Московский институт электроники и математики НИУ ВШЭ
**Кафедра Информационно-коммуникационных технологий**

# ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к дипломному проекту
На тему: ≪Защищенный канал связи≫

Студент: **Майоров Алексей Владимирович**

Руководитель проекта: **Леохин Юрий Львович**

Допущен к защите _____ 2013 г.

Консультанты проекта:

Специальная часть **Леохин Юрий Львович**

Охрана труда **Михайлов Е. Б.**

Зав. кафедрой проф. д.т.н. **Азаров В. Н.**

**Москва 2013**

# Аннотация

В дипломной работе описан процесс разработки защищенного канала связи. Он разбит на несколько этапов: сбор информации о различных угрозах компьютерной сети, обзор существующих решений, разработка архитектуры и реализация канала связи и тестирования системы для возможности внедрить в компанию.

# Содержание

# Раздел 1. Обзорно-аналитическая часть

## 1.1 Введение

Защита информации является неотъемлемой составной частью общей проблемы информационной безопасности, роль и значимость которой во всех сферах жизни и деятельности общества и государства на современном этапе неуклонно возрастают.

Производство, связь, транспорт, банковское дело, финансы, наука и образование, средства массовой информации всё больше зависят от интенсивности информационного обмена, полноты, своевременности, достоверности и безопасности информации.

В связи с этим проблема безопасности информации стала предметом острой озабоченности руководителей органов государственной власти, предприятий, организаций и учреждений независимо от их организационно-правовых форм и форм собственности.

Бурное развитие средств вычислительной техники открыло перед человечеством небывалые возможности по автоматизации умственного труда и привело к созданию большого числа разного рода автоматизированных информационно-телекоммуникационных и управляющих систем, к возникновению принципиально новых, так называемых информационных технологий.

Интернет - это глобальная сеть, насчитывающая тысячи компьютеров, разбросанных по всему миру. Когда два компьютера взаимодействуют между собой, весь трафик от источника до пункта назначения проходит через множество других устройств (маршрутизаторов, коммутаторов и т.д.). Эти сетевые устройства администрируют посторонние организации, и никто не может поручиться за их честность (в большинстве случаев невозможно узнать заранее, по какому пути пойдут пакеты к пункту назначения).

На любом из хостов, встречающихся на пути пакетов, можно просмотреть их содержимое или изменить что-либо в них. Все это создает

серьезные и существенные проблемы, чем выше уровень значимости или конфиденциальности передаваемых данных.

Для решения этоих проблем и применяются защищенные информационные каналы. Их можно представить себе как некий туннель. Информация помещается с одной стороны туннеля и прочесть ее можно только с другой стороны.

На самом деле, передаваемая информация модифицируется таким образом, чтобы её невозможно было изменить или просмотреть на пути её следования. При применении этого механизма обеспечивается как сокрытие информации, так и невозможность ее подмены на всем пути ее следования.

## 1.2 Виды угроз безопасности

Проблема обеспечения безопасности становится всё более актуальной в связи с более интенсивным развитием вычислительных средств и различных систем передачи информации. Все меры по безопасности направляются на предотвращение насанкционированного доступа к информации.

Возможности злоумышленников по взлому информации, которая передаётся по каналаам связи, развиваются и совершенствуются с огромной скоростью. Для защиты информации требуется использование специальных методов и средств, а не просто разработка частных механизмов защиты.

Какое-либо действие или событие, которое приводит к несанкционированному доступу к информационному ресурсу, разрушению или искажению хранимой, обрабатываемой или передаваемой информации, называется угрозой безопасности.

Угрозы разделяются на умышленные (преднамеренные) и случайные (непреднамеренные). Источником случайных угроз могут быть ошибки в программном обеспечении, неправильные действия пользователя или администратора, выход из строя каких-либо аппаратных средств. Главная же цель умышленных угроз – это нанесение ущерба всем пользователям информационной системы.

Умышленные угрозы делятся на два типа угроз: активные и пассивные.

Пассивные угрозы не оказывают влияние на функционирование, они направлены лишь на использование несанкционированной информации. Один из примеров пассивной угрозы – прослушивание. Путём прослушивания каналов связи возможно получить доступ к нужной информации.

Активные угрозы оказывают целенаправленное воздействие на программные и аапаратные ресурсы. И их целью является нарушение процесса функционирования всей системы. Разрушение линий связи, вывод из строя устройств или операционных систем, искажение сведений в базах –

всё это является активными угрозами. Источником этих угроз чаще всего являются действия самого злоумышленника или трояны и вирусы.

Основные угрозы безопасности:

- *Несанкционированный обмен информацией.*

Несанкционированный обмен информацией между пользователями может приветси к получению одним из них данных, доступа к которым у него быть не должно. Это равносильно раскрытию информации.

- *Несанкционированный доступ к информации.*

При получении несанкционированного доступа к защищенной информации у злоумышленника появляется возможность нанести огромный вред всей системе (например, корпоративной сети).

- *Раскрытие конфиденциальности информации.*

Для раскрытия конфидециальности информации можно использовать прослушивание каналов или несанкционированный доступ к базам данных.

- *Компрометация информации.*

Чтобы скомпрометировать информацию чаще всего вносят ложные изменения в базы данных. Из-за этого пользователь либо отказывается от пользования этой базой, либо принимает попытки по её восстановлению для того, чтобы найти ложные данные и их устранить. Если же продолжится использование этой базы, то возможны принятия неверных решений на основе ложных данных, что может вылиться в тяжелые последствия.

- *Несанкционированное использование информационных ресурсов.*

Даже если не касаться информации, её можно использовать в ущерб для пользователей и администрации. Ущерб может быть достаточно

сильным, в зависимости от ценности и продуктивного использования полученной информации.

- *Ошибочное использование ресурсов.*

Данный вид угрозы чаще всего является просто следствием ошибок программного обеспечения. Тем не менее всё равно может привести к случайному раскрытию каких-либо ресурсов.

- *Отказ от обслуживания.*

Источником отказа от обслуживания чаще всего служит сама вычислительная система.

- *Отказ от информации.*

Суть отказа состоит в том, что отправитель или получатель не признает факт приёма или получения данных.

Основными способами несанкционированного доступа к информации являются:

- перехват электронных излучений;
- принудительное электромагнитное облучение линий связи с целью получения паразитной модуляции;
- применение подслушивающих устройств;
- дистанционное фотографирование;
- перехват акустических излучений и восстановление текста принтера;
- хищение носителей информации и документальных отходов;
- чтение остаточной информации в памяти системы после выполнения несанкционированных запросов;
- копирование носителей информации с преодолением мер защиты;

- маскировка под зарегистрированного пользователя;
- использование недостатков языков программирования и операционных систем;
- включение в библиотеки программ специальных блоков типа «Троянский конь»;
- незаконное подключение к аппаратуре и линиям связи;
- злоумышленный вывод из строя механизмов защиты;
- внедрение и использование компьютерных вирусов.

## 1.3 Классификация сетевых атак

*Снифферы пакетов*

Сниффер – это программа, использующая сетевую карту, которая работает в promiscuousmode. Суть этого режима в том, что пакеты, которые сетевой адаптер получает по физическим каналам, отправляются на обработку приложения. Пакеты передаются через домен, и сниффер просто перехватывает все сетевые пакеты. На данный момент снифферы законны. Их часто используют для анализа трафика или диагностик различных неисправностей. Благодаря снифферу всё равно можно узнать конфидециальную и полезную информацию, потому что многие сетевые приложения передают данные в текстовом формате (telnet, FTP, SMTP).

Пользователи зачастую применяют одинаковые логины и пароли к различным приложениям, поэтому перехват имен и паролей создает большую опасность. Есть пользователи, которы для доступа ко всем приложениям используют один пароль. Когда приложение работает в режиме клиент-сервер и все данные передаются в текстовом формате, то такую информацию легко можно перехватить и использовать для доступа к другим ресурсам. Злоумышленники знают, что часто используется один пароль на множество различных ресурсов. И, когда удаётся узнать пароль от одного из них, можно получить доступ к множесту других ресурсов, где имеется важная информация. Когда злоумышленник получает доступ к ресурсу на системном уровне, то может создать нового пользователья, который в любой момент сможет использовать этот ресурс и всю информацию.

*IP-спуфинг*

Выдача злоумышленника за санционированного пользователя внутри сети называется IP-спуфингом. Можно использовать два способа для достижения этой цели. При первом способе злоумышленник может быть авторизован внешним адресом, которому разрешен доступ к сетевым ресурсам или же использовать IP-адрес, который находится в пределах

доступа санкционированных IP-адресов. Чаще всего IP-спуффинг это начальная точка для других атак. Например, Dos-атака начинается с чужого адреса, благодаря чему скрывает настоящего злоумышленника.

Чаще всего главной задачей IP-спуфинга является добавление вредоносных команд или ложных данных в общий поток данных между сервером и клиентом. Чтобы отправить трафик на ложный IP-адрес, то есть для получения двусторонеей связи, злоумышленнику нужно будет поменять таблицы маршрутизации. Но чаще всего это не нужно, потому что главная задача состоит в получении доступа к данным от системы. А ответит ли приложение особого значения не имеет.

Но, если злоумышленник меняет таблицы маршрутизации и перенаправляет трафик на ложный IP-адрес, он получит все пакеты и будет иметь возможность отвечать на ник, как будто он является тем самым санкционированным пользователем.

*Отказ в обслуживании (Denial of Service - DoS)*

Самая популярная форма атаки – это DoS-атака. Против таких атак очень трудно быть защищенным на сто процентов. Сами по себе DoS-атаки очень просты, чтобы организовать атаку этого типа много знаний не надо. Но это тот случай, когда за простотой в реализации возможен огромный причиняемый вред. Поэтому к DoS-атакам относятся очень серьёзно и администраторы пристально следят за этим. Основные разновидности DoS-атак:

- Trinity
- PoD (Ping of Death)
- TCP SYN Flood
- Trinco
- TFN

Суть DoS-атак не похожа на другие атаки. С помощью этих атак не получить какой-либо информации из сети или доступа к ней. Из-за Dos-атаки сеть становится недоступной для пользования из-за того, что превышается допустимый предел функционирования этой сети.

Смысл DoS-атаки заключается в том, чтобы занять все соединения, которые использует серверное приложение и не допускать обслуживания обычных пользователей, держа соединения в занятом состоянии.(например, FTP или WEB сервера). Для DoS-атак используются обычные TCP и ICMP протоколы. DoS-атаки опираются на общие слабости во всей системной архитектуре, а не на какие-либо программные ошибки или бреши в системе. При DoS-атаках производительность всей сети сводится к нулю. Её переполняют множеством ненужных пакетов или сообщают ложную информацию о состоянии сетевых ресурсов. Такие атаки очень тяжело предотвратить, потому что нужно будет координировать все действия с провайдером. Если трафик, который предназначен, чтобы переполнить сеть, не остановить у провайдера, то на входе в саму сеть это сделать уже не удастся, потому что полоса пропускания уже будет забита. Распределенная атака DoS или DDoS-атака – это атака типа DoS, которая одновременно проводится с различных устройств.

*Парольные атаки*

Троянский конь, простой перебор (bruteforce), IP-спуфинг или же сниффинг пакетов - всё это может привести к парольной атаке. Зачастую, благодаря IP-спуфингу и сниффингу пакетов можно получить логин и пароль. Но злоумышленники также пытаются подобрать его путём многочисленных попыток доступа. Этот метод называется простым перебором или bruteforceattack. Для таких атак используется специальная программа, которая пытается получить доступ к серверу. Если пароль подбирается, то злоумышленник получает доступ к информации на правах обычного пользователя. Но если у пользователя есть дополнительные

привелегии доступа, то злоумышленник может организовать себе “проход”, который будет использовать в будущем для доступа к ресурсам, например, если пользователь поменяет свой пароль.

Главная проблема возникает, когда пользователь применяет один и тот же пароль ко всем системам : в Интернете или в корпоративной сети. Так как устойчивость самого слабого хоста равна устойчивости пароля, то злоумышленник, получивший пароль через этот хост, будет иметь доступ ко все остальным ресурсам, например, корпоративной сети.

*Атаки типа Man-in-the-Middle*

Для того, чтобы злоумышленник смог организовать атаку типа Man-in-the-Middle, у него должен быть доступ к пакетам, которые передаются по сети. Такой доступ к пакетам, которые передаются от провайдера, имеет сотрудник этого провайдера. При атаках Man-in-the-Middle используются протоколы маршрутизации и транспортные протоколы. Основная цель этих атак – анализ трафика и получение доступа к сетевым ресурсам, перехват сессий и получение информации о пользователях сети, организация и проведение DoS-атак, замена передаваемых данных или подмена на ложную информацию.

*Атаки на уровне приложений*

Есть несколько способов для проведения атак на уровне приложений. Один из самых популярных заключается в использовании слабостей серверного программного обеспечения (FTP, HTTP). Злоумышленник, благодаря этим слабостям, может получить доступ к компьютеру от имени пользователя, который работает с приложением. Чаще всего это администратор с правами доступа к системе, а не простой пользователь. Сведений об этих атаках много и они часто пополняются и обновляются, поэтому администраторы могут спокойно исправлять проблемы с помощью патчей.

Основная проблема с таким типом атак состоит в том, что они чаще всего используют порты, которым разрешается проходить через межсетевой экран. Например, злоумышленник соберется использовать для атак 80 TCP порт. Межсетевой экран представляет доступ к этому порту пользователю при работе с Web-сервером. Поэтому атака через этот порт будет рассматривать как обычный трафик для 80 порта.

