28 17 16 15 73 ec 4e 9c 30 c2 9c a2 68 b0 08 a9 88 db 5d 1c 56 55 57 4b 43 02 ff

Заключение

В результате разработки получена программа для операционной системы *Ubuntu*, позволяющая двум пользователям обмениваться между собой текстовыми сообщениями по локальной или глобальной сети. При использовании один (любой) экземпляр программы подключается в качестве клиента к другому экземпляру (серверу), и с помощью установленного соединения пользователи осуществляют двухсторонний обмен мгновенными сообщениями. Сообщения пользователей защищены криптографической схемой на основе вызова функций из сторонней криптографической библиотеки *TweetNaCl*. Криптографическая схема включает в себя электронную подпись для аутентификации сообщений, шифрование с открытым ключом, а также подтверждение соответствия открытых ключей шифрования и электронной подписи собеседников с помощью криптографической хеш-функции.

Составными частями разработки является данная работа, содержащая письменное описание использованных методов, средств и алгоритмов, схема алгоритма программного экземпляра, а также собственно программа, содержащая функции организации соединения, защищённого обмена мгновенными сообщениями, построения графического интерфейса пользователя и вспомогательные функции.

Разработка выполняет функции установления соединения (реализация различна для режима клиента и для режима сервера) по сетевым протоколам *IPv4* и *IPv6*, генерации и обмена постоянными (два способа) и сеансовыми (один способ) ключами электронной подписи и шифрования, шифрования и расшифрования сообщений, подписывания сообщений электронной подписью и проверки электронной подписи сообщений, подтверждения соответствия открытых ключей шифрования и электронной подписи собеседников с помощью криптографической хеш-функции, пересылки сообщений с помощью установленного защищённого соединения.

Разработка выполнена и протестирована на операционной системе *Ubuntu 14.04 LTS*, аппаратная платформа разработки — *x86*. За счёт использования функций стандартной библиотеки языка программирования *C* и высокой стандартизации различных дистрибутивов операционной системы *GNU/Linux*, данная разработка должна правильно работать также и на современных версиях других дистрибутивов этой операционной системы. Интерфейс сокетов Беркли является частью группы стандартов *POSIX*, поэтому программа должна работать на всех современных *UNIX*-подобных системах с минимальными изменениями. При использовании библиотеки *Winsock* и незначительной правке исходного кода данная разработка должна правильно работать также и на операционных системах семейства *Windows*.

Графический интерфейс пользователя удовлетворяет требованию удобной работы с программой самых различных пользователей, в том числе не являющихся специалистами по вычислительной технике. Он основывается на применении широко распространённой кроссплатформенной библиотеки элементов интерфейса *GTK+* для обеспечения высокой мобильности по отношению к программным платформам.

При переносе на другие аппаратные платформы, например *IA-64* или *ARM*, могут возникнуть небольшие проблемы с типами данных, но они не потребуют большого количества правок исходного кода. Таким образом, разработка получилось очень мобильной по отношению к программным и аппаратным платформам.

Данная разработка может быть применена пользователями глобальной или локальной сети для личной и деловой переписки, а также разработчиками программного обеспечения в качестве основы для их проектов в областях сетевого программирования, применения криптографических средств защиты информации, построения графического интерфейса пользователя, что позволит им ускорить процесс разработки своих продуктов.

Для работы программы в идеале требуется её запуск на компьютере с операционной системой *Ubuntu 14.04 LTS* и процессором архитектуры *x86* либо совместимым, так как тестирование происходило именно на этой конфигурации. В случае использования других операционных систем и/или аппаратных платформ может потребоваться перекомпиляция и незначительные изменения исходного кода.

В будущем в первую очередь стоит внедрить функционал передачи файлов, обеспечить возможность простой компиляции под разные программные и аппаратные платформы, провести интернационализацию интерфейса, а также расширить возможности программы по защите информации как при её передаче по сети, так и при использовании на конечных устройствах пользователей. Заметными плюсами стали бы построение сети доверия, подобной используемой в программе *PGP*, возможности общения клиентов через центральный сервер, сохранения истории переписки в файл, гибкой настройки различных параметров через файл конфигурации, удобного одновременного общения с несколькими пользователями в одном или нескольких чатах.

