

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт
Факультет приборостроения,
информационных технологий и
электроники

Кафедра
«Информационная безопасность
систем и технологий»

Специальность 10.05.02 –

Информационная безопасность
телекоммуникационных систем
Разработка защищенных
телекоммуникационных систем

Специализация –

ДИПЛОМНЫЙ ПРОЕКТ

на тему

ПРОГРАММНЫЙ КОМПЛЕКС ЭМУЛЯЦИИ УСТРОЙСТВА ШИФРОВАНИЯ
ДЛЯ IP-СЕТИ

Студент _____ Борисов Сергей Андреевич

Руководитель инженер 1 категории АО «ПНИЭИ» _____ Хворостухин С.П.

Нормоконтролёр к.т.н., доцент кафедры ИБСТ _____ Фатеев А.Г.

Рецензент начальник НИС АО «ПНИЭИ» _____ Зорин А.Н.

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой _____ С.Л. Зефилов

Работа защищена с отметкой _____ (протокол заседания ГЭК
от _____ № _____)

Секретарь ГЭК _____ А.Г. Фатеев

Пенза 2018

Оглавление

Введение.....	4
1 Обзор протоколов IP, IPSec.....	5
1.1 Описание протокола IP.....	5
1.2 Описание протоколов в составе IPSec	6
2 Обзор протоколов установления сеансов защищенной связи.....	14
2.1 Протокол IKE	14
2.2 Протокол TLS	15
3 Разработка структуры программного комплекса.....	22
4 Определение форматов и структур данных	24
4.1 Формат хранимой конфигурации	24
4.2 Формат конфигурационного файла.....	25
4.3 Формат сообщений при установлении сеанса защищенной связи	26
4.4 Формат пакетов в процессе шифрования и расшифрования.....	32
5 Разработка алгоритмов модулей программного комплекса	36
6 Реализация разработанных алгоритмов программных модулей.....	45
6.1 Реализация алгоритма обработки входящих пакетов	45
6.2 Реализация алгоритма инициализации программного комплекса.....	47
6.3 Реализация алгоритма обработки ARP пакетов.....	50
6.4 Реализация алгоритма обновления ARP кэша	50
6.5 Реализация алгоритма обработки IP пакетов.....	50
6.6 Реализация алгоритма обработки ICMP пакетов.....	51
6.7 Реализация алгоритма обработки ESP пакетов.....	52
6.8 Реализация алгоритма обмена ключами.....	54
7 Тестирование разработанного программного комплекса	58
8 Разработка руководства пользователя	67
8.1 Общие сведения	67
8.2 Первоначальная настройка	68
8.3 Пользовательская настройка.....	69

8.4 Работа с программой.....	71
8.5 Сообщения об ошибках	71
9 Вредные психофизиологические факторы, влияющие на опорно-двигательный аппарат специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие	72
Заключение	75
Список используемых источников.....	76
Приложение А Исходный код программного комплекса эмуляции устройства шифрования	78
Приложение Б Протокол проверки ВКР в системе «Антиплагиат».....	103

Введение

В настоящее время практически любая организация ведет свою деятельность с применением информационных технологий, которые так или иначе используют обмен информацией посредством сетевого взаимодействия. Наибольшее распространение среди данных технологий имеют сети, построенные по протоколам IP, например интернет. При обмене информацией в данных сетях возникает необходимость сохранить конфиденциальность передаваемых данных, так как успех любой деятельности заключается в получении и использовании определенных сведений раньше чем конкуренты, а чем ценнее данные сведения тем больше будут убытки при их утечке.

Для обеспечения конфиденциальности информации одним из самых надежных методов является их защита с применением криптографических средств защиты информации, применение которых, при использовании надежных алгоритмов шифрования, гарантирует преобразование данных таким образом, что они станут полностью бесполезны для злоумышленника.

При выборе средств защиты от угрозы раскрытия конфиденциальных сведений организации часто используют специализированные устройства шифрования способные надежно функционировать в различных условиях обеспечивая при этом безотказность в течении долгого промежутка времени.

Данные устройства являются программно-аппаратными комплексами, разработка которых зависит от качества проводимого тестирования, зачастую включающего большое количество изменений в разрабатываемое устройство. Использование программных эмуляторов, выполняющих функции готового устройства шифрования помогают существенно снизить расходы при производстве аппаратной составляющей устройства, так как предоставляется возможность отладить большую часть функций устройства без использования его аппаратной части.

Целью данного дипломного проекта является разработка программного комплекса эмуляции устройства шифрования IP-сети.

1 Обзор протоколов IP, IPSec

1.1 Описание протокола IP

IP (Internet Protocol) – протокол сетевого уровня стека TCP/IP. Функции и описание данного протокола приведено в стандарте RFC 791 следующим образом: «Протокол IP осуществляет передачу блоков данных, называемых дейтаграммами, от отправителя к получателю, где отправители и получатели являются узлами в сети, идентифицируемыми адресами фиксированной длины (IP-адресами). Протокол IP обеспечивает при необходимости также фрагментацию и сборку дейтаграмм для передачи данных через сети с малым размером пакетов». Протокол IP выполняет объединение сегментов сети в единую сеть, обеспечивая доставку пакетов данных между любыми хостами сети через произвольное число промежуточных узлов. По сетевой модели OSI протокол IP находится на третьем уровне. Данный протокол не гарантирует надёжной доставки пакета до адресата они могут быть доставлены не в том порядке, в котором отправлялись, дублироваться, оказаться повреждёнными или не прийти вовсе [1].

Структура пакета протокола IP версии 4 представлена в таблице 1, описание некоторых его полей приведено ниже.

Таблица 1 – Структура IP пакета версии 4

Биты	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	Версия			IHL			Тип обслуживания								Длина пакета																	
32	Идентификатор															Флаги		Смещение фрагмента														
64	TTL							Протокол								Контрольная сумма заголовка																
96	IP адрес отправителя																															
128	IP адрес получателя																															
160	Параметры (если IHL > 5)																															
...	Данные																															

- версия. Значение данного поля указывает на используемую версию протокола, для IP четвертой версии это значение равно 4;
- тип обслуживания. Указывает параметры желаемого качества обработки дейтаграмм для протокола верхнего уровня и назначает им приоритет;
- IHL (Internet Header Length). Данное поле содержит значение длины заголовка IP пакета в 32-битных словах. Оно указывает на начало блока данных (payload) в пакете;
- длина пакета (Total Length). Данное поле содержит значение длины пакета в октетах, включая заголовок и данные. Минимальное корректное значение для этого поля равно 20, максимальное - 65 535;
- идентификатор (Identification). Значение поля назначается отправителем пакета и предназначено для определения правильной последовательности фрагментов при сборке пакета. Для фрагментированного пакета все фрагменты имеют одинаковый идентификатор;
- флаги. Первый бит должен быть всегда равен нулю, второй бит DF (don't fragment) определяет возможность фрагментации пакета и третий бит MF (more fragments) показывает, не является ли этот пакет последним в цепочке пакетов;
- смещение фрагмента (Fragment Offset). Значение данного поля определяет позицию фрагмента в потоке данных;
- время жизни (TTL). Данное поле содержит значение количества маршрутизаторов, которые может пройти этот пакет. При прохождении маршрутизатора это число уменьшается на единицу. Если значение этого поля равно нулю, то пакет должен быть отброшен [1];
- протокол. Данное поле содержит идентификатор интернет-протокола следующего уровня, который указывает какому протоколу принадлежат данные в пакете.

1.2 Описание протоколов в составе IPSec

1.2.1 Основные функции и возможности IPSec. IPSec – набор протоколов, обеспечивающих защиту данных, передаваемых по протоколу IP. К основным

возможностям относится подтверждение подлинности (аутентификация), проверка целостности и шифрование IP пакетов. IPsec также включает в себя протоколы для защищённого обмена ключами в сети Интернет.

В первую очередь это протоколы обеспечения аутентичности (Authentication Header – заголовок аутентификации) и конфиденциальности (Encapsulating Security payload – протокол инкапсулирующей защиты содержимого), а также механизмы управления криптографическими ключами. На более низком архитектурном уровне располагаются конкретные алгоритмы шифрования, контроля целостности и аутентичности. В основе находится база данных, называемая доменом интерпретации (Domain of Interpretation), которая содержит информацию об алгоритмах, их параметрах и идентификаторов протоколов [2].

IPSec реализуется на узле сети или шлюзе безопасности, что обеспечивает защиту IP трафика. В основе защиты лежат требования, которые определены в базе данных политики безопасности (Security Policy Database), составляемой системным администратором. Полученные пакеты обрабатываются одним из трех способов на основании соответствия информации заголовка IP или транспортного уровня записям в базе данных политики безопасности. Варианты развития событий для каждого пакета:

- отбрасывается сервисом безопасности IPSec;
- пропускается без изменения;
- обрабатывается сервисом IPSec на основе применения определенной политики.

Протоколы в составе IPSec реализуют механизмы безопасности на IP уровне, определяя нужные протоколы безопасности, алгоритмы и предоставляя криптографические ключи необходимые сервисам. Они также могут использоваться для обеспечения защиты путей между узлами сети, между шлюзами безопасности или между шлюзом безопасности и узлом сети.

Эти протоколы могут использоваться отдельно либо совместно друг с другом. Каждый протокол поддерживает два режима использования:

транспортный и туннелирования. В транспортном режиме обеспечивается защита в основном для протоколов более высокого уровня, а в режиме туннелирования для защиты IP заголовков исходных пакетов.