*Сетевая разведка*

Сбор информации о сети с ипользованием общедоступных приложений называется сетевой разведкой. Перед атакой злоумышленник занимается сетевой разведкой, то есть собирает всю возможную информацию о сети. Разведка проводится в виде DNS запросов, сканирования портов и pingsweep (эхо-тестирование). Благодаря DNS запросам злоумышленник определяет кто владеет каким доменом и какие адреса им присвоены. С помощью эхо-тестирования адресов можно увидеть, какие хосты работают в среде. После получения списка хостов можно просканировать порты, чтобы получить весь список услуг, которые поддерживают эти хосты. Проанализировав все характеристики приложений можно получить информацию, достаточную для взлома.

*Злоупотребление доверием*

Суть этого метода в использовании отношений доверия, которые существуют в сети. Рассмотрим этот метод на примере периферийной части корпоративной сети. DNS, SMTP и HTTP серверы чаще всего распологаются именно в этом сегменте сети. Взлом одного из серверов приведет автоматически к взлому других, так как они находятся в одной сегменте сети. Потому что эти сервера доверяют друг другу. Другим примером является межсетевой экран. Система с внешней стороны межсетевого экрана имеет отношения доверия с системой, которая установленая с его внутренней стороны. При взломе внешней системы, злоумышленник автоматически

получает доступ во внутреннюю систему, которая защищена межсетевым экраном, благодаря их отношениям доверия.

*Переадресация портов*

Одной из разновидностей злоупотребления доверием является переадресация портов. Представим, что есть межсетевой экран и два хоста по обе его стороны. Когда взломаны внешние хосты, то их могут использовать для передачи через межсетевой экран трафика, который не был бы пропущен в обычном случае. При этом нет ни одного нарушения правила на межсетевом экране. В итоге внешний хост в результате переадресации получает доступ к внутреннему хосту.

*Несанкционированный доступ*

Несанкционированный доступ – это не отдельный тип атак. Почти все сетевые атаки используются для получения этого доступа. Перед тем как получить логин telnet, злоумышленник должен получить подсказку на своей системе. При подключении к портам telnet появляется сообщение о том, что нужна авторизация для пользования этим ресурсом. Если после этого злоумышленник продолжит любые попытки получить доступ, то все они будут считать “несанкционированными”.

*Вирусы и приложения типа "троянский конь"*

Рабочие машины пользователя очень часто являются уязвимыми для различных троянов и вирусов. Вирус – это вредоносная программа, которая внедряется в другую программу для выполнения различных функций на рабочей машине. Примеров вирусов множество, одни стирают какие-то файлы, другие добавляют или просто заражают все файлы. Троянский конь, в отличие от вирусов, - это полноценная программа, а не вставка. Он выглядит как обычное полезное приложение, хотя на деле несёт совершенно другую роль. Троян позволяет манипулировать удаленным компьютером, получать

чужие интернет-аккаунты по почте, вести логи на удаленном компьютере и многое другое. Трояны делятся на несколько типов (Mail Senders, BackDoor, Log Writers и др.).

## 1.4 Анализ существующих решений и подходов к организации защищенных каналов связи

Основной способ организации защищенных каналов связи – это VPN (*Virtual Private Network)*. Рассмотрим же его подробнее.

### *1.4.1 VPN сети*

Виртуальная частная сеть ( VirtualPrivateNetwork, VPN) – является логической сетью, которая создаётся поверх другой сети, к примеру Internet. С учётом того что коммуникации происходят по обычным публичным сетям, где используются небезопасные протоколы, то благодаря шифрованию создаются закрытые каналы обмена информацией. Благодаря VPN можно свободно объеденить несколько офисов компании в единую сеть, где будут использоваться неподконтрольные каналы для связи между ними.

У VPN есть много свойств выделенной линии, но создаётся она в рамках общей сети, напримет Internet. Благодаря методам туннелирования, все пакеты данных передаются через общую сеть, как по обычному соединению. Образуется так называемый туннель между парой отправитель-получатель. Туннель – это безопасное соединение, которое позволяет инкапсулировать данные одного протокола в пакеты другого.

Благодаря туннелированию возможна передача пакетов одного протокола в логической среде, которая использует другой протокол. Поэтому решилась проблема взаимодействия разнотипных сетей. Благодаря этому преодоление разных внешних протоколов и схем ардесации стало возможным, что еще и обеспечивает целостность передаваемых данных.

Любая компания со своей сетевой инфраструктурой может с помощью аппаратного или программного обеспечения быть подготовлена к использованию VPN. Прокладка кабеля напоминает всю организацию виртуальной сети. Соединение между пользователем и конечным устройством туннеля устанавливается по протоколу PPP.

Самый распростанённый метод для создания VPN – это инкапсуляция сетевых протоколов в PPP и дальнейшая инкапсуляция созданных пакетов в протокол туннелирования. Чаще всего в роли протокола туннелирования выступает IP. Этот подход называют туннелирование второго уровня.

Альтернативным подходом является инкапсуляция сетевого протокола сразу в протокол туннелирования. И этот метод называется туннелирование третьего уровня.

Методика остаётся практически такой же, вне зависимости от того, какие протоколы используются или какие цели преследуются. Для соединения с удаленным узлом чаще всего используется один протокол, а для инкапсуляции данных для передачи через туннель, другой.

### *1.4.2 Классификация сетей VPN*

Множество компаний строят свою стратегию по использованию Internet в качестве основной среды передачи информации, даже если она является важной. И всё это благодаря технологии VPN.

Есть много признаков для классификации VPN. Это наиболее используемые:

- «рабочий» уровень модели OSI;
- архитектура технического решения VPN;
- способ технической реализации VPN.

*Классификация VPN по “рабочему” уровню модели OSI*

Защищенный канал – технология безопасной передачи данных по незащищенный сети. По некоторому виртуальному пути обеспечивается защита данных, которая происходит между двумя узлами сети.

На разных уровнях модели OSI можно построить защищенный канал.


От выбранного уровня модели OSI зависит весь функционал реализуемой VPN и совместимость с другими средствами защиты, поэтому классификация VPN по “рабочему” уровню модели OSI прдставляет огромный интерес.

По признаку «рабочего» уровня модели OSI различают следующие группы VPN:

- VPN канального уровня

На канальном уровне модели OSI есть возможность обеспечить инкапсуляцию разных видов трафика третьего уровня, а также построения виртуальных туннелей по типу: “точка- точка”. Эти средства использует VPN канального уровня. К ним относятся продукты, где используется протокол PPTP (Point-to-Point Tunneling Protocol) и L2F (Layer 2 Forwarding) и стандарт L2TP (Layer 2 Tunneling Protocol).

- VPN сетевого уровня;

На сетевом уровне происходит инкапсуляция IP в IP. Самым известным и часто используемымы протоколом этого уровня являеся – IPSEC (IPSecurity). Он пердназначен для туннелирования, шифрования и аутентификации IP-пакетов.

Протокол IKE (Internet Key Exchange) очень тесно связан с протоколом IPSEC. Он решает задачи по управлению и безопасности в обмене ключами между устройствами. IKE устанавливает защищенное соединение и автоматизирует обмен ключами, а IPSEC только кодирует пакеты. IKE значительно повышает безопасность передаваемой информации за счёт возможности смены ключа для уже установленного соединения.

- VPN сеансового уровня.

Есть другой подход, который реализуется на транспортном уровне модели OSI. Для каждого сокета по отдельности ретранслируется трафик из защищенной сети в незащищенную сеть Internet. Сокет идентифицируется комбинацией TCP-соединения и заданным UDP портом.

Шифрование, а значит и защиту иформации, передающейся по туннелю обеспечивает криптографический протокол TLS (Transport Layer Security). Также используется протокол SOCKS для аутентифицированного прохода межсетевого экрана.

*Классификация VPN по архитектуре технического решения*

По архитектуре технического решения принято выделять три основных вида виртуальных частных сетей:

- внутрикорпоративные VPN (Intranet VPN)

Для обеспечения защищенного взаимодействи между предприятиями или подразделениями внутри предприятия, которые объеденены корпоративными сетями связи, используются внутрикорпоративные сети VPN

- VPN с удаленным доступом (Remote Access VPN)

Для обеспечения защищенного удаленного доступа к корпоративным ресурсам удалённым сотрудникам компании используют VPN с удаленным доступом.

- межкорпоративные VPN (Extranet VPN).

Для обеспечения защищенного обмена информацией с различными поставщиками, партнёрами по бизнесу, заказчиаками, клиентами и другими используют межкорпоративные сети VPN. Благодаря Extranet можно получить прямой доступ из сети одной компании к сети другой.

*Классификация VPN по способу технической реализации*

В зависимости от типа VPN-устройств определяются характеристики и конфигурации виртуальной сети.

Различают VPN по способу технической реализации:

- VPN на основе маршрутизаторов

Для создания защищенных каналов предпологается применение маршрутизаторов. Через маршрутизатор проходит вся информация из локальной сети, поэтому вполне логично повесить именно на маршрутизатор задачи шифрования. У компани CiscoSystems, например, есть устройства, которые подходят для этих задач.

- VPN на основе межсетевых экранов

Множество межсетевых экранов поддерживают функции шифрования и туннелированя данных. Подобное решение подходит только для небольших сетей, где передаются небольшие объемы информации. Главный недостаток этого метода заключается в сильной зависимости произовительности от аппаратного обеспечения, на котором работает межсетевой экран. Также недостатком явшяется довольно высокая стоимость решения в пересчёте на одно рабочее место.

- VPN на основе программных решений

Существует программная возможность для реализации VPN. Но эти продукты уступают по производительности специальным устройствам. При удалённом доступе требования для полосы пропускания очень малы, поэтому программные продукты легко обеспечивают производительность, которой хватает для удаленного доступа. Главным достоинством программных

решений является удобство в применении, относительная дешевизна и гибкость.

- VPN на основе специализированных аппаратных средств со встроенными шифропроцессорами

Шифрование в таком методе осуществляется специализированными микросхемами, поэтому обеспечена очень высокая производительность. Такие VPN-устройства обеспечивают высокий уровень защиты. Из недостатков можно выделить, что их цена довольно дорогая.

## 1.5 Методы шифрования информации

Шифрование – главное средство для обеспечения безопасности. Благодаря нему происходит сокрытие инормации от неавторизованных пользователей и, наоборот, предоставление авторизованным доступа к ней. Пользователь у которого есть подходящий ключ и для расшифровки информации называется авторизованным.

Главная цель любого шифрования – это как можно больше усложнить доступ к информации для неавторизованных пользователей, даже если они знают алгоритм, котрый используется для шифрования и у них есть зашифрованный текст. Целостность и секретность информации в безопасности, пока у неавторизованного пользователя нет ключа.

Шифрование обеспечивает три важных состояния безопасности информации:

- ***Конфиденциальность.***

Шифрование нужно для того, чтобы сокрыть информацию при передаче или хранении.

- ***Целостность.***

Шифрование нужно для снижения угрозы изменения самой информации в моменты её передачи или хранения.

- ***Идентифицируемость.***

Шифрование нужно для аутентификации источника иформации и для того, чтобы предотвращать отказ отправителя информации от того, что данные отправлял именно он.

Существуют два основных метода для шифрования информации: симметричное шифрование (также называемое шифрованием секретным ключом) и асимметричное шифрование (также называемое шифрованием с открытым ключом).

### *1.5.1 Симметричное шифрование*

Симметричным шифрованием называется шифрование на секретном ключе, потому что для шифрования и дешифрования используется одинаковый ключ. Это значит, что отправитель и получатель информации имеют один и тот же ключ.


Это шифрование обеспечивает конфиденциальность информации в зашифрованном состоянии, а расшифровать её могут только те пользователи, которые обладают ключом. Это очень лёгкий и быстрореализуемый метод шифрования.

*Схема работы:*

- Генерация ключа.

Выбирается ключ шифрования *d* и алгоритм *E, D* (функции шифрования и расшифрования).

- Шифрование и передача сообщения

Первый пользователь шифрует информацию с помощью ключа *d* и передаёт второму пользователю полученный шифротекст *c (E (m,d) =c).*

- Дешифрование сообщения

Пользователь два, с помощью того же ключа *d*, расшифровывает шифротекст *c (D (c,d) =m).*

*Симметричные криптографические алгоритмы:*

- Простая перестановка

Является одним из самых простых методов шифрования. Всё сообщение записывается в таблицу по столбцам. Когда текст записан колонками, путём чтения текста по строкам образуется шифровка. Для того, чтобы использовать этот способ двум пользователям (получатель и отправитель), нужно договориться о размере таблицы, что и является общим ключом простой перестановки.

- Одиночная перестановка по ключу

Этот метод очень похож на предыдущий, но главным его отличием является то, что колонки таблицы переставляются по ключевому слову, набору чисел или фразе длинной в строку всей таблицы.

- Двойная перестановка

Для того, чтобы повысить скрытность, можно повторно зашифровать сообщение, которое уже было зашифровано. В этом и заключается суть двойной перестановки. Чтобы это было возможно, размер второй таблицы подбирается так, чтобы длины её столбцов и строк отличались от длин в первой таблице. Самый лучший вариант, если они будут взаимно простыми. Помимо этого, во второй таблице можно переставлять строки, когда в первой столбцы. Также таблицу можно заполнять по спирали, зигзагом, змейкой или любым другим способом.

- Перестановка “Магическиий квадрат”

Магический квадрат – это квадратные таблицы со вписанными в их ячейки последовательно натуральными числами от 1. В сумме по каждой

строке, диагонали или столбцу должно быть одно и тоже число. Суть шифрования магичесским квадратом заключается во вписывании букв фразы последовательно в квадрат, где позиция буквы в предложении соответствует порядковому номеру, а в пустые клетки ставят точку. После проделанной шифровации, текст записывается в строку (слева направо).