Список использованных источников

1. Магда Ю.С. *UNIX для студента*. — СПб.: БХВ-Петербург, 2007. — 480 с.
2. Гласс Г., Эйблс К. *UNIX для программистов и пользователей*. — СПб.: БХВ-Петербург, 2004. — 848 с.
3. Brian "Beej Jorgensen" Hall. «Beej's Guide to Network Programming» — <http://beej.us/guide/bgnet>
4. Norman Matloff. Overview of Computer Networks — <http://heather.cs.ucdavis.edu/~matloff/Networks/Intro/NetIntro.pdf>
5. Программа обмена мгновенными сообщениями *Telegram* <https://telegram.org>
6. Telegram Reaches 1 Billion Daily Messages <https://telegram.org/blog/billion>
7. Программа обмена мгновенными сообщениями *Cryptocat* <https://crypto.cat>
8. Криптографическая библиотека *TweetNaCl* <http://tweetnacl.cr.yp.to>
9. Сетевая библиотека *Winsock* <https://msdn.microsoft.com/ru-ru/library/dd335942.aspx>
10. Криптографическая библиотека *OpenSSL* <https://www.openssl.org>
11. Cryptocat Design and Functionality <https://github.com/cryptocat/cryptocat/wiki/Design-and-Functionality>
12. Cryptocat Threat Model <https://github.com/cryptocat/cryptocat/wiki/Threat-Model>
13. Cryptocat Server Deployment Instructions <https://github.com/cryptocat/cryptocat/wiki/Server-Deployment-Instructions>
14. Telegram FAQ <https://telegram.org/faq>
15. Telegram FAQ for the Technically Inclined <https://core.telegram.org/techfaq>
16. Daniel J. Bernstein, Bernard van Gastel, Wesley Janssen, Tanja Lange, Peter Schwabe, Sjaak Smetsers. TweetNaCl: a crypto library in 100 tweets — <http://tweetnacl.cr.yp.to/tweetnacl-20140917.pdf>
17. Rutger Engelhard — Securing Communication <http://nacl.cr.yp.to/securing->

[communication.pdf](#)

18. Daniel J. Bernstein, Tanja Lange, Peter Schwabe — The security impact of a new cryptographic library <http://cr.yp.to/highspeed/coolnacl-20120725.pdf>

19. Daniel J. Bernstein — Entropy Attacks! <http://blog.cr.yp.to/20140205-entropy.html>

20. Функции библиотеки *OpenSSL Crypto* <https://www.openssl.org/docs/crypto>

21. Операционная система *Ubuntu* <http://www.ubuntu.com>

22. Ubuntu adoption and reception
[https://en.wikipedia.org/wiki/Ubuntu_\(operating_system\)#Adoption_and_reception](https://en.wikipedia.org/wiki/Ubuntu_(operating_system)#Adoption_and_reception)

23. Компилятор *GCC* <https://gcc.gnu.org>

24. Библиотека элементов интерфейса *GTK+* <http://www.gtk.org>

25. Заголовочный файл *GCC Poison* <http://blog.leafsr.com/2013/12/02/gcc-poison>

26. Service Name and Transport Protocol Port Number Registry — IANA
<http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt>

27. Custom Port Test <http://whatsmyip.org/port-scanner>

28. Криптография <https://ru.wikipedia.org/wiki/Криптография>

29. Криптографическая библиотека *NaCl* <http://nacl.cr.yp.to>

30. The *BSD 2-Clause License* <http://opensource.org/licenses/bsd-license.php>

31. Атака типа «Человек посередине»
https://ru.wikipedia.org/wiki/Человек_посередине

32. Криптографический пакет *PGP* <https://ru.wikipedia.org/wiki/PGP>

33. Совершенная прямая секретность
https://ru.wikipedia.org/wiki/Perfect_forward_secrecy

34. Атака повторного воспроизведения
https://ru.wikipedia.org/wiki/Атака_повторного_воспроизведения

35. Редактор графического интерфейса *Glade* <https://glade.gnome.org>

36. Анализатор трафика *Wireshark* <https://www.wireshark.org>

- 37. Сокеты Беркли https://ru.wikipedia.org/wiki/Сокеты_Беркли
- 38. Хеширование <https://ru.wikipedia.org/wiki/Хеширование>
- 39. Криптографическая хеш-функция
https://ru.wikipedia.org/wiki/Криптографическая_хеш-функция
- 40. Nonce, нонс <https://ru.wikipedia.org/wiki/Nonce>
- 41. Лицензия *BSD* https://ru.wikipedia.org/wiki/Лицензия_BSD
- 42. Группа стандартов *POSIX* <https://ru.wikipedia.org/wiki/POSIX>
- 43. Расиряемый язык разметки *XML* <https://ru.wikipedia.org/wiki/XML>
- 44. Официальный репозиторий *Osteria* <https://github.com/postboy/osteria>