В механизмах аутентификации и конфиденциальности для передачи данных между двумя компьютерами протокол IPSec предусматривает предварительное выполнение следующих шагов (фаз):

а) каждый компьютер должен быть, прежде всего, аутентифицирован. Различные реализации IPSec могут предусматривать различные методы. Создается общий главный ключ на обоих компьютерах. Создается защищенный канал для ведения переговоров о параметрах в ходе второй фазы [3];

б) эта фаза является ключевой в IPSec и заключается в создании защищенной связи SA (Security Association). При этом отправитель и получатель должны договориться о применяемых протоколах шифрования, определить ключи, времена их действия, а также другие параметры. Ниже приводится список параметров защищенной связи. Эти параметры сохраняются в базе данных обеих сторон. Защищенная связь создается процедурой управления ключами с помощью одного из Интернет-протоколов обмена ключами IKE (Internet Key Exchange). Протокол IKE опирается на протоколы ISACMP (Internet Security Association and Key Management Protocol – протокол управления ассоциациями и ключами защиты в сети Интернет) и Oakley. Протокол Oakley реализует обмен ключами, выполняемый по усовершенствованной схеме Диффи-Хеллмана. Защищённая связь однозначно определяется следующими тремя характеристиками, которые передаются по защищенному каналу [3];

в) индекс параметров защиты, SPI (Security Parameters Index). Данное значение присваивается защищенной связи и имеет только локальное значение. Индекс параметров защиты передается в заголовках AH и ESP, для того, чтобы система, которая принимает, имела возможность определить защищенную связь, в которой должен быть обработан пакет [3];

г) адрес IP пункта назначения. Значение содержит адрес пункта назначения защищенной связи, который может представлять систему конечного пользователя, сетевой объект типа брандмауэра или маршрутизатора;

д) идентификатор протокола защиты. Данный идентификатор определяет, является ли данная связь защищенной связью АН или это защищенная связь ESP. Ядро IPSec по передаче данных составляет два протокола:

1) АН (Authentication Header – заголовок аутентификации), гарантирующий аутентификацию передаваемых сообщений, т.е. подлинность источника сообщений и целостность данных;

2) ESP (Encapsulating Security Payload – защищенная полезная нагрузка), шифрующий передаваемые данные, т.е. обеспечивает конфиденциальность данных. ESP, кроме этого, может выполнить функции протокола АН.

Защищенная связь характеризуется следующими основными параметрами:

- счетчик порядкового номера. Значение, которое используется при генерации значений поля порядкового номера в заголовках АН или ESP [4];

- окно защиты от угрозы повтора. Необходимо для выявления воспроизведенных пакетов среди полученных пакетов АН или ESP [4];

- информация АН. Алгоритм аутентификации, ключи, параметры продолжительности жизни ключей и другие необходимые параметры, используемые в рамках АН [4];

- продолжительность жизни данной защищенной связи;

- информация ESP. Алгоритм шифрования и аутентификации, ключи, значения инициализации, параметры продолжительности жизни ключей и другие необходимые параметры, используемые в рамках ESP;

- режим протокола IPSec. Туннельный или транспортный;

- максимальная единица передачи (Maximum Transmission Unit). Максимальная единица передачи (максимальный размер пакета, который может

быть передан без фрагментации) для всех участков маршрута и переменные времени существования [3].

1.2.2 Протокол АН. Аутентифицирующий заголовок является обычным опциональным заголовком и располагается между основным заголовком пакета IP и полем данных. Его наличие не оказывает никакого влияния на процесс передачи информации транспортного и более высокого уровней. Основным назначением данного протокола является обеспечение защиты от атак, связанных с несанкционированным изменением содержимого пакета и от подмены исходного адреса сетевого уровня. Протоколы более высоких уровней должны быть изменены в целях осуществления проверки аутентичности полученных данных [5].

В режиме открытой передачи самым внешним заголовком дейтаграммы является заголовок IP, за которым расположен заголовок АН, а затем - поле данных дейтаграммы. Протокол АН защищает всю дейтаграмму, за исключением изменяемых полей. Однако вся информация, содержащаяся в дейтаграмме, передается открыто, поэтому она может быть перехвачена. Таким образом при передаче требуется меньше ресурсов, чем при работе в режиме туннеля. Однако обеспечивает более низкий уровень защиты.

В режиме туннеля создается новый заголовок IP, который становится самым внешним заголовком дейтаграммы. Заголовок АН помещается после нового заголовка IP. Затем размещается сама дейтаграмма (заголовок IP и данные). В этом случае протокол АН позволяет идентифицировать всю дейтаграмму, и получатель может определить, была ли она изменена во время передачи [5].

Формат АН приведен в таблице 2. Он состоит из заголовка длиной 96 бит и данных переменной длины, состоящих из 32-битовых слов. Аутентифицирующий заголовок содержит следующие поля: Next Header указывает на следующий заголовок, Payload Len представляет длину пакета, SPI является указателем на контекст безопасности и Sequence Number содержит последовательный номер пакета.

Таблица 2 – Формат заголовка АН

Биты	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	Next Header								Payload Len								Reserved															
32	Security Parameters Index (SPI)																															
64	Sequence Number																															
96	Authentication Data																															

Значение поля «последовательный номер пакета» формируется отправителем и служит для защиты от атак, связанных с повторным использованием данных процесса аутентификации. Так как сеть Интернет не гарантирует правильный порядок доставки пакетов, получатель должен хранить информацию о максимальном номере пакета в последовательности, который прошел успешную аутентификацию, а также о получении некоторого числа пакетов, содержащих предыдущие номера последовательности.

В отличие от алгоритмов вычисления контрольной суммы, которые направлены на исправление случайных ошибок при передаче, механизмы обеспечения целостности данных в открытых телекоммуникационных сетях должны иметь средства защиты от внесения целенаправленных изменений. Одним из таких механизмов является специальное применение алгоритма MD5: в процессе формирования заголовка аутентификации последовательно вычисляется хэш-функция от объединения самого пакета и некоторого предварительно согласованного ключа, а затем от объединения полученного результата и преобразованного ключа.

1.2.3 Протокол ESP. ESP (Encapsulating Security Payload) – протокол сетевого уровня стека TCP/IP входит в набор протоколов IPSec, обеспечивающих защиту данных, передаваемых по межсетевому протоколу IP. ESP обеспечивает конфиденциальность данных и позволяет идентифицировать отправителя, а также обеспечить целостность данных и защиту от воспроизведения [6].

Структура ESP пакета представлена в таблице 3.

Таблица 3 – Структура ESP пакета

Биты	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	Security Parameters Index (SPI)																															
32	Sequence number																															
64	Payload data																															
	Padding																															
																	Padding length							Next header								
	IntegrityCheckValue (ICV)																															

Описание полей заголовка ESP приведено ниже.

– security parameters index – индекс параметров безопасности. Значение данного поля вместе с IP адресом получателя и протоколом безопасности (ESP), определяет защищённое виртуальное соединение для данного пакета [6];

– sequence number – последовательный номер. Данное поле предназначено для защиты от повторной передачи. В нем содержится монотонно возрастающее значение параметра. Получатель может отказаться от услуги по защите от повторной передачи пакетов, но это значение всегда должно присутствовать в заголовке. Отправитель всегда должен учитывать это поле [6];

– payload data – это поле содержит данные (в зависимости от выбора режима – туннельного или транспортного, оно может содержать весь исходный инкапсулированный пакет или только его данные) в соответствии с полем «Next Header». Это поле является обязательным и состоит из целого числа байтов. При использовании алгоритма шифрования этого поля, который требует данные синхронизации криптопроцессов, это поле может содержать их [6];

– padding – дополнение. Данное поле необходимо для выравнивания нешифрованных данных до нужного размера, если это необходимо при использовании алгоритма шифрования, который требует конкретного размера шифруемых данных [6];

– padding length. Данное поле содержит размер дополнения (в байтах) [6];

– next header. Значение данного поля определяет тип данных, которые содержатся в поле «Payload data» [6];

– integrity check value – контрольная сумма. Значение данного поля необходимо для аутентификации и проверки целостности пакета [6].

В режиме открытой передачи заголовок ESP указывается после заголовка IP дейтаграммы. Если у дейтаграммы уже есть заголовок IPSec, то заголовок ESP помещается перед этим заголовком. Концевик ESP и идентификационные данные, если они есть, указываются после поля данных.

В режиме открытой передачи заголовок IP не идентифицируется и не зашифровывается. В этом случае адреса, указанные в заголовке, могут быть перехвачены во время передачи дейтаграммы по сети. Для обмена данными в режиме открытой передачи требуется меньше ресурсов, чем при работе в режиме туннеля. Однако такой режим обеспечивает более низкий уровень защиты. В большинстве случаев при работе с протоколом ESP применяется режим открытой передачи.

В режиме туннеля создается новый заголовок IP, который становится самым внешним заголовком дейтаграммы. После него помещается заголовок ESP, а затем - сама дейтаграмма (заголовок IP и данные). Концевик ESP и идентификационные данные, если они есть, добавляются в конец поля данных. Если одновременно применяются средства шифрования и идентификации, то ESP полностью защищает исходную дейтаграмму, так как эта дейтаграмма становится полем данных в новом пакете ESP. Новый заголовок IP не защищается. При установлении соединения между шлюзами следует применять ESP в режиме туннеля [6].

2 Обзор протоколов установления сеансов защищенной связи

2.1 Протокол IKE

2.1.1 Общие сведения протокола IKE. IKE - протокол, который объединяет все компоненты IPsec в единое целое. В частности, IKE обеспечивает первоначальную аутентификацию между двумя сторонами, а также реализует обмен секретными ключами [7].

Существует возможность вручную установить сеансовый ключ. В данном случае протокол IKE не используется, но этот вариант не рекомендуется и поэтому практически не используется. IKE обычно прослушивает и отправляет на UDP-порт 500, но сообщения IKE могут также приниматься на UDP-порту 4500. Поскольку UDP является дейтаграммным (ненадежным) протоколом, IKE включает восстановление после передачи пакета с ошибкой, в том числе потерю пакетов, повтор пакетов и подделку пакетов.

Существует две версии протокола, IKE и более новая его версия - IKEv2. В спецификациях и функционировании этих протоколов есть некоторые различия. IKEv2 устанавливает параметры соединения за одну фазу, состоящую из нескольких шагов. Процесс работы IKE можно разбить на две фазы [7].

2.1.2 Первая фаза. IKE создает безопасный канал между двумя узлами сети, называемый ассоциацией безопасности (SA), который представляет собой набор политик и ключей, используемых для защиты информации. ISAKMP SA - это общая политика и ключи используемые сторонами в этом протоколе для защиты обмена сообщениями. Также, в этой фазе два узла согласуют сеансовый ключ по алгоритму Диффи-Хеллмана. Первая фаза IKE может проходить в одном из двух режимов:

а) основной режим - состоит из трёх обменов между двумя сторонами - отправителем и получателем:

1) при первом узлы согласовывают алгоритмы и хэш-функции, используемые для защиты соединения IKE, посредством установления IKE SA для каждого узла [7];

2) используя алгоритм Диффи-Хеллмана, участники обмениваются секретным ключом. Также стороны проверяют и идентифицируют друг друга путём передачи и подтверждения последовательности псевдослучайных чисел [7];

3) производится подтверждение обмена Диффи-Хеллмана. В результате выполнения основного режима создается безопасный канал для последующего ISAKMP — обмена;

б) агрессивный режим – использует меньшее число обменов сообщениями. В первом обмене передается практически вся необходимая для установления IKE SA информация: открытые значения Диффи-Хеллмана, подтверждаемые другим участником и идентификатор пакета. Отвечающая сторона во втором сообщении посылает в ответ все необходимые данные для прохождения идентификации. В третьем сообщении первый узел проходит идентификацию и подтверждает участие в обмене [7].