*Блочный и поточный шифр.*

Также шифры могут отличаться структурой шифруемой информации, соответственно подразделяются на блочный шифр и поточный.

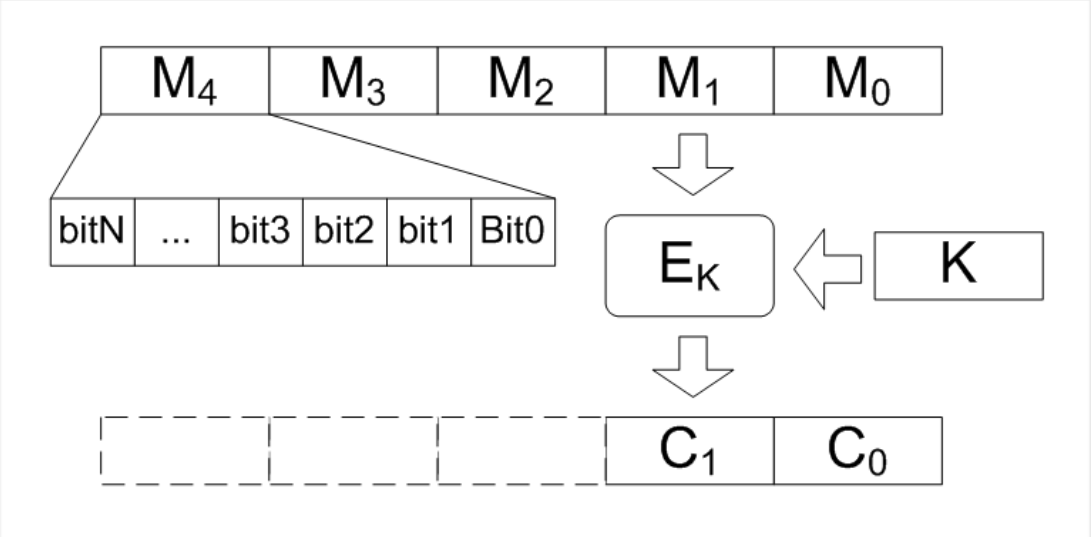
*Блочный шифр* является разновидностью симметричного шифра, который выполняет операции над блоками, то есть определёнными порциями данных фиксированного размера. Чаще всего размер блока составляет 64 или 128 бит. Порой используются и другие значенич блоков. Блочный шифр занимается преобразованием определенного блока текста в один шифротекст, вне зависимости от того какие данные были зашифрованы до этого.

Из-за того, что размер шифруемого текста не всегда кратен размеру самого блока, появляется проблема дополнения. Она имеет несколько решений. Один из них - это передача в незашифрованном виде полезной части зашифрованных данных, а после расшифровки - лишние байты просто откидывать. Второй способ применяется чаще.

Представим что алгоритм оперирует восьмибайтовыми блоками, а, например, в последнем блоке всего три байта данных. Тогда все неиспользуемые байты, кроме последнего нужно заполнить любыми значениями, а в последнем байте записать число, которое равно тому количеству неиспользуемых байт. При расшифровке тогда нужно отбросить с конца то количество байт, которое указано в последнем байте последнего блока. Но тогда возможна сложность. Если исходное сообщение имело длину, кратную размеру блока, то требуется добавить 0 байт, а последний байт должен содержать число байт дополнения. Эту проблему решили добавлением дополнительного блока, последний байт которого будет

содержать размер блока и, следовательно, весь дополнительный блок будет отброшен при расшифровки информации.

Блочный шифр - это совокупность двух алгоритмов : шифрования и дешифрования. Их оба можно представить в виде функций. Тогда функция шифрования E на входе будет получать блок данных M, размер которого будет n бит и ключ K, размер которого будет k бит. А на выходе получается блок шифротекста C, размер которого n бит.

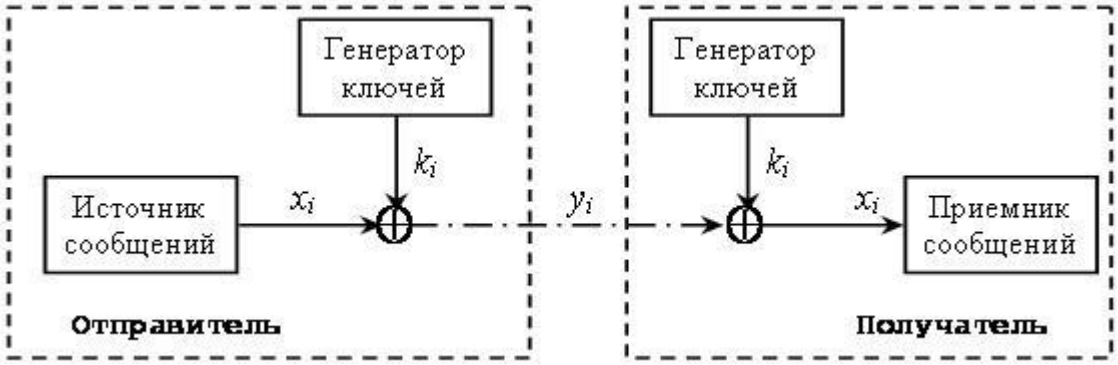



Общая схема работы блочного шифра

Главным достоинством блочных шифров является подобие процедур дешифрования и шифрования, отличие которых заключаются лишь в порядке выполнения действий. Это сильно облегчает создание устройств шифрования, потому что позволяет использовать одни и те же блоки в цепях шифрования и дешифрования.

*Поточный шифр* является разновидностью симметричного шифра, который выполняет преобразования входных данных по одному биту (байту) за операцию. При использовании поточного алгоритма шифрования нет надобности разбивать сообщение на блоке, поэтому он может работать в реальном времени. Значит, при передаче потока символов, каждый из них шифруется и передаётся сразу.

Работа поточного шифра представлена на рисунке ниже




Генератор ключей выдаёт поток битов $s_i$. Они будут использоваться в качестве гаммы. Источник генерирует биты открытого текста – $x_i$. Открытый текст складывается с гаммой по модулю два и получаются биты зашифрованного сообщения $y_i$.

$$y_i = x_i \oplus k_i, i = 1, 2, \ldots, n$$

Для того, чтобы из шифроекста $y_1 \ldots y_n$ получить сообщение $x_1 \ldots x_n$, надо сгенерировать такую же ключевую последовательность $k_1 \ldots k_n$, что и при шифрации, и использовать для дешифрации формулу, приведенную ниже.

$$x_i = y_i \oplus k_i, \ i = 1, 2, \ldots, n$$

*Виды симметричных шифров:*

- блочные шифры
  - AES
  - ГОСТ 28147-89
  - DES (Data Encryption Standard)
  - 3DES
  - RC2
  - RC5
  - Blowfish

- Twofish
- NUSH
- IDEA
- CAST
- CRAB
- 3-WAY
- Khufu иKhafre

- потоковые шифры
  - RC4
  - SEAL
  - WAKE (World Auto Key Encryption algorithm)

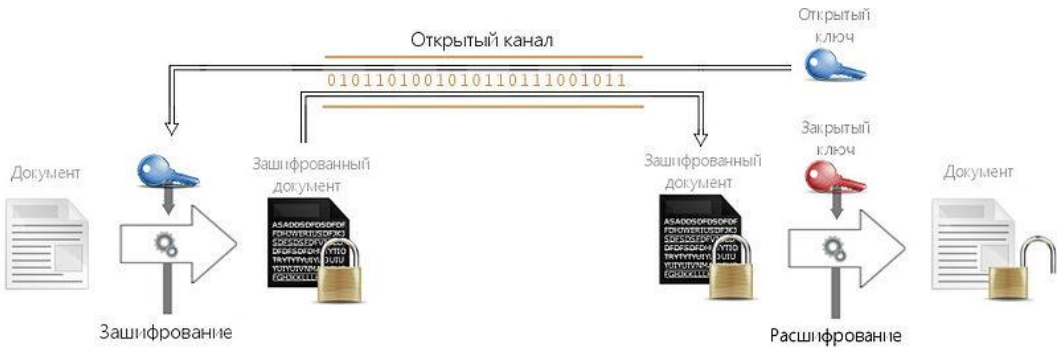
*Достоинства и недостатки:*

- Достоинства:
  - Большая скорость.
  - Легкость в реализации.
  - Для соизмеримой стойкости меньшая длина ключа.
  - Хорошо изучена.
- Недостатки:
  - Высокая сложность в управлении ключами в большой сети. Ключи, которые надо передавать, хранить, создавать и уничтожать в сети возрастают с огромной скоростью. Например, для 10 абоненов надо 45 ключей, тогда для 1000 надо будет 499500.
  - Сложности при обмене ключами. Требует наличие какого-нибудь секретного канала для передачи ключа обеим сторонам.

### *1.5.2 Асимметричное шифрование*

Отправитель и получатель имеют ключ. Эти ключи связаны друг с другом, но в тоже время разные. Допустим, если сообщение зашифровано с помощью ключа K1, то для дешифрации сообщения понадобиться ключ K2. И наоборот. Один из этих ключей называется – открытым, а другой – секретным.

Секретный ключ хранится в тайне владельцем пары ключей, а открытый передаётся вместе с сообщением в открытом виде, поскольку другой ключ вычислить невозможно.




Шифрование выполняется с открытым ключом, тогда дешифрация информации возможна только при наличии секретного ключа.

*Схема работы:*

- Генерация пары ключей

Первый пользователь выбирает алгоритм *( E, D )*, также пару открытый, закрытый ключи – *( e, d )*. И по открытому каналу отсылает открытый ключ *e* второму пользователю.

- Шифрование и передача сообщения

Второй пользователь производит шифрование информации с использованием открытого ключа первого *e*.

*E (m, e) =c*

И передает первому пользователю шифротекст *c*.

- Дешифрование сообщения

Первый пользователь с использованием закрытого ключа *d*, дешифрует шифротекст *c*.

*D (c, d) =m*

Чтобы наладить канал связи в обе стороны, надо проделать первые две операции на обеих сторонах. Для того, чтобы каждый знал свой открытый, закрытый ключи и открытый ключ собеседника. Закрытый ключ не передаётся по незащищенному каналу, поэтому остаётся засекреченным.

*Виды асимметричных шифров:*

- RSA (Rivest-Shamir-Adleman)
- DSA (Digital Signature Algorithm)
- Elgamal
- Diffie-Hellman
- ECDSA (Elliptic Curve Digital Signature Algorithm)
- ГОСТ Р 34.10-2001
- Rabin
- Luc
- McEliece
- Williams System

*Достоинства и недостатки:*

- Достоинства:
  - Отсутствует нужда в передаче секретного ключа по надёжному каналу.
  - Ключ, который нужно держать в секрете, знает только одна сторона (в симметричном шифровании этот ключ был известен обеим сторонам).

- Пару открытого и секретного ключа можно оставлять неизменным значительное время (при симметричном шифровании обновлять ключи нужно после каждой передачи).
- Отсутствие сильного роста числа ключей в больших сетях по сравнению с асимметричным шифрованием.

- Недостатки:
  - Тяжелее внести какие-либо изменения в алгоритм.
  - Увеличена длина ключей.
  - Шифрация и дешифрация с использованием секертного и открытого ключа происходит на порядок медленее, чем шифрация и дешифрация у тоже сообщения, но при использовании симметричного алгоритма.
  - Нужно намного больше вычислительных ресурсов, поэтому на практике сочетают использование ассиметричных систем и других алгортимов.

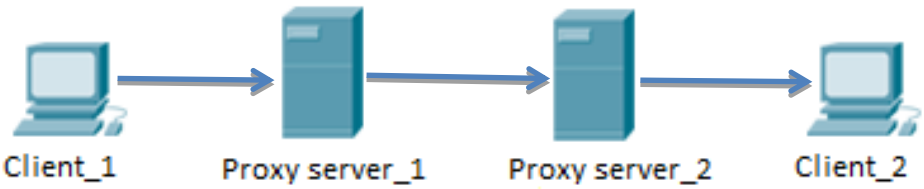
## 1.6 Выводы

В этом разделе был проведён анализ основных угроз безопасности компьютерной среды, рассмотретно множество видов сетевых атак. Проанализировали различные решения и походы к организации защищенного канала. И узнали о различных методах шифрования информации.

# Раздел 2. Технологическая и специальная часть.

## 2.1 Разработка архитектуры защищенного канала на базе стека протоколов TCP/IP

Схема защищенного канала связи




Защищенный состоит из:

- Client_1
- Proxy server_1
- Proxy server_2
- Client_2

Рассмотрим подробнее каждый из составляющих:

1. Client_1

На машине Client_1 установлено приложение, которое шлет данные на указанный порт и адрес Proxyserver_1

2. Proxy server _1

Proxysever_1 принимает данные от client_1, производит шифрацию полученного и шлёт зашифрованные данные на Proxy_server_2.

3. Proxy server_2

Proxyserver_2 получает данные от Proxy_sever_1, производит дешифрацию и отправляет их Client_2.

4. Client_2

Client_2 получает расшифрованные данные от Proxyserver_2.

На Client_1 установленна программа, в соответствии с которой все данные с порта передаются на IP-адрес (и другой порт) прокси-сервера.

На Proxyserver_1 и Proxyserver_2 установлены proxy-сервера.