2.1.3 Вторая фаза. В данной фазе протокола IKE присутствует только один режим, называемый быстрым. Быстрый режим не является полным обменом, так как используется только после создания SA в процессе первой фазы. Он согласует общую политику IPsec, получает общие секретные ключи для алгоритмов протоколов IPsec (AH или ESP), устанавливает IPsec SA. Использование последовательной нумерации сообщений обеспечивает защиту от атак повтора. Быстрый режим производит обновление общих секретных ключей, используя алгоритм Диффи-Хеллмана из первой фазы. [7]

2.2 Протокол TLS

2.2.1 Общие сведения протокола TLS. Протокол защиты транспортного уровня TLS (Transport Layer Security), стандартизирован в документе RFC 2246 [9]. Этот механизм расположен в виде подуровня между транспортным и прикладным уровнями TCP/IP. TLS является версией протокола защищенных сокетов SSL (Security Sockets Layer). Под сокетом понимается совокупность номера порта транспортного уровня и IP-адреса. Сокет однозначно

идентифицирует прикладной процесс в Internet. Преимуществом использования протоколов ИБ на прикладном уровне является возможность оптимального построения механизма защиты в зависимости от требований конкретного приложения. Преимуществом протоколов SSL/TLS является обеспечение прозрачности средств защиты для прикладных программ.

TLS состоит из двух субпротоколов, один из которых предназначен для передачи данных по установленной защищенной связи, а второй для установления защищенной связи. Эта процедура установления защищённой связи SA (Security Association) является такой же ключевой, как и в IPSec.

2.2.2 Установление защищенной связи. Процесс установления защищенной связи (SA) между клиентом и сервером состоит из четырех этапов представленных на рисунке 1 [9].

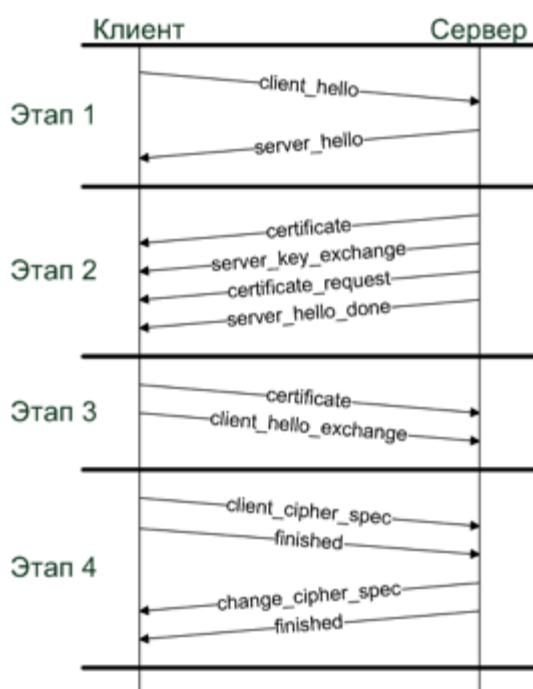


Рисунок 1 – Процесс установления защищенной связи

На первом этапе клиент и сервер определяют параметры защиты, такие как версия протокола, идентификатор сеанса, комплект шифров, метод сжатия и исходные случайные числа.

На втором этапе сервер передает сертификат, сообщение обмена ключами и запрос сертификата. Сервер сообщает об окончании фазы приветственного сообщения.

На третьем этапе клиент передает запрошенный сертификат и сообщение обмена ключами.

На четвертом этапе происходит смена комплектов шифров и завершение работы протокола.

2.2.2.1 Определение характеристики защиты. На этом этапе производится инициализация соединения, и определяются связанные с ним параметры защиты. Процесс инициализируется клиентом, который отправляет серверу сообщение «clienthello» с параметрами: версия протокола, начальные случайные значения, идентификатор сеанса, метод сжатия и комплект шифров. Случайные значения используются для защиты от атаки «повтор».

Работа TLS описывается в двух важных понятиях – сеанс и соединение. Сеанс описывает набор параметров защиты, которые могут использоваться несколькими соединениями. Идентификатор сеанса говорит о намерении клиента создать новый сеанс или создать новое соединение в пределах одного и того же сеанса [9].

Каждый комплект шифров определяет алгоритм обмена ключей и список, состоящий из следующих полей:

- алгоритм шифрования;
- алгоритм хеш-функции;
- тип шифра (поточный, блочный);
- длина хеш-кода;
- длина вектора инициализации (IV).

Протокол TLS поддерживает следующие методы обмена ключей для шифрования с общим ключом и вычисления значений открытых ключей по протоколу HMAC [10]:

- RSA. Секретный ключ зашифровывается открытым ключом получателя по алгоритму RSA. Для этого отправителю необходим сертификат открытого ключа получателя [10];

– метод Диффи-Хеллмана с фиксированными параметрами. Метод обмена ключами по алгоритму Диффи-Хеллмана, в котором сертификат сервера хранит открытые параметры алгоритма Диффи-Хеллмана, которые подписаны центром сертификации. Клиент сообщает свои параметры алгоритма Диффи-Хеллмана либо в сообщении обмена ключами, либо в сертификате, если необходима аутентификация клиента. Это обеспечивает защиту от атаки «человек посередине» [10];

– метод Диффи-Хеллмана с одноразовыми параметрами. Данный метод используется при создания одноразовых секретных ключей. В этом случае стороны обмениваются открытыми ключами по алгоритму Диффи-Хеллмана, которые подписаны закрытым ключом отправителя, используя алгоритм RSA. Для проверки подписи получатель использует соответствующий открытый ключ. Для аутентификации открытых ключей применяются сертификаты. Это обеспечивает защиту от атаки «человек посередине» [10];

– анонимный метод Диффи-Хеллмана. Предполагает использование немодифицированного алгоритма Диффи-Хеллмана, без выполнения аутентификации. Данный метод уязвим к атаке «человек-посередине», когда злоумышленник проводит обмен ключами между сторонами по анонимному методу Диффи-Хеллмана [10].

После отправления сообщения «clienthello» (приветствие клиента), клиент ожидает от сервера сообщения «serverhello» (приветствие сервера), которое содержит согласованные алгоритмы шифра, алгоритмы обмена ключами и другие параметры, что и в сообщении «clienthello».

2.2.2.2 Аутентификация и обмен ключами сервера. Данный этап начинается с отправки сервером его сертификата. Отправляемое сообщение содержит сертификат X.509 [8]. Сообщение «certificate» требуется для любого метода обмена ключами, кроме анонимного метода Диффи-Хеллмана. При использовании метода Диффи-Хеллмана с фиксированными параметрами сообщение сертификата является сообщением обмена ключами, так как в нем

содержатся предлагаемые сервером открытые параметры алгоритма Диффи-Хеллмана.

Затем отправляется сообщение «server_key_exchange». Отправка данного сообщения не требуется в случаях - если сервер уже отправил сертификат для использования метода Диффи-Хеллмана с фиксированными параметрами, и если предлагается использовать схему обмена ключами по алгоритму RSA. Сообщение «server_key_exchange» необходимо когда используются следующие схемы:

- анонимный метод Диффи-Хеллмана;
- метод Диффи-Хеллмана с одноразовыми параметрами.

Стандартно подпись создается вычислением хеш-суммы сообщения с последующим зашифровыванием ее закрытым ключом отправителя. Функция хеширования вычисляется для аргумента, включающего не только параметры Диффи-Хеллмана или RSA, но и оба сообщения приветствия на этапе 1, представленного на рисунке 2. Это обеспечивает защиту от атак воспроизведения сообщений и атак фальсификации ответа. Затем сервер, который не использует анонимный метод Диффи-Хеллмана, запрашивает сертификат клиента. Сообщение «certificate_request» включает два параметра: «certificate_type» и «certificate_authorities». Параметр «certificate_type» указывает на используемый алгоритм шифрования с открытым ключом который может быть следующим:

- RSA, используемый только для подписи;
- RSA, используемый методом Диффи-Хеллмана с одноразовыми параметрами.

Параметр «certificate_authorities» включает список имен допустимых центров сертификации.

Последнее сообщение данного этапа это сообщение «server_done», которое свидетельствует о завершении фазы приветствия сервера. Данное сообщение не имеет параметров. После отправки этого сообщения сервер ожидает ответа клиента.

2.2.2.3 Аутентификация и обмен ключами клиента. После того как получено сообщение «server_done», клиент производит проверку того, что сервер предоставил действующий сертификат, также производится проверка корректности параметров сообщения «server_hello». Если сервер запросил сертификат, клиент начинает данный этап с отправки серверу сообщения «certificate». Затем отправляется сообщение «client_key_exchange», обязательное для данного этапа. Содержимое сообщения зависит от выбранного метода обмена ключами и может быть следующим:

- RSA. Клиент создает предварительный ключ размером 48 байт и зашифровывает его с открытым ключом сертификата сервера. Данный предварительный ключ позволяет вычислить главный ключ;

- метод Диффи-Хеллмана с одноразовыми параметрами, или анонимный метод Диффи-Хеллмана.

Для метода Диффи-Хеллмана так же, как для RSA предусмотрено создание предварительного ключа у клиента и сервера. На основании предварительного ключа клиент и сервер по специальному алгоритму с использованием хеширования создают главный ключ. При этом используются случайные значения (нонсы) в сообщениях этапа 1, представленного на рисунке 2. Из главного ключа создается набор ключей различного назначения. Процедура их создания аналогична процедуре создания главного ключа из предварительного ключа. Список этих ключей приведен ниже [8]:

- ключ симметричного шифрования, используемый сервером для шифрования данных и расшифрования их клиентом;

- ключ симметричного шифрования, используемый клиентом для шифрования данных и расшифрования их сервером;

- секретный ключ, применяемый клиентом для вычисления кода аутентичности, используемого по протоколу HMAC для аутентификации сообщений в части целостности данных и неотказуемости [10];

– секретный ключ, применяемый сервером для вычисления кода аутентичности, используемого по протоколу HMAC для аутентификации сообщений в части целостности данных и неотказуемости;

– вектор инициализации IV, используемый в режиме блочного шифрования. Используются два вектора инициализации – один при передаче сообщений клиентом, другой – сервером.

2.2.2.4 Завершение. Этот этап завершает создание защищенной связи SA. Клиент отправляет сообщение «change_cipher_spec», затем копирует параметры состояния ожидания в поле текущего состояния. После этого клиент отправляет сообщение «finished», зашифрованное новым алгоритмом с новыми ключами и секретными значениями. Данное сообщение подтверждает успешность ключевого обмена и аутентификации.

В ответ на полученные сообщения сервер посылает свое сообщение «change_cipher_spec», копирует параметры состояния ожидания в текущее состояние и посылает свое сообщение «finished». После всех успешно произведенных действий клиент и сервер могут начать обмен данными на уровне приложения.

3 Разработка структуры программного комплекса

Программный комплекс представляет собой маршрутизатор обеспечивающий криптографическую защиту информации, передаваемой между такими же маршрутизаторами, отвечающими за другие подсети. Структура программного комплекса представлена на рисунке 2.

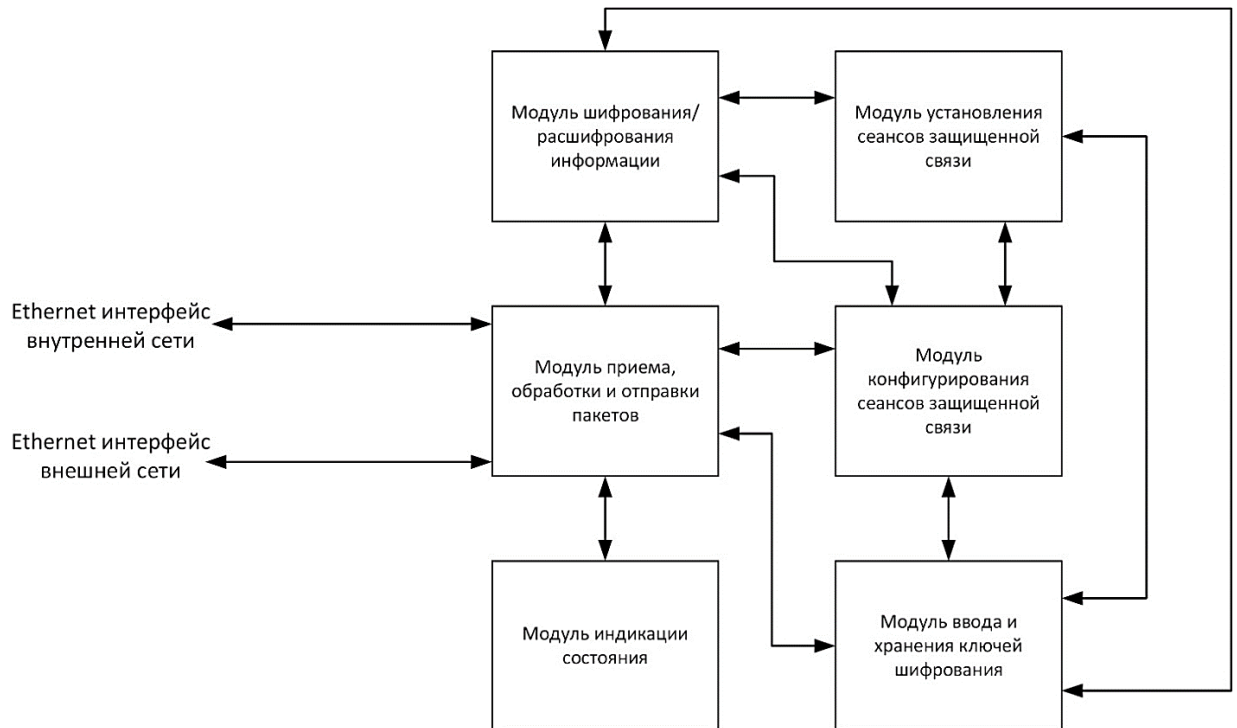


Рисунок 2 – Структура программного комплекса

Модули данного программного комплекса реализованы либо как библиотеки, либо как функции и выполняют следующие задачи:

- модуль приема, обработки и отправки пакетов – реализует прием и отправку пакетов на Ethernet интерфейсах внутренней и внешней сети, обработку пакетов, адресованных программному комплексу, а именно – пакетов протокола ARP, ICMP и ESP, маршрутизацию пакетов между подсетями;

- модуль шифрования/расшифрования информации – реализует функцию шифрования пакетов, маршрутизируемых из внутренней сети во внешнюю и расшифрования зашифрованных пакетов, пришедших из внешней сети;

- модуль установления сеансов защищенной связи – реализует функции обмена параметрами безопасности для выработки сеансовых ключей с другим таким же программным комплексом;

– модуль ввода и хранения ключей шифрования – реализует функцию чтения главного ключа из бинарного файла, его хранение, а также хранение сеансовых ключей каждого сеанса защищенной связи;

– модуль индикации состояния – реализует отображение параметров функционирования комплекса;

– модуль конфигурирования сеансов защищенной связи – реализует функции чтения конфигурационного файла, чтение системных параметров интерфейсов внутренней и внешней сети, хранение текущей конфигурации.

После анализа разработанной выше структуры программного комплекса было определено его место в системе защищенной связи, которое представлено на рисунке 3.

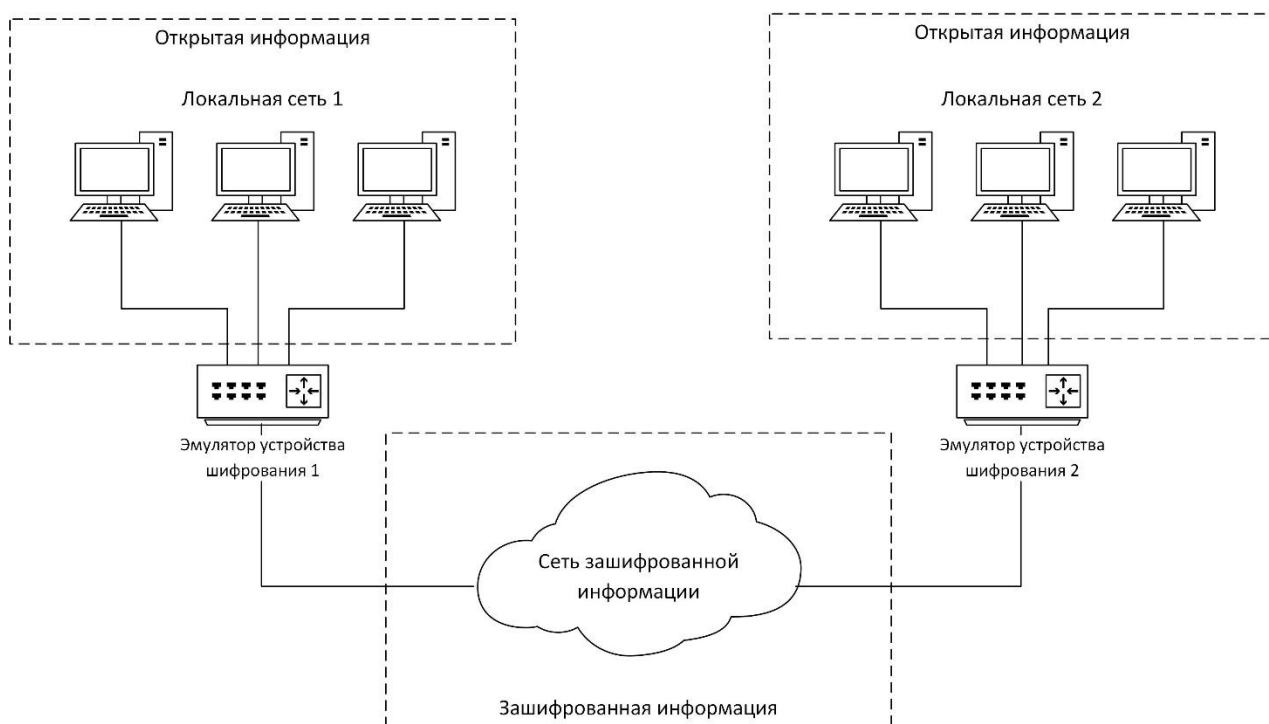


Рисунок 3 – Место эмулируемого устройства IP-шифрования в системе защищенной связи

4 Определение форматов и структур данных

4.1 Формат хранимой конфигурации

Программный комплекс использует следующие управляющие структуры данных:

- таблицу маршрутизации;
- ARP кэш;
- хранилище ключей.

Таблица маршрутизации является массивом записей, содержащих следующие параметры:

- IP адрес подсети;
- маску подсети;
- IP адрес шлюза;
- SPI шлюза.

ARP кэш также является массивом записей, содержащих информацию о адресах какого-либо узла сети, а именно:

- IP адрес узла сети;
- MAC адрес узла сети.

Хранилище ключей представляет собой структуру данных следующего вида:

- а) главный ключ;
- б) массив элементов, каждый из которых содержит:
 - 1) SPI шлюза, с которым установлено защищенное соединение;
 - 2) сеансовый ключ инициатора;
 - 3) сеансовый ключ респондента;
 - 4) авторизационное число инициатора;
 - 5) авторизационное число респондента.