Proxyserver – промежуточный сервер, который позволяет клиентам выполнять косвенные запросы к другим службам. Сначала происходит подключение клиента к proxyserver и подача запроса на определенный ресурс, который распологается на другом сервере. Proxyserver подключается к вышеуказанному серверу и получает ресурс. Иногда ответ сервера (или запрос клиента) может быть изменён proxyserver в определённых целях. Proxyserver защищает компьютеры клиентов от различных сетевых атак и сохраняет анонимность клиента.

Основные цели, для которых применяется proxyserver:

Чаще всего прокси-серверы применяются для следующих целей:

- Для обеспечения доступа в интернет с компьютеров локальной сети
- Возможность кэширования. Если клиент часто общается с одним внешним ресурсом, то можно создать копию ресурса на proxyserver. Это значительно снизит нагрузку на канал, а следовательно увеличит скорость полученииия нужной информации.
- Возможность сжатия данных. Proxyserver загружает информацию, а клиенту передаёт уже в сжатом виде. Такие proxyserver используются для экономии сетевого трафика клиента или компании.
- Защита от внешнего доступа. Proxyserver можно настроить так, что все локальные компьютеры сети будут обращаться к внешним ресурсам

только через proxyserver. Соответственно внешние компьюетеры не получат доступа к локальным вообще, так как виден только Proxyserver.

- Возможность ограничивать доступ из локальной сети на внешний ресуерс. Можно устанавливать квоты на трафик, фильтровать рекламу, запретить доступ к различным внешним ресурсам (разным веб-сайтам), то есть ограничивать использование интернета.
- Анонимный доступ к ресурсам.У proxyserver есть возможность скрыть сведения об источнике запроса или пользователе. В этом случае видна лишь информация о самом proxyserver. Также есть искажающие proxyserver, которые передаёт ложную информацию о пользователе серверу.
- Возможность обхода ограничений доступа ресурса. Proxyserver очень часто используется в странах, где доступ к разным ресурса ограничен законодательством.

Виды proxyserver:

- Прозрачный proxy – схема связи, при котором трафик перенаправляется на proxyserver средствами маршрутизатора (то есть неявно). А значит, что у клиента есть возможность использовать proxyserver без каких-либо настроек с его стороны, так как всё настроено на маршрутизаторе.
- Обратный proxy – proxyserver, который ретранслирует все запросы клиентов из сети на один или несколько серверов, которые расположены во внутренней сети. Это используется для повышения безопасности и балансировки сетевой нагрузки.

Принцип работы:

Компьютер клиента настроен программой так, что все его сетевые соединения по протоколу совершаются не на IP- адрес сервера, а на ip-адрес и порт заданного proxyserver.

Когда возникает необходимость у клиента обратиться к какому-либо ресурсу, то открывается сетевое соединение с proxyserver и происходит

запрос на ресурс. На стороне клиента разница незаметна, как если бы он обращался просто к самому ресурсу.

Когда proxyserver распознал данные запроса, проверил их на корректность и разрешение для клиента, он, не разрывая соединения, создаёт новое с нужным ресурсом и подаёт тот же запрос. Когда proxyserver получает данные, то он переправляет их клиенту.

Поэтому proxyserver является полнофункциональным сервером для каждого протокола, который поддерживает. Имеет полный контроль над всеми деталями этого протокола, а также есть возможность применять различные политики, которые заданы администратором.

Proxyserver – это один из самых популярных способов для выхода в интернет из локальных сетей в различных предприятиях и организациях. И этому способствуют такие факторы:

- Основной протокол, используемый в интернете – HTTP. В его стандартах описана поддержка работы через proxy.
- Proxy поддерживается огромных количеством браузеров и операционных систем.
- Контроль доступа и учёт трафика каждого пользователя локальной сети.
- Фильтрация трафика (вирусы).
- Proxyserver достаточно мнинимальных прав в любой операционной системе с поддержкой стека протоколов TPC/IP для корректной работы.
- Множество программ используют различные собственные протоколы. В proxy они могут использовать HTTP (как альтернативу) или же SOCKS-proxy (как универсальных proxy, который подходит почти для всех протоколов).
- Повышение безопасности всей локальной сети, благодаря proxyserver.

Для написания proxysever была выбрана библиотека Boost.Asio, потому что её, полжалуй, можно назвать лучшей реализацией среди существующих. Asio – кроссплатформенная библиотека, предназначенная для сетевого взаимодействия и других видов низкоуровневого ввода-вывода. Включает в себя поддержку сокетов, таймеров, последовательных портов, файлов и ссылки Windows. Библиотека написана Крисом Колхофом.

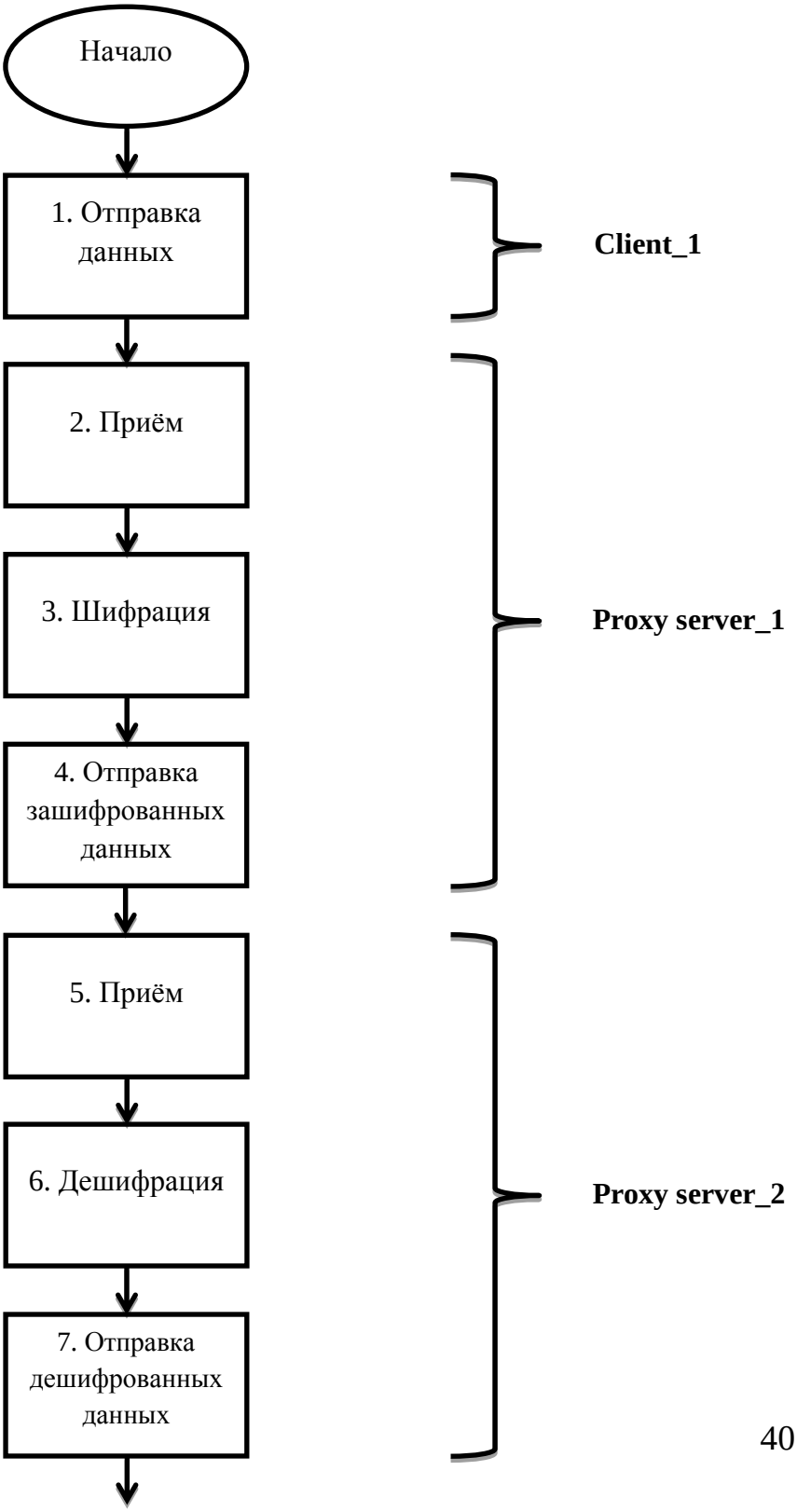
К основным возможностям Boost.Asio можно отнести следующие:

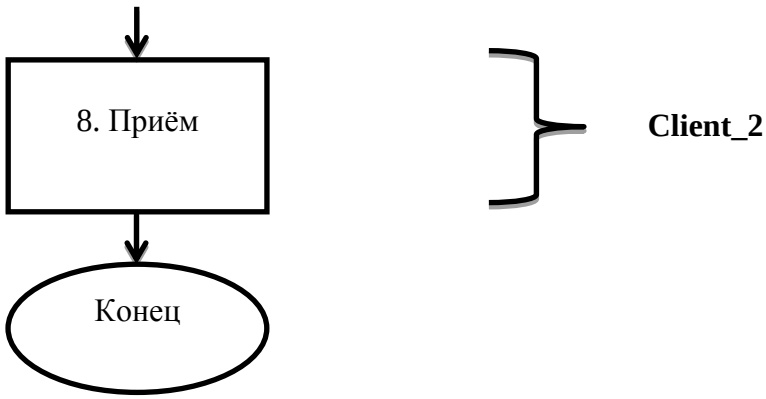
- возможность написания кросс-платформенного сетевого кода, работающего на большом количестве платформ - Windows, практически все Unix-подобные ОС, Tru64
- поддержка как IPv4, так и IPv6
- работа с TCP & UDP
- поддержка выполнения асинхронных операций
- возможность использования интерфейсов, совместимых с std::iostream
- поддержка SSL
- поддержка отложенных операций (таймеры)

Самым главным преимуществом Boost.Asio является то, что на каждой платформе реализуется наиболее эффективная стратегия работы (epoll на Linux 2.6,kqueue на FreeBSD/MacOSX, Overlapped IO на MS Windows), и то, что библиотека позволяет использовать разные стратегии — синхронная и асинхронная работа с сокетами, использование потокового ввода-вывода, совместимого с std::iostream. И эти стратегии можно смешивать, например, принимать соединения используя асинхронную обработку событий, а затем, запускать нить, которая будет использовать синхронный ввод-вывод данных.

## 2.2 Разработка алгоритма функционирования системы защищенного канала

Ниже представлен алгоритм защищенного канала связи:






1. Отправка первым клиентом данных на первый proxyserver.
2. Первый proxyserver принимает данные, присланные от клиента.
3. Первый proxyserver производит шифрацию присланных данных.
4. Первый proxyserver отправляет зашифрованные данные на ip-адрес и порт второго proxysever.
5. Второй proxyserver принимает данные от первого.
6. Второй proxyserver дешифрует полученные данные.
7. Второй proxyserver отправляет расшифрованные данные второму клиенту.
8. Второй клиент принимает данные.

## 2.3 Выбор инструментальных средств для реализации защищенного канала

Основной целью перед началом работы – это выбор языка программирования, на котором всё будет реализовано. Мой выбор пал на три популярных и удобных для работы в нашей области языка: C++, Python, Java. Ниже кратко узнаем каждый из этих языков, познакомимся с их преимуществами и недостатками и выберем какой больше всего подходит для работы.

### *2.3.1 Python*

Python - интерпретируемый, объектно-ориентированный язык программирования высокого уровня. Высокоуровневые структуры, которые встроены в него, вместе с динамическими типизацией и связыванием делают язык привлекательным для быстрой и качественной разработки приложений. Также его можно использовать в качестве сценарного языка для связи программных компонентов. Синтаксис Python очень прост в изучении, в нем отводится огромное значение читаемости кода, а это уменьшает затраты на создание программных продуктов. Python поощряет повторное использование кода, также есть поддержка модулей и пакетов. Большая библиотека и интерпритатор Python доступны бесплатно для платформ в виде исходных кодов и свободно могут распространяться.

*Достоинства языка*

Одним из главных достоинств Python является реализация интепретатора почти на всех операционных системах и платформах. Первым языком с этой функцией был C, но его типы могли занимать различное количество памяти на разных машинах, поэтому качество переносимости программ уменьшалось. У Python нету этого недостатка.

Следующее достоинство – это расширяемость языка. Сам язык был задуман как расширяемый, поэтому этому придаётся огромное значение. Имеется возможнсть совершенствования языка всеми желающими. Есть

возможность написать на своё дополнение к Python на C, скомпилировать и получим “расширенный” интерпретатор, у которого будут новые возможности. Или можно вставить исходный код в программу и использовать как встроенную оболочку.

Следующая важная черта – это наличие огромного количества подключаемых модулей, которые обеспеивают дополнительные возможности. Модули пишутся на C или на Python и могут быть созданы квалифицированными специалистами. В качестве примера можно привести модули:

- NumericalPython - расширенные математические возможности (манипуляции с матрицами и векторами);
- Tkinter – использование графического интерфейса GUI для построения приложений.
- OpenGL - использование библиотеки графического моделирования двух- и трехмерных объектов

*Недостатки языка*

Главным и, наверно, единственным недостатком Python является невысокая скорость выполнения программы из-за её интепретируемости. Зато это окупается, когда нужно написать программу не очень критичную к скорости выполнения.

### *2.3.2 Java*

Java – высокоуровневый объектно-ориентированный язык программирования, который был разработан компанией SunMicrosystems. В настоящее время возможности этого языка сильно увеличились. Сейчас Java это не отдельный язык программирования, а целое семейство технологий.

*Достоинства языка*

- *Безопасность.*

Главный принцип в разработке языка Java состоит в обеспечении защиты от несанкционированного доступа. Программы, написанные на языке Java, не могут получить доступ к системным ресурсам или вызывать глобальные функции, поэтому обеспечивается достаточно высокий уровень безопасности, в сравнение с другими языками.

- *Объектная ориентированность*

Java – это чисто объектно-ориентированный язык, его объектная парадигма не ослабляет возможности языка, как, например, в C++. В Java используются более развитые классы и методы.

- *Надежность*

Язык Java способствует обнаружению различных ошибок уже на ранних стадиях разработки приложения. В ней нету строгой типизации, что приводит к отсутствию множества ошибок. В других языках программирования программисту нужно самому распределять память, которую использует программа, но часто просто забывают освобождать ту память, которая всё еще используется какой-либо частью приложения. Java почти полностью снимает все эти проблемы, благодаря сборщику мусора, который используется для освобождения незанятой памяти.

Сборка мусора – это одна из главных особенностей языка Java, которая предназначена для удаления всех ненужных объектов из памяти. Благодаря

этой системе программист может не так внимательно следить за использованием памяти.

Когда создаются объекты в Java можно просто не беспокоиться о том, что система сборки мусора позаботится об их удалении. Объект будет полностью удалён из памяти, когда на него не останется ни одной ссылки из других объектов.

Процесс сборки мусора имеет очень низкий приоритет, поэтому почти не отнимает ресурсов у самих приложений.

Одной из наиболее мощных и одновременно опасных черт C++ является указатели или адреса в памяти. Неправилная работа с указателями является причиной множеств ошибок. В Java нету возможности работать непосредственно с указателями, хотя дескрипторы объектов и реализованы в виде указателей. Нельзя обратиться к произвольному адресу в памяти или же преобразовать целое число в указатель.

- *Интерактивность*

Java полностью удовлетворяет потребность в создании интерактивных сетевых программ. В нём есть решения, которые позволяют писать код, выполняющий одновременно множество различных функций, а также отслеживает что и когда должно произойти. Благодаря простым в обращении подпроцессам Java даёт возможность реализировать в программе конкретное поведение и не нужно отвлекаться на глобальные обработки событий.

- *Независимость от архитектуры ЭВМ*

Переносимость кода – одна из самых актуальных проблем. В Java же наложены жесткие требования на язык. Зато, однажды написанная программа, всегда будет запускаться в любом месте и в любое время ( при наличии виртуальной Java-машины).

- *Интерпретация плюс высокая производительность*

Возможность Java исполнять код на любой платформе достигается тем, что её программы транслируются на некое промежуточное представление, которые называется байт-кодом или bytecode. Байт-код же может интерпретироваться везде, где есть среда Java. Множество других систем, которые пытались обеспечить независимость от платформы, имели один важный недостаток - потеря производительности (например Perl). Байт-код с лёгкостью переводится в машинные коды, даже несмотря на то, что использует интерпретатор. Самое главное, что достигается высокая производительность.

- *Простота изучения*

Язык Java намного проще в изучении, чем другие языки программирования (например всё тот же C++). Главным отличием Java от C++ в синтаксисе – это отсутствие заголовочных файлов, препроцесоора, директивы define и операторов typeded.

*Недостатки языка*

Чаще всего к недостаткам идеи виртуальной машины относят то, что байт-код снижает производительность приложений и алгоритов, которые реализованы на Java.

### *2.3.3 C++*

C++ — компилируемый статически типизированный язык программирования общего назначения. Поддерживает разные парадигмы программирования, но наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования.

Название «C++» происходит от Си , в котором унарный оператор ++ обозначает инкремент переменной.

В 1990-х годах язык стал одним из наиболее широко применяемых языков программирования общего назначения. При создании C++ стремились сохранить совместимость с языком Си.

*Достоинства языка*

- Объектно-ориентированное программирование в C++ (подход, при котором основными концепциями являются понятия объектов и классов).
- Перегрузки операторов. В C++ можно переопределять семантику почти всех операторов и служебных символов.
- Создание алгоритмов для разных типов данных на основе использования шаблонов.
- В C++ появилось namespace - пространство имен. Для того, чтобы глобальное пространство имен распилить на независимые зоны во избежание конфликтов имен переменных и функций.
- Завязывание языка на типы. Перегрузки функций.Когда объявляются несколько функции с одним и тем же именем, но количество аргументов и их типы могут быть различны, также и возвращаемое значение может быть различного типа. Это делает написание программ более удобным. Так как нет необходимости давать различным функциям, делающим примерно одно и тоже, но с аргументами различных типов, разные имена и запоминать их.

- Возможность имитации расширения языка для поддержки парадигм, которые не поддерживаются компиляторами напрямую.
- Кроссплатформенность языка (функционирование на разных операционных системах и аппаратных платформах).
- Работа на низком уровне с адресами и памятью.
- Выскоая совместимость с языком С.

*Недостатки языка*

- Основным и главным недостатком является сложность и избыточность С++.
- Отсутствие сборщика мусора.
- Невозможность компиляции шаблонов.

### *2.3.4 Выбор языка реализации*

На основе анализа всех достоинств и недостатков трёх языков, что были выбраны ранее плюс наличие у языка C++ одной из лучших библиотек для работы с сетью Boost.Asio и знание на приличном уровне этого языка, было принято решение взять за основу реализацию защищенного канала на языке C++.

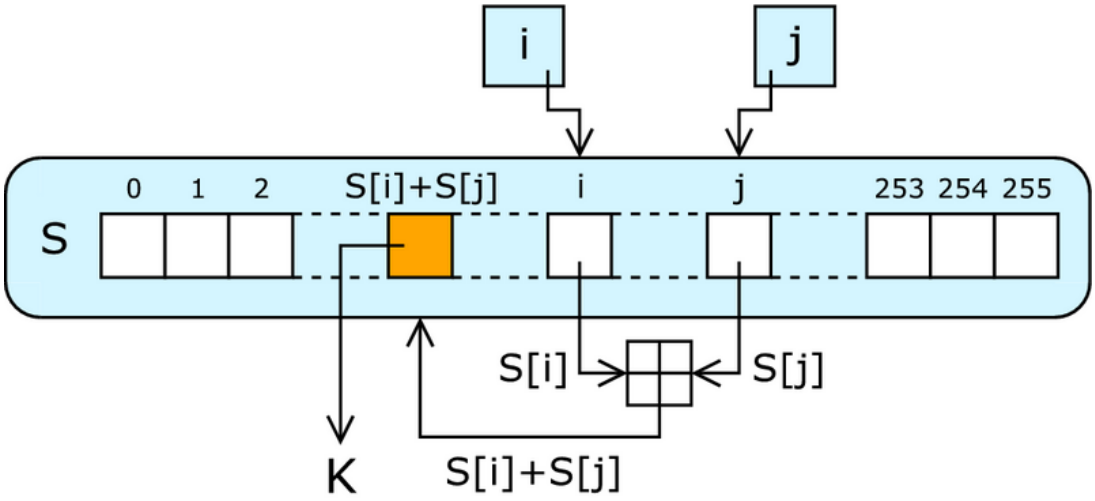
### *2.3.5 Выбор алгоритма шифрования*

Здесь хотелось выбрать достаточно стойкий шифр с высокой скоростью работы и несложной реализацией. Поэтому выбора пал на потоковый шифр RC4.

*RC4*

RC4 – потоковый шифр, а это значит, что каждый символ текста шифруется в зависимости от его положения в тексте и ключа.

Алгоритм RC4 построен на основе параматризированного ключом генератора псевдослучайных битов с равномерным распределением. Длина ключа может быть от 40 до 2048 бит.




Генератор ключевого потока RC4

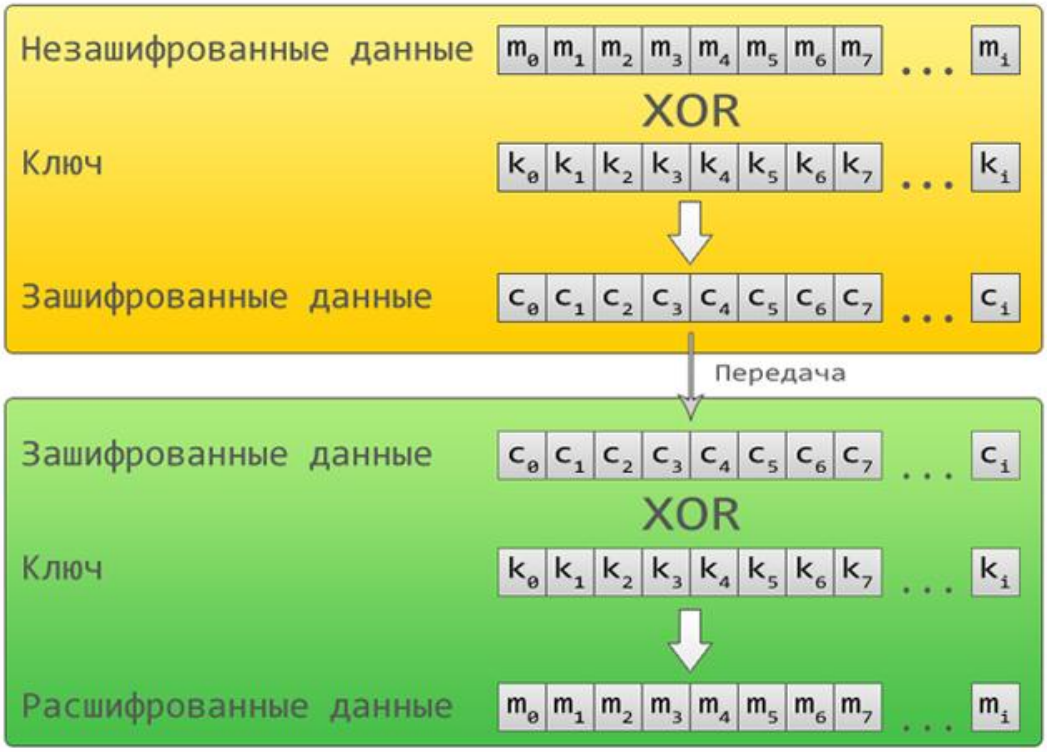
Алгоритм поточных шифров главным образом состоит из генератора гаммы, который выдаёт сам ключевой поток. Идёт генерация последовательности битов ($k_i$), которая потом соединяется с открытым текстом ($m_i$). Всё это делается суммированием по модулю два и у нас получается шифрограмма ($c_i$):

$$c_i = m_i \oplus k_i.$$

Путём регенерации ключевого потока ($k_i$) и сложения с шифрограммой ($c_i$) по модулю два получаем происходит расшифровка. Получаем исходный

незашифрованный текст ($m_i$), благодаря свойствам суммирования по модулю два.

$$m_i = c_i \oplus k_i = (m_i \oplus k_i) \oplus k_i$$




RC4 является алгоритмом, который определяется размером блока (*S-блок*). Длину *S-блока* определяет размер слова – *n*. Чаще всего *n* делают равным восьми, но для повышеня безопасности эту цифру надо увеличить. Увеличение размера *S-блока* не ведёт ни к каким противоречиям в алгоритме. Если увеличить *n* до 16 бит, то элементов в *S-блоке* станет 65536, поэтому время начальной итерации будет значительно больше. Зато скорость шифрования вырастет.

Состояние RC4 представляет собой массив размером $2^n$ и двух счётчиков. Массив – это наш *S-блок*, поэтому он содержит $2^n$ возможных значений слова. Счётчики обозначаются *i* и *j* соответственно.

Инициализация RC4 состоит из двух частей :

- Инициализация самого *S-блока*

Этот алгоритм называется **Key-Scheduling Algorithm** (**KSA**). В алгоритме используется ключ, который пользователь подаёт на вход. Он сохранён в *Key* и длина его составляет *L* байт. Процесс инициализации начинается с заполнения массива *S*, дальше идёт перемешивание массива, которое определяется ключом. Действие на *S* всегда выполняется только одно, поэтому должно выполняться утверждение, при котором *S* содержит только один набор значений, который был получен при первой инициализации (*S[i] := i*).

- Генерация псевдо-случайного слова *K*

***Pseudo-RandomGenerationAlgorithm (PRGA)*** – генератор псевдослучайной последовательности. Значения, хранящиеся в *S*, переставляются генератором ключевого потока RC4. За один цикл определяется одно *n*-битное слово из потока *K*. Потом исходный текст для зашифровки складывается по модулю два с ключевым словом и получется зашифрованный текст.

## 2.4 Тестирование защищенного канала

### *2.4.1 Методы тестирования*

Существуют несколько основных методов тестирования:

- Тестирования методом “черного ящика” (Blackboxtesting).

При этом методе тестирования у тестировщика есть доступ к программному обеспечению только через интерфейс программы, который доступен обычному пользователю. Модуль для тестов может эмулировать клики мышкой или нажатия клавиш программного обеспечения, которое тестируется. Он позволяет наблюдать, всё ли идёт правильно и совпадают ли эти действия с нажатями клавиш на мышке и клавиатуре в реальности. Чаще всего тестирование черного ящика ведётся с использование документов, где описываются требования к системе.

- Тестирование методом “белого ящика” (Whiteboxtesting).

При этом методе тестирования у тестировщика есть доступ к исходному коду и он может писать код, связанный с библиотеками тестируемого программного обеспечения, сам. При таком подходе проводятся тесты отдельных составляющих программы. При таком методе обеспечивается работоспособность и устойчивость всех компонентов системы.

- Тестирование методом “серого ящика” (Greyboxtesting).

При этом методе тестирования у тестировщика тоже есть доступ к коду программы, как и в методе белого ящика. Но в самом процессе теста, тестировщик не нуждвется в этом доступе.