4.2 Формат конфигурационного файла

Конфигурационный файл является файлом формата xml и имеет представленную на рисунке 4 структуру описания параметров конфигурации.

```
<CONFIGURATION>
  <INTERFACES>
    <EXTERNAL> ... </EXTERNAL>
    <INTERNAL> ... </INTERNAL>
  </INTERFACES>
  <KEYFILE> ... </KEYFILE>
  <ROUTETABLE>
    <ROUTE>
      <NETADDR> ... </NETADDR>
      <NETMASK> ... </NETMASK>
      <GATEWAY> ... </GATEWAY>
    </ROUTE>
    ...
    <ROUTE>
      <NETADDR> ... </NETADDR>
      <NETMASK> ... </NETMASK>
      <GATEWAY> ... </GATEWAY>
    </ROUTE>
  </ROUTETABLE>
</CONFIGURATION>
```

Рисунок 4 – Структура конфигурационного файла

В корневом элементе <CONFIGURATION> содержится три элемента:

а) <INTERFACES> - хранит имена сетевых интерфейсов, которые будут использованы для приема и передачи пакетов. В тэге <EXTERNAL> хранится имя интерфейса внешней сети, а в тэге <INTERNAL> имя интерфейса внутренней сети;

б) <KEYFILE> - хранит имя файла с криптографическим ключом;

в) <ROUTETABLE> - хранит параметры маршрутов других подсетей в элементах <ROUTE>, каждый из которых имеет по три элемента:

1) <NETADDR> - IP адрес подсети;

2) <NETMASK> - маска подсети;

3) <GATEWAY> - IP адрес шлюза.

4.3 Формат сообщений при установлении сеанса защищенной связи

Протокол IKE использует передачу сообщений через UDP порты 500 и/или 4500. Алгоритм обмена ключами и аутентификационной информацией по протоколу IKE приведен ниже.

Для создания сеанса инициатор отправляет пакет №1, структура которого представлена в таблице 4.

Таблица 4 – Структура пакета №1

Байты	0	1	2	3	4	5	6	7
Тело пакета	SPI инициатора (случайное число вырабатывается инициатором с ГСЧ)							
	SPI респондента (случайное число вырабатывается респондентом с ГСЧ) При первой отправке = 0							
	Next payload = 43	Version = 32	Exchange type = 34	Flag = 8	Номер сообщения изначально 0			
	Длина сообщения = 68				Next payload = 40	Flags = 128	Len = 12	
	Vendor (тип устройства)							
	Next payload = 35	Falgs = 128	Len = 12		Случайное число			
	инициатора				Next payload = 0	Flags = 128	Len = 16	
	Type = 11	Резерв = 0			Ключ+серия			
	Ключ+серия							

После приема этого пакета респондент проверяет версию протокола IKE, проверяет, что тип устройства находится в списке разрешенных. После этого, если тип обмена совпадает, респондент проверяет номер ключа и серии, при совпадении сохраняет у себя SPI инициатора и случайное число инициатора. Затем отправляет пакет №2, структура которого представлена в таблице 5, для обмена ключами.

Таблица 5 – Структура пакета №2

Байты	0	1	2	3	4	5	6	7
Тело пакета	SPI инициатора (случайное число вырабатывается инициатором с ГСЧ)							
	SPI респондента (случайное число вырабатывается респондентом с ГСЧ)							
	Next payload = 43	Version = 32	Exchange type = 34	Flag = 32	Номер сообщения			
	Длина сообщения = 68				Next payload = 40	Flags = 128	Len = 12	
	Vendor (тип устройства)							
	Next payload = 36	Falgs = 128	Len = 12		Случайное число			
	респондента				Next payload = 0	Flags = 128	Len = 16	
	Type = 11	Резерв = 0			Ключ+серия			
	Ключ+серия							

При получении этого пакета инициатор проверяет версию протокола IKE, проверяет, что тип устройства находится в списке разрешенных. После этого, если тип обмена совпадает, то инициатор проверяет номер ключа и серии, при совпадении сохраняет у себя SPI респондента и случайное число респондента. На основе случайных чисел вырабатываются сеансовые ключи и ключ авторизации. На основе ключа авторизации с помощью значений SPI инициатора и респондента, а также значения `esp_spi` инициатора, полученного с ГСЧ вырабатывается авторизационное число инициатора и отправляется в пакете №3, структура которого представлена в таблице 6, для обмена аутентификационной информацией.

Таблица 6 – Структура пакета №3

Байты	0	1	2	3	4	5	6	7
Тело пакета	SPI инициатора (случайное число вырабатывается инициатором)							
	SPI респондента (случайное число вырабатывается респондентом)							
	Next payload = 43	Version = 32	Exchange type = 35	Flag = 8	Номер сообщения			
	Длина сообщения = 88				Next payload = 40	Flags = 128	Len = 12	
	Vendor (тип устройства)							
	Next payload = 39	Falgs = 128	Len = 24		P_more = 0	P_reseved = 0	P_len = 20	
	P_num = 1	P_id = 3 (ESP)	P_spi_size = 4	P_n_trans = 1	Esp_spi инициатора (с ГСЧ)			
	T_more = 0	T_reserve d = 0	T_len = 8		T_type = 1(encrypt)	T_reserved = 0	T_id = 1024	
	Next payload = 0	Flags = 128	Len = 24		Method = 2 (shared_key)	Резерв = 0		
	Авторизационное число инициатора							

После приема этого пакета респондент проверяет версию протокола ike, проверяет, что тип устройства находится в списке разрешенных. Затем, если тип обмена совпадает, то на основе имеющихся случайных чисел вырабатываются сеансовые ключи и ключ авторизации. На основе ключа авторизации с помощью значений SPI инициатора и респондента, а также значения esp_spi инициатора, полученного из пакета, вырабатывается авторизационное число инициатора, которое сравнивается с принятым в пакете значением. Если сравниваемые числа совпали, то с помощью SPI инициатора и респондента, а также значения esp_spi респондента, полученного с ГСЧ, вырабатывается авторизационное число респондента. Оно отправляется в пакете №4, структура которого представлена в

таблице 7, для обмена аутентификационной информацией, после отправки которого респондент считает сеанс установленным.

Таблица 7 – Структура пакета №4

Байты	0	1	2	3	4	5	6	7
Тело пакета	Spi инициатора (случайное число вырабатывается инициатором)							
	Spi респондента (случайное число вырабатывается респондентом)							
	Next payload = 43	Version = 32	Exchange type = 35	Flag = 32	Номер сообщения			
	Длина сообщения = 88				Next payload = 40	Flags = 128	Len = 12	
	Vendor (тип устройства)							
	Next payload = 39	Falgs = 128	Len = 24		P_more = 0	P_reseved = 0	P_len = 20	
	P_num = 1	P_id = 3 (ESP)	P_spi_size = 4	P_n_trans = 1	Esp_spi респондента (с ГСЧ)			
	T_more = 0	T_reserved = 0	T_len = 8		T_type = 1(encrypt)	T_reserved = 0	T_id = 1024	
	Next payload = 0	Flags = 128	Len = 24		Method = 2 (shared_key)	Резерв = 0		
	Авторизационное число респондента							

После приема этого пакета инициатор проверяет версию протокола ike, проверяет, что тип устройства находится в списке разрешенных. Затем, если тип обмена совпадает, то на основе имеющихся случайных чисел вырабатываются сеансовые ключи и ключ авторизации. На основе ключа авторизации с помощью значений SPI инициатора и респондента, а также значения esp_spi респондента, полученного из пакета, вырабатывается авторизационное число респондента, которое сравнивается с принятым в пакете значением. Если сравниваемые числа совпали, то инициатор считает сеанс установленным.

Для создания сеансовых ключей и ключей авторизации составляется последовательность данных, представленная в таблице 8.

Таблица 8 – Последовательность данных для расчета ключей авторизации

Байты	0-7	8-15	16-23						
Данные	Случайное число инициатора	Случайное число респондента	0	0	0	0	0	0	0

На эту последовательность, длиной 24 байта рассчитывается имитовставка, по алгоритму, приведенному в национальном стандарте ГОСТ 28147-89 «Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования» [11], ICV(1) на ключе направления, которая является первыми 32 битами сеансового ключа.

Для последующих байт ключа (имитовставки ICV(2), ICV(3)...ICV(n-1), ICV(n)) последовательность данных для расчета имитовставки представлена в таблице 9.

Таблица 9 – Последовательность данных для расчета ключей авторизации

Байты	0-3	4-11	12-19	20-23			
Данные	ICV(n-1)	Случайное число инициатора	Случайное число респондента	0	0	0	0

Таким образом рассчитываются все необходимые для установления сеанса связи ключи, представленные в таблице 10.

Таблица 10 – Вычисленные ключи

Байты	0-31							
Сеансовый ключ инициатора	ICV(1)	ICV(2)	ICV(3)	ICV(4)	ICV(5)	ICV(6)	ICV(7)	ICV(8)
Сеансовый ключ респондента	ICV(9)	ICV(10)	ICV(11)	ICV(12)	ICV(13)	ICV(14)	ICV(15)	ICV(16)
Авторизационный ключ инициатора	ICV(17)	ICV(18)	ICV(19)	ICV(20)	ICV(21)	ICV(22)	ICV(23)	ICV(24)
Авторизационный ключ респондента	ICV(25)	ICV(26)	ICV(27)	ICV(28)	ICV(29)	ICV(30)	ICV(31)	ICV(32)

Для создания авторизационного числа инициатора составляется последовательность данных, представленная в таблице 11.

Таблица 11 – Последовательность данных для расчета авторизационного числа инициатора

Байты	0-7	8-15	16-19	20-23			
Данные	SPI инициатора	SPI респондента	Esp_spi инициатора	0	0	0	1

На эту последовательность длиной 24 байта рассчитывается имитовставка ICV(1) на ключе авторизации инициатора, выработанном на шаге выработки сеансовых ключей, которая является первыми 32 битами авторизационного числа инициатора.

Для последующих байт (имитовставки ICV(2), ICV(3)... ICV(n-1), ICV(n)) последовательность данных для расчета имитовставки имеет длину 32 байта, данная последовательность представлена в таблице 12.

Таблица 12 – Последовательность данных для расчета авторизационного числа инициатора

Байты	0-3	4-11	12-19	20-23	24-31							
Данные	ICV(n-1)	SPI инициатора	SPI респондента	Esp_spi инициатора	0	0	0	n+1	0	0	0	0

Вычисленное авторизационное число инициатора представлено в таблице 13.

Таблица 13 – Вычисленное авторизационное число инициатора

Байты	0-15			
Данные	ICV(1)	ICV(2)	ICV(3)	ICV(4)

Для создания авторизационного числа респондента составляется последовательность данных, представленная в таблице 14.

Таблица 14 – Последовательность данных для расчета авторизационного числа инициатора

Байты	0-7	8-15	16-19	20-23			
Данные	SPI инициатора	SPI респондента	Esp_spi респондента	0	0	0	5

На эту последовательность длиной 24 байта рассчитывается имитовставка ICV(1) на ключе авторизации респондента, выработанном на шаге выработки сеансовых ключей, которая является первыми 32 битами авторизационного числа респондента.

Для последующих байт (имитовставки ICV(2), ICV(3)... ICV(n-1), ICV(n)) последовательность данных для расчета имитовставки имеет длину 32 байта, данная последовательность представлена в таблице 15.

Таблица 15 – Последовательность данных для расчета авторизационного числа респондента

Байты	0-3	4-11	12-19	20-23	24-31							
Данные	ICV(n-1)	SPI инициатора	SPI респондента	Esp_spi респондента	0	0	0	n+5	0	0	0	0

Вычисленное авторизационное число респондента представлено в таблице 16.

Таблица 16 – Вычисленное авторизационное число респондента

Байты	0-15			
Данные	ICV(1)	ICV(2)	ICV(3)	ICV(4)

4.4 Формат пакетов в процессе шифрования и расшифрования

Эмулятор принимает из внутренней сети незашифрованный пакет, который имеет структуру, представленную в таблице 17.

Таблица 17 – Структура незашифрованного пакета

Байты	0	1	2	3	4	5	6	7
0	Ethernet заголовок							
8								
16	IP заголовок							
24								
32	Данные IP пакета							
...								
...								

После получения незашифрованного пакета, он подготавливается к шифрованию путем удаления Ethernet заголовка, пакет подготовленный к шифрованию имеет структуру, представленную в таблице 18.

Таблица 18 – Структура пакета, подготовленного к шифрованию

Байты	0	1	2	3	4	5	6	7
0	IP заголовок							
8								
16								
24	Данные IP пакета							
...								

После подготовки пакет шифруется и инкапсулируется в поле данных заголовка ESP пакета, пакет после шифрования имеет структуру, представленную в таблице 19.

Таблица 19 – Структура пакета после шифрования

Байты	0	1	2	3	4	5	6	7
0	ESP заголовок							
8	Зашифрованный IP пакет							
...								
...								
...	Дополнение							
...			Pading Length	Next header	Авторизационное число			

Для отправки в другую подсеть зашифрованный пакет инкапсулируется в IP пакет, адрес назначения которого будет являться IP адресом шлюза, за которым находится нужная подсеть, затем этот пакет инкапсулируется в Ethernet

кадр, адрес назначения которого будет являться MAC адресом шлюза, структура выходного пакета представлена в таблице 20.

Таблица 20 – Структура выходного пакета

Байты	0	1	2	3	4	5	6	7
0	Ethernet заголовок							
8								
16	IP заголовок							
24								
32	ESP							
...								
...	Зашифрованный IP пакет							
...								
...	Дополнение							
...								
...	заголовок							
...			Pading Length	Next header	Авторизационное число			

После приема такого пакета шлюз его расшифровывает и отбрасывает все заголовки кроме исходного IP заголовка, после проведения данных операций пакет принимает исходный вид, структура расшифрованного пакета представлена в таблице 21.

Таблица 21 – Структура расшифрованного пакета

Байты	0	1	2	3	4	5	6	7
0	IP заголовок							
8								
16								
...	Данные IP пакета							

Для отправки пакета адресату шлюз инкапсулирует расшифрованный пакет в Ethernet кадр, где адресом назначения будет MAC адрес узла с IP адресом,

указанным в поле адрес назначения IP заголовка расшифрованного пакета, структура пакета, отправляемого адресату шлюзом представлена в таблице 22.

Таблица 22 – Структура пакета, отправляемого адресату шлюзом

Байты	0	1	2	3	4	5	6	7
0	Ethernet заголовок							
8								
16	IP заголовок							
24								
32	Данные IP пакета							
...								

5 Разработка алгоритмов модулей программного комплекса

Алгоритм обработки входящих пакетов представлен на рисунке 5.

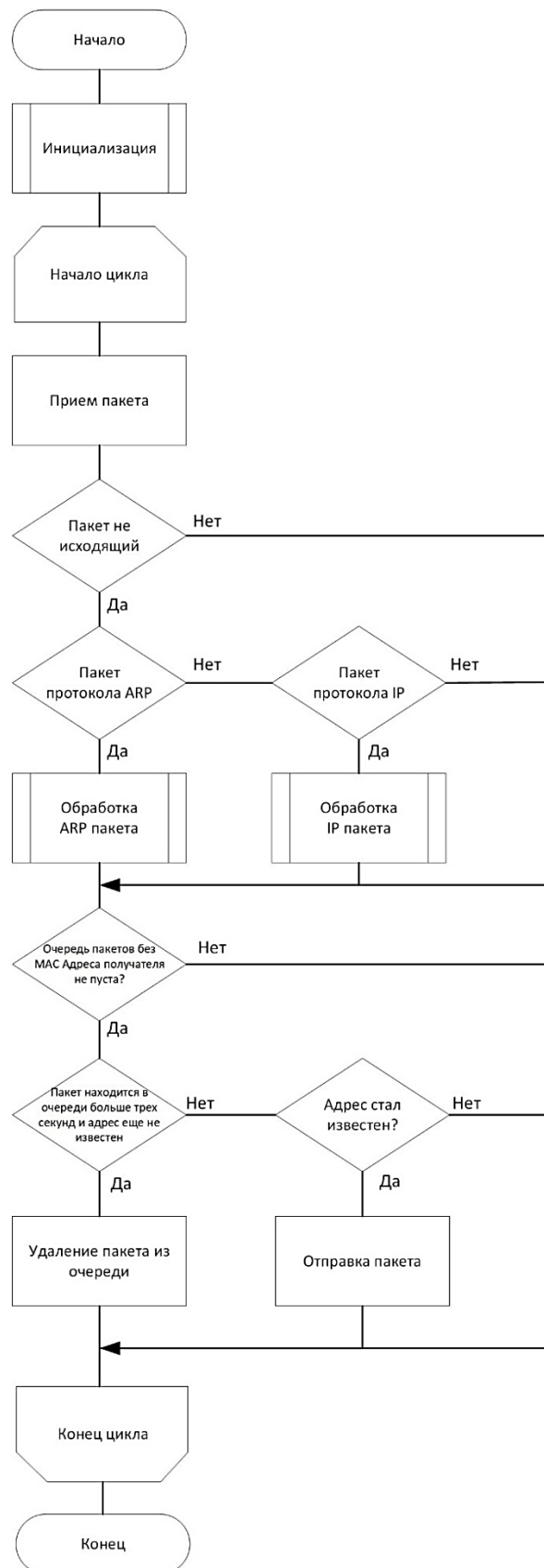


Рисунок 5 – Алгоритм обработки входящих пакетов

При запуске программного комплекса производится инициализация всех его параметров, если она не завершилась с ошибкой, то начинается прием входящих пакетов протоколов ARP и IP с дальнейшей их обработкой, по завершении обработки принятого пакета производится проверка очереди пакетов, для которых не известен MAC адрес назначения, если он был найден, то пакет отправляется по назначению, а если пакет находится в таблице слишком долго и MAC все еще не известен, то пакет удаляется из очереди.

Алгоритм инициализации параметров программного комплекса представлен на рисунках 6 и 7.



Рисунок 6 – Алгоритм инициализации параметров программного комплекса



Рисунок 7 – Алгоритм инициализации параметров программного комплекса

При инициализации считывается имя конфигурационного файла, который содержит все необходимые для функционирования программного комплекса параметры, параметры сетевых интерфейсов инициализируются системными значениями, если данные параметры не определены программный комплекс аварийно завершает работу.

Алгоритм обработки ARP пакетов представлен на рисунке 8.

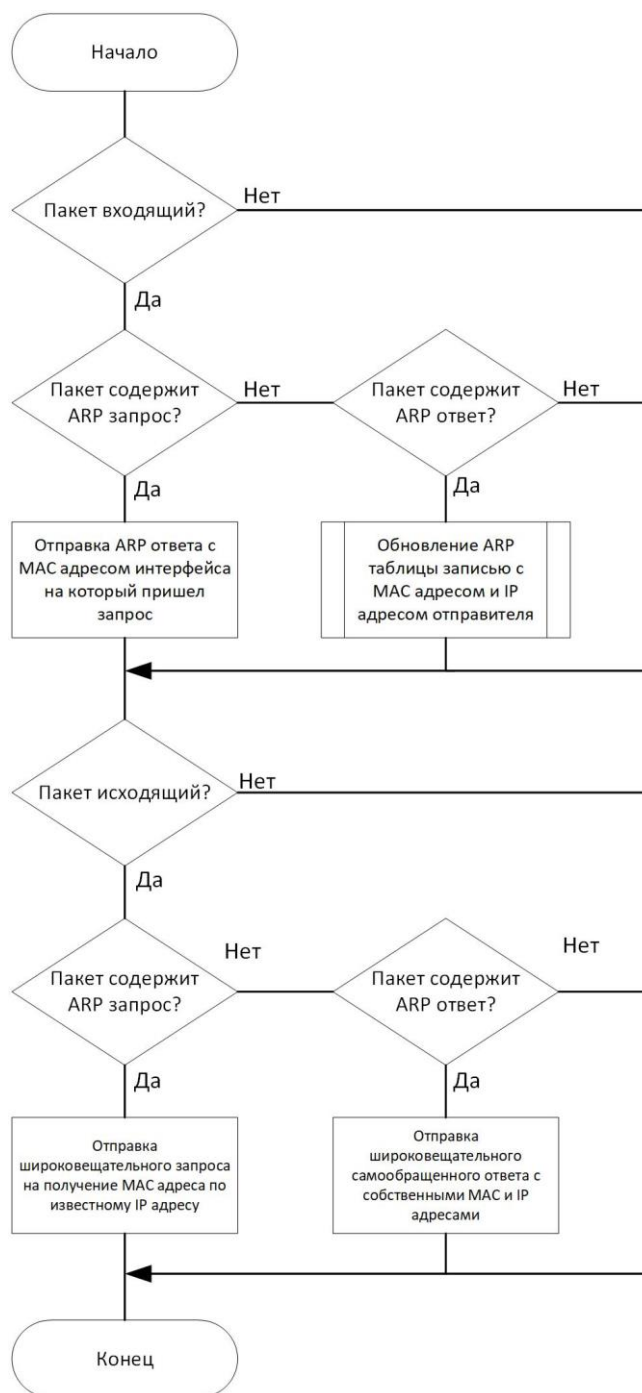


Рисунок 8 – Алгоритм обработки ARP пакетов

Обработка ARP пакета осуществляется следующим образом – если пакет входящий и это ARP-запрос, то на него необходимо собрать ответ и отправить, иначе если это ARP-ответ, то его данными необходимо обновить ARP кэш, но если пакет исходящий, значит какой-либо модуль программного комплекса решил послать запрос, чтобы узнать неизвестный MAC адрес, либо выполнить самообращенный ответ для оповещения всех хостов сети о том, что программный комплекс функционирует.