Все методы тестирования продуктивны. Основной идеей является наличие доступа к исходному коду или нет. При тестировании методом белого ящика тестировщик получает возможность доступа к коду тестируемой программы, а при тестировании методом черного ящика имеет

только общедоступный пользовательский интерфейс программы. Для нашего тестирования больше подходит метод белого ящика.

### *2.4.2 Тестирование и его результаты.*

Проведем тестирования всех компонентов системы, по методу “белого ящика”.

Начнём с приложения, которое слушает на указанном порту и отправляет весь траффик на указанный адрес и порт.

Для этого в код была добавлена возможность отображения лога в консоли. При тестировании имеем:

0000: Listen at port 4343, remote host (62.67.42.162, 5190)

0000: Connection accepted from (192.168.1.34, 49961)

0001: Thread started

0001: Connecting to (62.67.42.162, 5190)

0001: Remote host: (192.168.1.34, 49961)

0001: Recevied from (192.168.1.34, 49961)

0001: 00-01-02-03-04-05-06-07-08-09-0A-0B-0C-0D-0E-0F

0001: 0000: 87

0001: Sent to (62.67.42.162, 5190)

0001: Recevied from (192.168.1.34, 49961)

0001: 00-01-02-03-04-05-06-07-08-09-0A-0B-0C-0D-0E-0F

0001: 0000: 5D-6A

0001: Sent to (62.67.42.162, 5190)

0001: Recevied from (192.168.1.34, 49961)

0001: 00-01-02-03-04-05-06-07-08-09-0A-0B-0C-0D-0E-0F

0001: 0000: 0A 0B

0001: Sent to (62.67.42.162, 5190)

0001: Recevied from (192.168.1.34, 49961)

0001:00-01-02-03-04-05-06-07-08-09-0A-0B-0C-0D-0E-0F

0001: 0000: 0A 0B

0001: Sent to (62.67.42.162, 5190)

0001: Recevied from (62.67.42.162, 5190)

0001: 00-01-02-03-04-05-06-07-08-09-0A-0B-0C-0D-0E-0F

0001: 0100: 76 3D 31 0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A

0001: 0110: 20 63 6C 6F 73 65 0D 0A 43 6F 6E 74 65 6E 74 2D

0001: 0120: 54 79 70 65 3A 20 74 65 78 74 2F 68 74 6D 6C 0D

0001: 0000: 48 54 54 50 2F 31 2E 31 20 33 30 32 20 46 6F 75

0001: 0010: 6E 64 0D 0A 44 61 74 65 3A 20 46 72 69 2C 20 30

0001: 0020: 34 20 53 65 70 20 32 30 30 39 20 30 38 3A 35 33

0001: 0030: 3A 30 33 20 47 4D 54 0D 0A 53 65 72 76 65 72 3A

0001: 0080: 54 0D 0A 43 61 63 68 65 2D 63 6F 6E 74 72 6F 6C

0001: 0090: 3A 20 6E 6F 2D 63 61 63 68 65 2C 20 6E 6F 2D 63

0001: 00B0: 22 2C 20 70 72 69 76 61 74 65 0D 0A 4C 6F 63 61

...

0001: 0040: 20 41 70 61 63 68 65 0D 0A 50 72 61 67 6D 61 3A

0001: 0050: 20 6E 6F 2D 63 61 63 68 65 0D 0A 45 78 70 69 72

0001: 0060: 65 73 3A 20 46 72 69 2C 20 30 31 20 4A 61 6E 20

0001: 0070: 31 39 39 39 20 30 30 3A 30 30 3A 30 30 20 47 4D

0001: 0130: 0A 0D 0A 52 65 64 69 72 65 63 74 20 70 61 67 65

0001: 00A0: 61 63 68 65 3D 22 53 65 74 2D 43 6F 6F 6B 69 65

0001: 0150: 73 20 6E 6F 74 68 69 6E 67 20 74 6F 20 73 65 65

0001: Connection reset by (62.67.42.162, 5190)

0001: Connection closed

Приложение слушает по порту 4343 и отправляет всё на указанный нами адрес и порт - 62.67.42.162, 5190.

Теперь произведем тестирование шифрования и дешифрования RC4. Для теста классов производится шифрование из текста в BASE64 и обратно.

Ключ: 748gfsd896hfsf6gh8

Строка для шифрования: subject for the test = 1

Результат шифрования:
FsO0VsO0woUeE8KjwqJ7YC5YwoDCoQsdeMKiw6sUGsOjEg==

Дешифрация: subject for the test = 1

Ключ: f7dgsdkgsagfn66383nf3

Строка для шифрования: testirovanie dlya diploma 123

Результат шифрования:
WcO1F2jDoHfDrBMBCMKdClMULMKgw7bCvGPDuh8ldcOsQFxswqfDsw==

Дешифрация: testirovanie dlya diploma 123

Это означает, что созданные классы для шифрации и дешифрации работают верно.

Тестирование всех отдельных компонентов сделано. По ним видно, что всё функционирует правильно.

## 2.5 Выводы

В данном разделе было осуществлено знакомство с архитектурой и алгоритмом функционирования защищенного канала. Проанализированы и выбраны инструментальные средства для его реализации (язык программирования и метод шифрования). И были произведены тесты всех составляющих частей, что показывает функционирование защищенного канала.

# Раздел 3. Охрана труда

## 3.1 Исследование всевозможных вредоносных факторов при использовании персональных компьютеров и их влияние на пользователей.

Охрана труда – это система лечебнно-профилактических и гигенических средств, законодательных актов, которые обеспечивают сохранность здоровья и безопасность человека в процессе труда.

Не существует полностью безопасных производственных процессов, поэтому основной задачей охраны труда является сведение к минимуму вероятности заболевания работника и обеспечение ему комфорта для максимальной производительности.

При работе за персональным компьютером человеку угрожает множество вредных факторов, которые ведут к уходшению здоровья. При долгой работе за компьютером появляется головная боль, начинается резь в глазах, появляются боли в мышцах шеи и рук. Это всё может привести к ухудшению качества зрения, различным кожным заболевания и даже мигреням.

Есть прямая зависимость между долгой работой за персональным компьютером и такими недомоганиями, как боли в спине и шее, быстрой утомляемости глаз, стрессовые состояния, возбудимости, депрессии, нарушения сна, головокружения и многим другим. Всё это очень сильно подрывает здоровье работников.

Дисплеи или мониторы являются главным источником всех проблем, связанных с орхраной здоровоья людей, которые работают за персональным компьютером. От них исходит вредоносное излучение, которое плохо влияет на здоровье пользователей.

Работа с персональным компьютером, как и любой производственный процесс, сопряжен с наличием опасных факторов.

Фактор, воздействие которого приводит к ухудшению здоровья работника, называется – опасным фактором.

Фактор, который приводит к заболеванию или летальному исходу, называется – вредным фактором. Вредный фатор может стать опасным, при должном уровне продолжительности воздействия.

Примерная конфигурация рабочего места, оснащенного персональным компьютером:

- Персональный компьютер с процессором IntelCorei7, клавиатура мышь, жесткий диск SATAII, DVD привод.
- Принтер HP LaserJet (A4).
- Монитор HP ProDisplay P221.

Проанализируем возможные вредные факторы, которые могут появиться при эксплуатации персональной машины, описанноый выше.

Питание персонального компьютера произвотдится от сети в 200В. Для человека безопасное напряжение – это 40В, значит при работе с ПК возникает риск поражения электрическим током.

В мониторе высоковольтный блок вырабатывает напряжение до 20кВ. Для человека вредным напряжением считается от 5 до 300кВ. При 15-25кВ появляется мягкое рентгеновское излучение, что является вредным фатором для человека.

Благодаря кадрово-частотной развёртке появляется изображение на мониторе. При кадровой развёртке частота достигает 85Гц. При строчной – 42Гц. Пользователь попадает в эту зону с электромагнитным излучением, чем может ухудшить себе здоровье.

Также дисплей персонального компьютера создаёт ультрафиолетовое излучение. Это излучение становится вредным для человека при достижении плотности в 10 Вт/м$^2$.

При длительной работе за персональным компьютером фактор ультрафиалетового излучения может нанести вред здоровью пользователя.

Из-за статического электричества мелкие чатицы и пыль притягиваются к экрану персонального компьютера во время его работы. Вся пыль, которая

собирается на экране ухудшает видимость и может попасть на лицо и в лёгкие. Всё это может вызвать различные заболевания кожи или дыхательных путей.

Проанализировав все возможные вредные факторы, которые возоможны при использовани персонального компьютера, следует защититься от них. При использовании персонального компьютера есть шанс возникновения следующих вредных факторов:

- *Статическое электричество*

Электрическая пыль приводит к появлению угрей, вызывает воспаление кожи и может испортить линзы.

Наэлектризованный экран дисплея притягивает частицы, которые летают в воздухе. Они скапливаются вблизи человеческого лица, воздух ухудшается и всё это может привести к кожным заболеваниям. Человек дышит этим запылившемся воздухом. Эффект может усилиться, если в помещении находятся ковры или какие-либо пыльные вещи.

- *Электромагнитное излучение*

При достижение частоты электромагнитного поля 60Гц и выше возможны изменения в клетках. Молекулы любого типа начинают колебаться из-за того, что электромагнитное поле постоянно колеблется с частотой порядка 60Гц.

Из-за этого у человека может ухудшиться клеточный иммунитет, ухудшиться активность ферментов. Есть случаи, когда это приводило к возникновению опухолей.

- *Электрический ток*

При воздействии на человека электрического тока есть возможность получить травму. Когда ток проходит через тело человека, то может оказать

следующие воздействия:

- Разложение крови (электрическое воздействие).
- Нагрев тканей (термическое воздействие).
- Раздражение тканей организма (биологическое воздействие).
- Сокращение мышц (механическое воздействие).

Тяжесть нанесенной травмы электрическим током зависит от:

- Возраста и пола человека.
- Состояния здоровья человека.
- Величины тока.
- Время, которое протекал ток.
- Путь, по которому протекал.
- Окружающей вокруг среды.
- Сопротивления самого человека.
- Частоты тока.

По степени опасности общие травмы (электроудары) разделяются на:

- Судорога и сокращение мышц, но без потери сознания.
- Судорога и сокращение мышц, с потерей сознания.
- Нарушение кровообращения и потеря сознания
- Клиническая смерть

Самым опасным токо для человека является ток 20 – 100ГЦ. Так как персональный компьютер питается от сети переменного тока частотой 50Гц, тот этот ток можно считать опасным для человека. Электрические ожоги, электроофтальмия – всё это местные травмы.

- *Ультрафиолетовое излучение*

Электромагнитное излучение в области, которая лежит в диапазоне 200

– 400 нм и примыкает к коротким волнам называется – ультрафиолетовым излучением.

Существуют несколько спектральных областей:

- Бактерицидная область спектра 200 – 280 нм.
- Зрительная область спектра 280 – 315 нм.
- Оздоровительная область спектра 315 – 400 нм.

При больших дозах и длительном воздействии могут возникнуть такие последствия:

- Рак кожи.
- Гибель клеток.
- Катаракта (повреждения глаз).
- Различные фототоксичные реакции.

- *Компьютерный стресс*

Пользователи, которые часто используют персональный компьютер, очень часто подвергаются различным болезням сердечно-сосудистой системы, психологическим стрессам и функциональным нарушениям нервной системы. Иммунитет человека очень сильно ослабевает.

Исходя из всего этого был выведен новый тип заболевания – компьютерный стресс.

Симтомов у этого заболевания очень много и они очень разнообразны. Из-за взаимосвязи всех органов человека наличие одного из симптомов маловероятно, поэтому чаще всего имеется целая группа.

Основные из симотомов:

- Физические недомогания:
  - Усталость и сонливость.
  - Головные боли.
  - Боли в ногах и спине.

    - Напряженные мышцы, боли в руках, покалывания.
- Глазные заболевания:
    - Жжение.
    - Острые боли.
- Ухудшение работоспособности:
    - Тяжело сосредоточиться.
    - Повышенная раздражительность.
- Проблемы с визуальным восприятием
    - Мутность в голове.
    - Расплывчатое зрение.
    - Появление двойного зрения.

Из анализа воздействия различных факторов на организм человека, который был проведён выше, следует необходимость защиты от них.

## 3.2 Методы и средства защиты пользователей от воздействия на них опасных и вредных факторов

В предыдущей главе мы проанализировали какие вредные факторы могут возникнуть при работе с персональным компьютером. В этой главе мы рассмотрим методы по защите пользователей от воздействия на них этих факторов:

- *Методы и средства защиты от электрического тока.*

Для того, чтобы обезопасить себя от поражения электрическим током используют зануление.

Преднамеренное соединение с нулевым проводником металличиеских составляющих, которые могут оказаться под напряжением, называется – заземление. Когда замыкается одна из фаз на заземляющем корпусе, в цепи появляется ток замыкания, который отключает сеть от пользователя. Снижается напряжение на корпусе, потому что ток короткого замыкания

вызывает перераспределение в сети. В этом заключается основная суть метода защиты человека от элекрического тока.

- *Методы и средства защиты от ультрафиолетового излучения.*

Плотность потока мощности Вт/м$^2$ является энергетической харатеристикой.

Принято, что единицей эр является биологический эффект воздействия. Один эр – поток 280 – 315 нм. И он равен потоку, мощность которого равна 1 Вт. При длительной работе за персональным компьютером сильно сказывается воздействие ультрафиолетового излучения.

Дозволенная доза облучения:

- За одну рабочую смену – 7,5 мэр*ч/м$^2$
- За сутки – 60 мэр*ч/м$^2$

Для того, чтобы защититься от ультрафиолетового излучения можно использовать :

- Защитные отчки и защитный фильтр.
- Побелка потолка и стен.
- Одежда из поплина и фланели.
- Зашторивать оконные проёмы.