Алгоритм обновления ARP кэша представлен на рисунке 9.



Рисунок 9 – Алгоритм обновления ARP кэша

Для обновления ARP кэша необходима пара значений – IP и MAC адреса, затем данными значениями либо обновляется старая запись, либо они добавляются как новая запись.

Алгоритм обработки IP пакетов представлен на рисунках 10 и 11.



Рисунок 10 – Алгоритм обработки IP пакетов

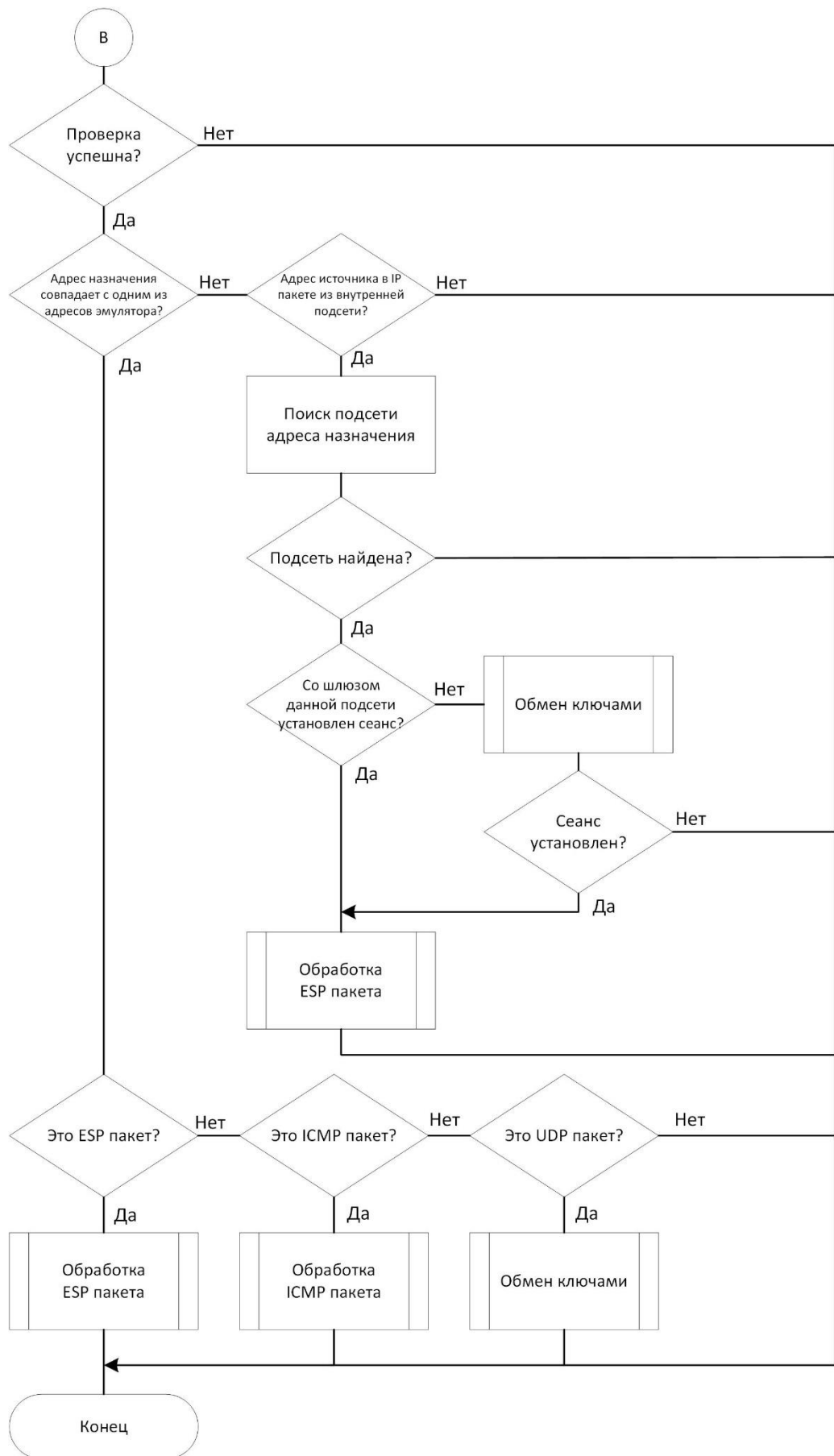


Рисунок 11 – Алгоритм обработки IP пакетов

Для обработки IP пакета необходимо получить сам IP пакет и доступ к хранилищу ключей. После того как пакет поступил в обработку проверяется его контрольная сумма, если она некорректна – пакет отбрасывается, затем происходит определение на каков адрес назначения пакета, если это пакет для программного комплекса, то производится проверка протокола IP пакета, если это ESP, ICMP или UDP, то пакет проходит на дальнейшую обработку, иначе он отбрасывается. Если принятый пакет не предназначается программному комплексу то производится поиск подсети назначения в таблице маршрутизации, если подсеть найдена и с ее шлюзом установлен сеанс защищенной связи, пакет маршрутизируется в эту подсеть, иначе начинается процесс установления сеанса, а пакет отбрасывается, также пакет отбрасывается в случае, если подсеть не была найдена.

Алгоритм обработки ICMP пакетов представлен на рисунках 12 и 13.

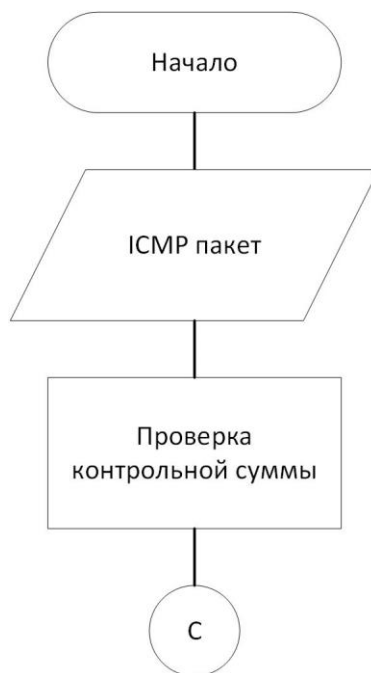


Рисунок 12 – Алгоритм обработки ICMP пакетов



Рисунок 13 – Алгоритм обработки ICMP пакетов

Обработка ICMP пакета производится следующим образом – у принятого пакета проверяется его контрольная сумма, если проверка завершилась успешно и тип пакета ICMP_ECHO, тип изменяется ICMP_ECHOREPLY, пересчитывается контрольная сумма и пакет отправляется обратно источнику.

Алгоритм обработки ESP пакетов представлен на рисунке 14.

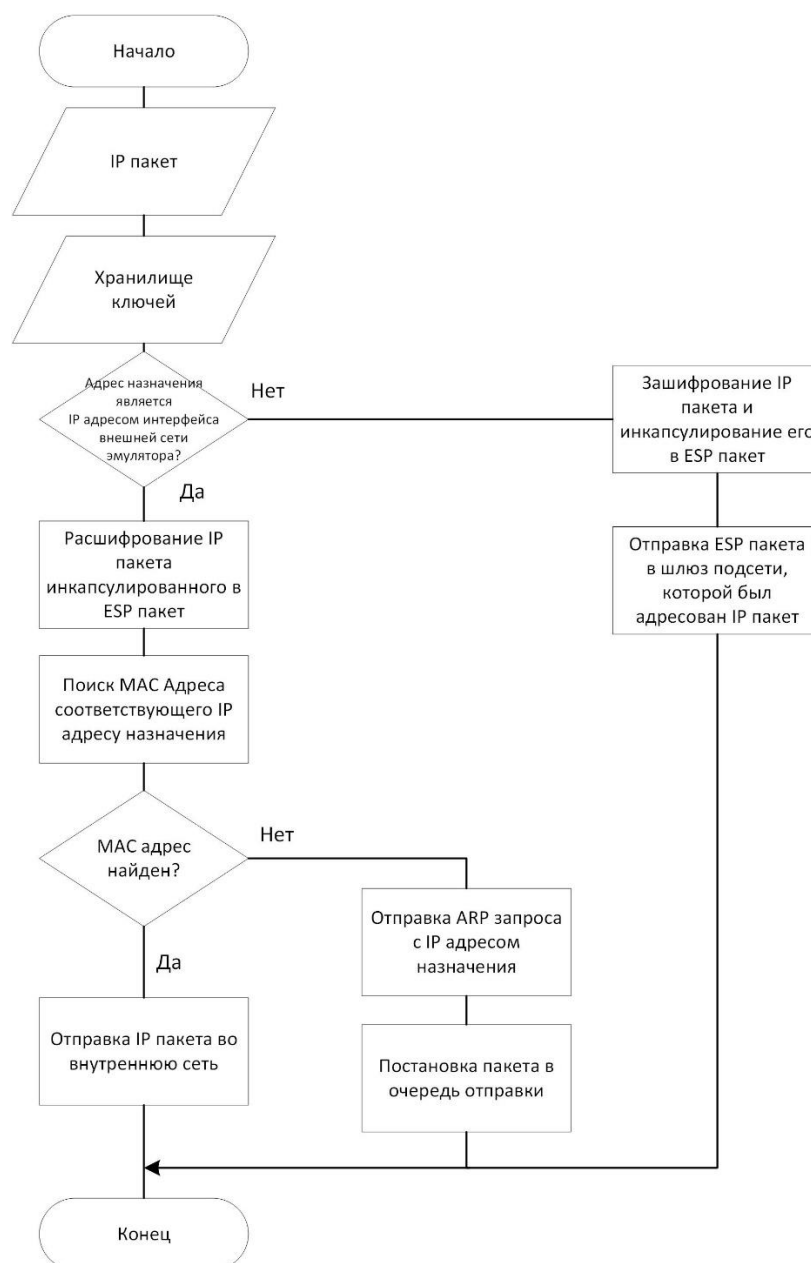


Рисунок 14 – Алгоритм обработки ESP пакетов

Обработка пакетов, маршрутизируемых во внешнюю сеть состоит в том, чтобы зашифровать исходный IP пакет, инкапсулировать его в ESP заголовок, который с свою очередь будет инкапсулирован для передачи пакета шлюзу сети назначения.