- *Методы и средства защиты от электромагнитного излучения.*

Для защиты от электромагнитных излучений используются следующие способы:

- Находится от монитора на расстоянии не менее, чем на 1,5 метра.
- Экранирование
- Максимальное время работы – 4 часа.
- Расстояние от источника должно быть не менее 50 см.

- *Методы и средства защиты от статического электричества.*

Для защиты от статического электричества используют следующие способы:

- Влажная уборка.
- Использование экранов.
- Наличие контурного заземления.
- Использования нейтрализаторов статического электричества.
- Отсутствие синтетических покрытий.
- Подвижность воздуха в помещении должна быть не более 0,2 м/с.

Использование пользователем специальной рабочей одежды из малоэлектризующихся материалов, способствует уменьшению влияния статического электричества. Это может быть обувь с кожаной подошвой, халат из хлопчатобумажной ткани. Лучше не применять одежду из капрона и шёлка.

- *Методы защиты от синдрома компьютерного стресса.*

Вероятность возникновения синдрома компьютерного стресса можно свети до минима, если исключить все отрицательные факторы воздействия. Для того, чтобы снять все симптомы компьютерного стресса можно использовать следующие упражнения:

- Головная боль после работы.

Главное упражнение – это круговые движения головы. Также активное использование глаз, то есть переводы взгляда в разные точки помещения. Можно использовать упражнение – круговое движение плечами.

- Утомляемость и сонливость.

Также здесь актуально упражнение кругового движения головой. Активный перевод взгляда в разные точки, на близкое и дальнее расстояние (оба глаза открыты или использовать только один глаз).

- Боли в ногах, спине и бёдрах.

Потягивание мышц спины и потягивание всего тела.

- Воспаления глаза.

Упражнение на мигание (быстрое и медленное мигание глазом). Упражнение на смыкание век. Также круговые движения головой и переводы взгляда в различные точки. Фокусировка взгляда на какой-либо точке, потом резко на другой.

- Боли в руках или запястьях.

Упражнение по общему потягиванию. Возможно будет полезным быстрые махи пальцами.

- Напряженность в туловище.

Упражнение по общему потдягиванию. Круговые движения плечами и головой. Напряжение спинных мышц.

- Ошибки при печатании.

Упражнения по фокусированию взгляда на какой-либо точке, а потом резкий перевод в другую. Перевод взгляда на ближние и дальние расстояния. Вращательные движения пальцами.

- Раздражительность.

Нужно расслабиться, откинуться на кресле, расслабить плечи. Хорошо помогают круговые движения головой.

## 3.3 Эргономические требования к рабочим местам.

Нужно придерживать следующих рекомендаций, чтобы защититься от вредных и опасных факторов, которые возникают при использовании персонального компьютера:

- Нужно правильно организовать рабочее место оператора, используя все ограничения при работе с техникой.
- Нужно правильно организовать рабочие места.

### *3.3.1 Требования к помещениям и организации рабочих мест*

Главное требование к помещению, где располагаются персональные компьютеры, нельзя допускать расположения рабочих мест в подвальных помещениях. Объем одного рабочего места не должен быть меньше 20 $м^3$, а его площадь не меньше 6 $м^2$. В помещениях с ПК должен быть увлажнитель, чтобы повысить влажность воздуха. После окончания, в перерывы и перед началом работы помещение должно проветриваться.

Рекомендуемый микроклимат в помещении с ПК:

- Температура 20°С
- Подвижность воздуха около 0,2 м/с.
- Относительная влажность воздуха – 60%.

В помещении с персональными компьютерами уровень звука не должен певышать 65 дБА. Если в помещении расположена другая техника (принтеры, сканеры или что-то другое), то максимальный уровнь звука не должен превышать 75 дБА.

В рабочем помещении должно быть искусственное и естественное освещение. На окнах должны быть регулируемые жалюзи, которые позволят полностью закрывать оконные проёмы.

Рабочие места должны так располагаться к свету, чтобы он падал на них сбоку (желательно слева).

Дневной свет не должен попадать на мониторы персональных

компьютеров. Появятся блики и пользователю будет тяжело работать на таком мониторе.

Расстояние между рабочими местами и стенами (или оконными проёмами) должно быть не менее 1.5 м.

Поверхность пола должна обладать антистатическими свойствами. Он должен быть удобен для влажной уборки. Ну и сама поверхность должна быть ровной.

Освещение на рабочем месте должно быть:

- На мониторы – 200 лк.
- На клавиатуру и документы – 400 лк.

Для более освещенного рабочего места можно установить светильники. Главное, чтобы дополнительное освещение не создавало бликов на мониторе.

Для освещения дисплейных классов рекомендуется использовать лампы, мощность которых не превышает 40Вт. Использовать нужно люминесцентные лампы типа ЛБ, излучение которых находится в диапазоне от 3500 до 4200°K.

Разрешается применение в светильниках ламп накаливания. Нужно избегать на потолке зон чрезмерной освещенности, чтобы освещение было не прямым. Освещенность помещения должны быть равномерным, а потолок должен быть однородным и плоским. При установке рабочих мест нужно имет ввиду, что мониторы должны располагаться не ближе, чем на 2 метра друг к другу. Между рабочими местами должна быть установлена перегородка, высота которой 1,5-2 метра.

Дисплей должен быть установлен так, что его можно будет повернуть по вертикали и горизонтали в пределах 30 градусов. Окраска корпусов рабочих станций должна быть выполнена в мягких томах. Корпус дисплея должен быть матовым. Клавиатура и другие устройства должны быть одного цвета и иметь коэффициент отражения – 0,4-0,6. На рабочем месте не должно быть блестящих устройств, которые могут создать блики.

Рабочий стул должен быть регулируемым по высоте и подъемно-поворотным.

Расстояние от экрана монитора до глаз пользователя не должно быть меньше 500мм. В помещении каждый день должна проводиться влажная уборка.

Также на рабочем месте должна находиться подставка для ног, ширина которой не менее 300мм, а глубина 400мм. Клавиатуру надо распологать на столе на расстоянии 200-300мм от его края.

### *3.3.2 Требования к организации работы*

Работа в дисплейных классах для учителей и преподавателей институтов не должна превышать четырех часов в день. Для инженеров не должна превышать шести часов в день. Для обычного пользователья – не более двух часов.

Каждые два часа нужно делать пятнадцатиминутные перерывы и время от времени менять рабочую позу.

Перерыв нужно делать раз в час, если пользователь работает в ночную смену. При возникновении каких-либо симптомов дискомфорта нужно сделать перерыв между работой. Каждый человек сам знает насколько ему нужно ограначить время работы, чтобы прийти в норму. Все работники на персональных компьютерах должны проходить периодические медицинские осмотры. Женщины во время беременности не допускаются к ПК.

## 3.4 Выводы

Методы, которые были выбраны для защиты от вредоносных факторов, обеспечивают защиту пользователям, которые работают за персональными компьютерами или с любой другой вычислительной техникой.

# Заключение

По результатам дипломной работы можно сделать следующие выводы: был выполнен анализ существующих решений по организации защищенного канала, результаты которого позволили сформулировать требования к защищенному каналу. Реализован защищенный канал, удовлетворяя эксплутационным требованиям компании. Особенность предлагаемого решения заключается в модификации алгоритмов шифрования. Для реализации защищенного канала было разработано:

- Реализовано приложение для прослушивания порта и передачи данных на прокси сервер
- Реализована шифрация на прокси-сервере
- Реализована дешифрация на прокси-сервере
- Реализован общий алгоритм работы защищенного канала связи

В разделе охраны труда были исследованы всевозможные вредоносные факторы при использовании персональных компьютеров, проведен анализ методов и средств защиты пользователей от них и разработаны эргономические требования к рабочим местам.

Все необходимые данные представлены в пояснительной записке к диплому.

# Список литературы


1. Дмитрий Скляров. Искусство защиты и взлома информации – Санкт-Петербург “БХВ – Петербург” 2004.
2. В. Ф. Шаньгин. Информационная безопасность компьютерных систем и сетей – Москва ИД “ФОРУМ” – ИНФРА – М 2011.
3. В. Ф. Шаньгин Защита компьютерно информации. Эффективные методы и средства. – Москва ДМК Пресс, 2010.
4. Решение компании CiscoSystems по обеспечению безопасности корпоративных сетей (издание II).
5. Учебник. Информациоонные системы маркетинга – Региональный финансово-экономический инс-т. – Курск, 2008.
6. Бадд Т. Объектноориентированное программирование. – Спб. Питер, 1997.
7. Практическое пособие по созданию приложений и аплетов на языке программирования Java. Создание приложений и аплетов на языке Java (Часть 1).
8. Петрова М. С., Петров С. В., Вольхин С. Н. Охрана труда на производстве и в учебном процессе. – Москва 2006.

# Приложение

```
#include <cstdlib>
#include <cstddef>
#include <iostream>
#include <string>

#include <boost/shared_ptr.hpp>
#include <boost/enable_shared_from_this.hpp>
#include <boost/bind.hpp>
#include <boost/asio.hpp>
#include <boost/thread/mutex.hpp>


namespace tcp_proxy
{
  namespace ip = boost::asio::ip;

  class bridge : public boost::enable_shared_from_this<bridge>
  {
  public:

    typedef ip::tcp::socket socket_type;
    typedef boost::shared_ptr<bridge> ptr_type;

    bridge(boost::asio::io_service& ios)
    : downstream_socket_(ios),
      upstream_socket_(ios)
    {}

    socket_type& downstream_socket()
    {
      return downstream_socket_;
    }

    socket_type& upstream_socket()
    {
      return upstream_socket_;
    }

void start(const std::string& upstream_host, unsigned short upstream_port)

void handle_upstream_connect(const boost::system::error_code& error)

void handle_downstream_write(const boost::system::error_code& error)
```

```
void handle_downstream_read(const boost::system::error_code& error, const
size_t& bytes_transferred)
void handle_upstream_write(const boost::system::error_code& error)

void handle_upstream_read(const boost::system::error_code& error, const size_t&
bytes_transferred)
void close()

socket_type downstream_socket_;
socket_type upstream_socket_;

enum { max_data_length = 8192 };
unsigned char downstream_data_[max_data_length];
unsigned char upstream_data_[max_data_length];

boost::mutex mutex_;

public:

class acceptor

  };
}

int main(int argc, char* argv[])

{
  if (argc != 5)
  {
    std::cerr << "usage: tcpproxy_server <local host ip> <local port> <forward
host ip> <forward port>" << std::endl;
    return 1;
  }

  const unsigned short local_port   = static_cast<unsigned short>(::atoi(argv[2]));
  const unsigned short forward_port = static_cast<unsigned short>(::atoi(argv[4]));
  const std::string local_host      = argv[1];
  const std::string forward_host    = argv[3];

  boost::asio::io_service ios;

  try
  {
    tcp_proxy::bridge::acceptor acceptor(ios,
                                         local_host, local_port,
```

```
                                    forward_host, forward_port);
    acceptor.accept_connections();
    ios.run();
  }
  catch(std::exception& e)
  {
    std::cerr << "Error: " << e.what() << std::endl;
    return 1;
  }

  return 0;
}



void start(const std::string& upstream_host, unsigned short upstream_port)
    {
      upstream_socket_.async_connect(
          ip::tcp::endpoint(
              boost::asio::ip::address::from_string(upstream_host),
              upstream_port),
           boost::bind(&bridge::handle_upstream_connect,
               shared_from_this(),
               boost::asio::placeholders::error));
    }

void handle_upstream_connect(const boost::system::error_code& error)
    {
      if (!error)
      {
        upstream_socket_.async_read_some(
            boost::asio::buffer(upstream_data_,max_data_length),
            boost::bind(&bridge::handle_upstream_read,
                shared_from_this(),
                boost::asio::placeholders::error,
                boost::asio::placeholders::bytes_transferred));

        downstream_socket_.async_read_some(
            boost::asio::buffer(downstream_data_,max_data_length),
            boost::bind(&bridge::handle_downstream_read,
                shared_from_this(),
                boost::asio::placeholders::error,
                boost::asio::placeholders::bytes_transferred));
      }
      else
```

```
        close();
    }


void handle_downstream_write(const boost::system::error_code& error)
    {
      if (!error)
      {
        upstream_socket_.async_read_some(
            boost::asio::buffer(upstream_data_,max_data_length),
            boost::bind(&bridge::handle_upstream_read,
                shared_from_this(),
                boost::asio::placeholders::error,
                boost::asio::placeholders::bytes_transferred));
      }
      else
        close();
    }

{
      if (!error)
      {
        async_write(upstream_socket_,
             boost::asio::buffer(downstream_data_,bytes_transferred),
             boost::bind(&bridge::handle_upstream_write,
                  shared_from_this(),
                  boost::asio::placeholders::error));
      }
      else
        close();
    }

  void handle_upstream_write(const boost::system::error_code& error)
    {
      if (!error)
      {
        downstream_socket_.async_read_some(
            boost::asio::buffer(downstream_data_,max_data_length),
            boost::bind(&bridge::handle_downstream_read,
                shared_from_this(),
                boost::asio::placeholders::error,
                boost::asio::placeholders::bytes_transferred));
      }
      else
        close();
```

```
    }


void handle_upstream_read(const boost::system::error_code& error,
                          const size_t& bytes_transferred)
    {
      if (!error)
      {
        async_write(downstream_socket_,
            boost::asio::buffer(upstream_data_,bytes_transferred),
            boost::bind(&bridge::handle_downstream_write,
                shared_from_this(),
                boost::asio::placeholders::error));
      }
      else
        close();
    }



void close()
    {
      boost::mutex::scoped_lock lock(mutex_);
      if (downstream_socket_.is_open())
        downstream_socket_.close();
      if (upstream_socket_.is_open())
        upstream_socket_.close();
    }

class acceptor
    {
    public:

      acceptor(boost::asio::io_service& io_service,
            const std::string& local_host, unsigned short local_port,
            const std::string& upstream_host, unsigned short upstream_port)
      : io_service_(io_service),
        localhost_address(boost::asio::ip::address_v4::from_string(local_host)),
        acceptor_(io_service_,ip::tcp::endpoint(localhost_address,local_port)),
        upstream_port_(upstream_port),
        upstream_host_(upstream_host)
      {}

      bool accept_connections()
      {
```

```
      try
      {
        session_ = boost::shared_ptr<bridge>(new bridge(io_service_));
        acceptor_.async_accept(session_->downstream_socket(),
            boost::bind(&acceptor::handle_accept,
                this,
                boost::asio::placeholders::error));
      }
      catch(std::exception& e)
      {
        std::cerr << "acceptor exception: " << e.what() << std::endl;
        return false;
      }
      return true;
    }

  private:

    void handle_accept(const boost::system::error_code& error)
    {
      if (!error)
      {
        session_->start(upstream_host_,upstream_port_);
        if (!accept_connections())
        {
          std::cerr << "Failure during call to accept." << std::endl;
        }
      }
      else
      {
        std::cerr << "Error: " << error.message() << std::endl;
      }
    }

    boost::asio::io_service& io_service_;
    ip::address_v4 localhost_address;
    ip::tcp::acceptor acceptor_;
    ptr_type session_;
    unsigned short upstream_port_;
    std::string upstream_host_;
  };

};
}
```

```
class CBase64
{
public:
	CBase64(){}

	char *Encrypt(const char * srcp, int len, char * dstp)
	{
		register int i = 0;
		char *dst = dstp;

		for (i = 0; i < len - 2; i += 3)
		{
			*dstp++ = *(base64_map + ((*(srcp+i)>>2)&0x3f));
			*dstp++ = *(base64_map + ((*(srcp+i)<< 4)&0x30 |
(*(srcp+i+1)>>4)&0x0f ));
			*dstp++ = *(base64_map + ((*(srcp+i+1)<<2)&0x3C |
(*(srcp+i+2)>>6)&0x03));
			*dstp++ = *(base64_map + (*(srcp+i+2)&0x3f));
		}
		srcp += i;
		len -= i;

		if(len & 0x02 ) /* (i==2) 2 bytes left,pad one byte of '=' */
		{
			*dstp++ = *(base64_map + ((*srcp>>2)&0x3f));
			*dstp++ = *(base64_map + ((*srcp<< 4)&0x30 |
(*(srcp+1)>>4)&0x0f ));
			*dstp++ = *(base64_map + ((*(srcp+1)<<2)&0x3C) );
			*dstp++ = '=';
		}
		else if(len & 0x01 )  /* (i==1) 1 byte left,pad two bytes of '='  */
		{
			*dstp++ = *(base64_map + ((*srcp>>2)&0x3f));
			*dstp++ = *(base64_map + ((*srcp<< 4)&0x30));
			*dstp++ = '=';
			*dstp++ = '=';
		}

		*dstp = '\0';

		return dst;
	}
```

```
	void *Decrypt(const char * srcp, int len, char * dstp)
	{
		register int i = 0;
		void *dst = dstp;

		while(i < len)
		{
			*dstp++ = (B64_offset[*(srcp+i)] <<2 |
B64_offset[*(srcp+i+1)] >>4);
			*dstp++ = (B64_offset[*(srcp+i+1)]<<4 |
B64_offset[*(srcp+i+2)]>>2);
			*dstp++ = (B64_offset[*(srcp+i+2)]<<6 |
B64_offset[*(srcp+i+3)] );
			i += 4;
		}
		srcp += i;

		if(*(srcp-2) == '=')  /* remove 2 bytes of '='  padded while encoding */
		{
			*(dstp--) = '\0';
			*(dstp--) = '\0';
		}
		else if(*(srcp-1) == '=') /* remove 1 byte of '='  padded while encoding
*/
			*(dstp--) = '\0';

		*dstp = '\0';

		return dst;
	};

	size_t B64_length(size_t len)
	{
		size_t  npad = len%3;
		size_t  size = (npad > 0)? (len +3-npad ) : len; // padded for multiple of
3 bytes
		return  (size*8)/6;
	}

	size_t Ascii_length(size_t len)
	{
		return  (len*6)/8;
	}
```

```
};


import socket, string, threading, os, select, sys, time, getopt
from sys import argv

def usage():
name = os.path.basename(argv[0])
print "usage:", name, "-l listen_port -a host -p port [-L file]"
print " -a host        - address/host to connect"
print " -p port        - remote port to connect"
print " -l listen_port  - local port to listen"
sys.exit(1)

PORT = False
REMOTE_HOST = False
REMOTE_PORT = False
CONSOLE = True


try:
opts, args = getopt.getopt(argv[1:], "l:a:p:L:ch?v")
for opt in opts:
opt, val = opt
if opt == "-a":
REMOTE_HOST = val
elif opt == "-p":
REMOTE_PORT = int(val)
elif opt == "-l":
PORT = int(val)
else:
usage()

if not PORT:
raise StandardError, "Enter listen port"

if not REMOTE_HOST:
raise StandardError, "Enter remote host"

if not REMOTE_PORT:
raise StandardError, "Enter remote port"

except Exception, e:
print "error:", e, "\n"
```

```
usage()

# Remote host
REMOTE = (REMOTE_HOST, REMOTE_PORT)

# Create logging contitional variable
log_cond = threading.Condition()
queue = []

def logger():
global queue
while 1:
log_cond.acquire()

while len(queue) == 0:
log_cond.wait()

if CONSOLE:
for line in queue:
        print line

queue = []
log_cond.release()

# Thread safe logger
def log(thread, msg):
if CONSOLE:
log_cond.acquire()
queue.append("%04d: %s" % (thread, msg))
log_cond.notify()
log_cond.release()

def printable(ch):
return (int(ch < 32) and '.') or (int(ch >= 32) and chr(ch))

# Pre-build a printable characters map
printable_map = [ printable(x) for x in range(256) ]

# Thread safe dumper
def log_dump(thread, msg):

if CONSOLE:
log_cond.acquire()
width = 16
```

```
header = reduce(lambda x, y: x + ("%02X-" % y), range(width), "")[0:-1]
queue.append("%04d: ----: %s" % (thread, header))
queue.append("%04d:       %s" % (thread, '-' * width * 3))

i = 0
while 1:
line = msg[i:i+width]
if len(line) == 0: break
dump = reduce(lambda x, y: x + ("%02X " % ord(y)), line, "")
char = reduce(lambda x, y: x + printable_map[ord(y)], line, "")
queue.append("%04X: %04X: %-*s| %-*s" % (thread, i, width*3, dump, width,
char))
i = i + width

log_cond.notify()
log_cond.release()

# Spy thread
def spy_thread(local, addr, thread_id):
log(thread_id, "Thread started")

try:
log(thread_id, "Connecting to %s..." % str(REMOTE))
remote = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
remote.connect(REMOTE)
except Exception, e:
log(thread_id, "Unable connect to %s -> %s" % (REMOTE, e))
local.close()
return

LOCAL = str(addr)
log(thread_id, "Remote host: " + LOCAL)

try:
running = 1;
while running == 1:
rd, wr, er = select.select([local, remote], [], [local, remote], 3600)
for sock in er:
        if sock == local:
  log(thread_id, "Connection error from " + LOCAL)
running = 0
        if sock == remote:
log(thread_id, "Connection error from " + REMOTE)
running = 0
```

```
for sock in rd:
        if sock == local:
val = local.recv(1024)
if val:
log(thread_id, "Recevied from %s (%d)" % (LOCAL, len(val)))
log_dump(thread_id, val)
remote.send(val)
log(thread_id, "Sent to %s (%d)" % (REMOTE, len(val)))
else:
log(thread_id, "Connection reset by %s" % LOCAL)
running = 0;

        if sock == remote:
val = remote.recv(1024)
if val:
log(thread_id, "Recevied from %s (%d)" % (REMOTE, len(val)))
log_dump(thread_id, val)
local.send(val)
log(thread_id, "Sent to %s (%d)" % (LOCAL, len(val)))
else:
log(thread_id, "Connection reset by %s" % str(REMOTE))
running = 0;

except Exception, e:
log(thread_id, ("Connection terminated: " + str(e)))

remote.close()
local.close()

log(thread_id, "Connection closed")

try:
# Server socket
srv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
srv.bind(("", PORT))
except Exception, e:
print "error", e
sys.exit(1)

counter = 1
threading.Thread(target=logger, args=[]).start()
log(0, "Listen at port %d, remote host %s" % (PORT, REMOTE))
```

```
while 1:
srv.listen(1)
local, addr = srv.accept()
log(0, "Connection accepted from %s, thread %d launched" % (addr, counter))
threading.Thread(target=spy_thread, args=[local, addr, counter]).start()
counter = counter + 1
```

```
#ifndef EncdDecdHPP
#define EncdDecdHPP

#pragma delphiheader begin
#pragma option push -w-
#pragma option push -Vx
#include <Classes.hpp> // Pascal unit
#include <SysInit.hpp>  // Pascal unit
#include <System.hpp> // Pascal unit

namespace Encddecd
{
extern PACKAGE void __fastcall EncodeStream(Classes::TStream* Input,
Classes::TStream* Output);
extern PACKAGE void __fastcall DecodeStream(Classes::TStream* Input,
Classes::TStream* Output);
extern PACKAGE AnsiString __fastcall EncodeString(const AnsiString Input);
extern PACKAGE AnsiString __fastcall DecodeString(const AnsiString Input);

}          /* namespace Encddecd */
using namespace Encddecd;
#pragma option pop       // -w-
#pragma option pop       // -Vx

#pragma delphiheader end.
#endif   // EncdDecd
```

```
void inline SWAP(unsigned char &c1, unsigned char &c2)
{
   unsigned char t = c1;
   c1 = c2;
   c2 = t;
}
```

```
class CRC4
{
private:
   unsigned char sbox[256];      /* Encryption array             */
   unsigned char key[256],k;     /* Numeric key values           */
        int  m, n, i, j, ilen;       /* Ambiguously named counters   */
public:

        CRC4 ()
        {
                memset(sbox,0,256);
                memset(key,0,256);
        }
        virtual ~CRC4 ()
        {
                memset(sbox,0,256);  /* remove Key traces in memory  */
                memset(key,0,256);
        }

        char *Encrypt(char *pszText,const char *pszKey)
        {
                i=0, j=0,n = 0;
                ilen = (int)strlen(pszKey);

                for (m = 0;  m < 256; m++)  /* Initialize the key sequence */
                {
                        *(key + m)= *(pszKey + (m % ilen));
                        *(sbox + m) = m;
                }

                for (m=0; m < 256; m++)
                {
                        n = (n + *(sbox+m) + *(key + m)) &0xff;
                        SWAP(*(sbox + m),*(sbox + n));
                }

                ilen = (int)strlen(pszText);
                for (m = 0; m < ilen; m++)
                {
                        i = (i + 1) &0xff;
                        j = (j + *(sbox + i)) &0xff;
```

```
                        SWAP(*(sbox+i),*(sbox + j));  /* randomly Initialize the
key sequence */
                        k = *(sbox + ((*(sbox + i) + *(sbox + j)) &0xff ));
                        //if(k == *(pszText + m))       /* avoid '\0' beween the
decoded text; */
                        //        k = 0;
                        *(pszText + m) ^=  k;
                }
                return pszText;

        }

        char *Decrypt(char *pszText,const char *pszKey)
        {
                return Encrypt(pszText,pszKey) ;  /* using the same function as
encoding */
        }
};


class CRC4
{
public:

   CRC4 ()
   {
      memset(sbox,0,sizeof(sbox));
      memset(key,0,sizeof(key));
   }
   virtual ~CRC4 ()
   {
      memset(sbox,0,sizeof(sbox));
      memset(key,0,sizeof(key));
   }

   char *Encrypt(char *pszText,const char *pszKey, int iTextLen = 0)
   {
      i=0, j=0,n = 0;
      ilen = (int)strlen(pszKey);

      for (m = 0;  m < 256; m++)
      {
         *(key + m)= *(pszKey + (m % ilen));
         *(sbox + m) = m;
```

```
        }
        for (m=0; m < 256; m++)
        {
           n = (n + *(sbox+m) + *(key + m)) &0xff;
           swap(*(sbox + m),*(sbox + n));
        }

        if(iTextLen > 0)
        {
           ilen = iTextLen;
        }
        else
        {
           ilen = (int)strlen(pszText);
        }

        for (m = 0; m < ilen; m++)
        {
           i = (i + 1) &0xff;
           j = (j + *(sbox + i)) &0xff;
           swap(*(sbox+i),*(sbox + j));
           k = *(sbox + ((*(sbox + i) + *(sbox + j)) &0xff ));
           *(pszText + m) ^=  k;
        }

        return pszText;
     }

     char *Decrypt(char *pszText, const char *pszKey, int iTextLen)
     {
        if(iTextLen < 1)
        {
           return NULL;
        }

        return Encrypt(pszText, pszKey, iTextLen) ;
     }

  private:
     unsigned char sbox[256];
     unsigned char key[256],k;
     int  m, n, i, j, ilen;/
  };
```