Наоборот, обработка пакетов, принимаемых из внешней сети заключается в отбрасывании всех лишних заголовков, расшифровке исходного пакета и отправке его во внутреннюю сеть.

Алгоритм обмена ключами для установления сеанса защищенной связи соответствует алгоритму, описанному в пункте 4.3.

6 Реализация разработанных алгоритмов программных модулей

6.1 Реализация алгоритма обработки входящих пакетов

Для реализации разработанных алгоритмов были использованы стандартный функционал библиотек ОС Linux для сетевого взаимодействия, в данном случае использовались пакетные сокеты канального уровня, и функционал стандартных библиотек языка C++, фрагмент кода в котором подключаются данные библиотеки, представлен ниже.

```
#include <algorithm>
#include <iostream>
#include <utility>
#include <fstream>
#include <vector>
#include <random>
#include <string>
#include <map>
#include <cstring>
#include <cstdlib>
#include <climits>
#include <ctime>
#include <sys/socket.h>
#include <linux/types.h>
#include <linux/if_ether.h>
#include <linux/if_packet.h>
#include <netinet/in.h>
#include <linux/ip.h>
#include <linux/if_arp.h>
#include <linux/icmp.h>
#include <linux/if.h>
#include <linux/udp.h>
#include <arpa/inet.h>
#include <unistd.h>
```

```
#include <sys/ioctl.h>
```

Представленные выше библиотеки позволяют удобно манипулировать необходимыми структурами данных, используемых как для сетевого взаимодействия, так и для хранения параметров, которые требуются программному комплексу для функционирования.

Перед началом приема входящих пакетов программный комплекс проходит процедуру инициализации, описанную в пункте 6.2, в которой создается прослушивающий сокет listener. Данный сокет используется для приема всех пакетов канального уровня, адресованных программному комплексу.

Для приема входящих пакетов используется функция `recvfrom()`, представленная ниже.

```
recvfrom(listener, (char *)&buffer[0], ETH_FRAME_LEN, 0,  
NULL, NULL);
```

Она принимает пакеты канального уровня и записывает их в массив байт `buffer`, после получения пакета проверяются его MAC адреса и если пакет не исходящий то проверяется его протокол и если это пакет протокола ARP или IP то он передается на дальнейшую обработку в функции для работы с этими протоколами, а именно – `ARP_packet_processing()` или `IP_packet_processing()`, иначе он будет отброшен.

После данных операций проверяется очередь сообщений, у которых отсутствует MAC адрес и если он был найден пакет будет отправлен.

Затем управление передается модулю индикации, который каждые 10 секунд отправляет уведомление о количестве обработанных пакетов, фрагмент кода реализующий данные операции представлен ниже.

```
if(((std::clock() - timer) / 1000) > 10) {  
    std::string message("обработано ");  
    message.append(std::to_string( (Internal.IP_count +  
External.IP_count) * 2 ));  
    message.append(" пакета/ов");
```

```

        notify_notification_show(notify_notification_new
("Эмулятор", message.c_str(), 0), 0);
        timer = std::clock();
    }

```

Затем снова ожидается прием входящих пакетов.

6.2 Реализация алгоритма инициализации программного комплекса

Инициализация программного комплекса проходит в функции Initialization(), которая занимается чтением параметров из конфигурационного файла, а именно имен внешнего и внутреннего сетевых интерфейсов, фрагмент кода представлен ниже.

```

NotifyNotification *n;
tinyxml2::XMLDocument conf;
conf.LoadFile("configuration.xml");

tinyxml2::XMLElement *configuration, *interfaces,
*keyfile, *routetable;

configuration = conf.FirstChildElement("CONFIGURATION");
if (!configuration) exit(EXIT_FAILURE);

interfaces = configuration-
>FirstChildElement("INTERFACES");
if (!interfaces) exit(EXIT_FAILURE);

std::string ifaces[2]; //инициализация имен интерфейсов
ifaces[0].append(interfaces-
>FirstChildElement("EXTERNAL")->GetText(), strlen(interfaces-
>FirstChildElement("EXTERNAL")->GetText()));

ifaces[1].append(interfaces-
>FirstChildElement("INTERNAL")->GetText(), strlen(interfaces-
>FirstChildElement("INTERNAL")->GetText()));

```

Затем создается пакетный сокет и производится вызов системных функций `ioctl()` для получения сетевых адресов каждого интерфейса. Полученные параметры записываются в переменные `External` и `Internal`, имеющие тип `IConf`, который является структурой, описанной в представленном ниже фрагменте кода.

```
struct IConf {
    unsigned char MAC[6];
    unsigned int IP;
    unsigned int netmask;
    unsigned int netaddr;
    int socket;
    sockaddr_ll s_ll;
    unsigned int SPI;
    unsigned short IP_count;
};
```

После этого для каждого из указанного в конфигурации сетевого интерфейса создается и привязывается свой сокет, для возможности дальнейшего взаимодействия с двумя интерфейсами сразу, сокет и его параметры также записываются в переменные `External` и `Internal`.

Если сокеты будут созданы успешно производится загрузка главного ключа в программный комплекс, фрагмент кода представлен ниже.

```
key_file.read( (char *) &Key_storage.main_key[0],
(size_t)(sizeof(unsigned long int) * 8));
```

Для хранения главного ключа и сеансовых ключей, которые будут вычисляться в процессе функционирования программного комплекса была создана структура данных, представленная во фрагменте кода ниже. В поле `main_key` данной структуры хранится главный ключ, а в поле `generated_keys` сеансовые ключи.

```
struct Key_bundle {
    struct Keys {
        struct Key_set {
```

```

        std::vector <unsigned long int> session_key;
        std::vector      <unsigned      long      int>
autentification_key;
        std::vector      <unsigned      long      int>
authorization_number;
    };
    bool flag;
    Key_set local;      Key_set outside;
    int exchange
    unsigned long long Initiators_random_number;
    unsigned long long Initiators_SPI;
    unsigned long long Respondents_random_number;
    unsigned long long Respondents_SPI;
};
std::vector <unsigned long int> main_key;
unsigned long int number;
unsigned long long int serial_number;
std::map < unsigned int, Keys > generated_keys;
};

```

После загрузки главного ключа загружается таблица маршрутизации, путем чтения ее из конфигурационного файла, фрагмент кода реализующий загрузку представлен ниже.

```

    routetable      =      configuration->FirstChildElement
("ROUTETABLE");

    if (!routetable) exit(EXIT_FAILURE);

    for (const tinyxml2::XMLElement * route = routetable-
>FirstChildElement("ROUTE");    route;    route    =    route-
>NextSiblingElement("ROUTE")) {
        const    tinyxml2::XMLElement    *netaddr,    *netmask,
*gateway;

        netaddr = route->FirstChildElement("NETADDR");
        netmask = route->FirstChildElement("NETMASK");

```

```

gateway = route->FirstChildElement("GATEWAY");
if ( !(netaddr && netmask && gateway) ) continue;
Route_table.push_back(Route(inet_addr(netaddr-
>GetText()), inet_addr(netmask->GetText()), inet_addr(gateway-
>GetText())));

```

Если все описанные выше параметры проинициализированы успешно, управление передается снова передается модулю приема и обработки входящих пакетов.

6.3 Реализация алгоритма обработки ARP пакетов

Обработка ARP пакетов реализована следующим образом – функции обработки ARP пакетов на вход подается принятый пакет затем она определяет запрос это или ответ путем чтения ARP заголовка, и в зависимости от типа принятого пакета либо генерирует ответ, либо обновляет ARP кэш, а если данная функция вызывается каким-либо модулем, то только для формирования запроса и его отправки с целью узнать необходимый MAC адрес.

6.4 Реализация алгоритма обновления ARP кэша

Алгоритм обновления ARP кэша реализован путем добавления в таблицу полученных адресов, а та в свою очередь сама решает обновить старую запись либо добавить новую, фрагмент кода представлен ниже.

```

ARP_cache[ip].resize(ETH_ALEN);
std::copy(MAC, MAC + ETH_ALEN, ARP_cache[ip].begin());

```

6.5 Реализация алгоритма обработки IP пакетов

Принятый IP пакет проходит проверку контрольной суммы, если он не был отброшен, то проверяется совпадает ли его адрес назначения с одним из адресов программного комплекса, если совпадение найдено, то пакет передается в обработку в одну из функций для обработки ICMP, ESP или UDP пакетов, если пакет имеет иной заголовок, то пакет будет отброшен.

Если определяется что данный пакет отправляется из внутренней сети во внешнюю, то производится поиск данной подсети в таблице маршрутизации фрагментом кода, представленным ниже.

```
for (Route &route: Route_table) {
    if (Subnet_check(ip->daddr, route.netmask, route.netaddr))
```

Если подсеть будет найдена и с ней будет установлен сеанс, то пакет будет отправлен в данную подсеть, иначе пакет будет отброшен и со шлюзом данной подсети начнется обмен ключами.

6.6 Реализация алгоритма обработки ICMP пакетов

Принятый ICMP пакет проходит проверку контрольной суммы, если проверка будет пройдена успешно, то производится проверка типа ICMP пакета, и если пакет требуемого типа, а именно ICMP_ECHO, то его тип меняется на ICMP_ECHOREPLY, производится перерасчет контрольных сумм, пакет перенаправляется отправителю и отправляется в сокет интерфейса, из которого пришел данный пакет. Данные операции представлены ниже, в фрагменте кода.

```
if (icmp_chksum == Internet_checksum( (unsigned short
*)icmp, recieved_bytes - (sizeof(ethhdr)+(ip->ihl * 4)) ) ) {
    if (icmp->type == ICMP_ECHO ) {
        icmp->type = ICMP_ECHOREPLY;
        icmp->checksum = Internet_checksum( (unsigned
short *)icmp, recieved_bytes - (sizeof(ethhdr) + (ip->ihl * 4))
);

        ip->check = Internet_checksum( (unsigned short
*)ip, recieved_bytes - (sizeof(ethhdr)) );
        //L2
        memcpy(eth->h_dest, eth->h_source, ETH_ALEN);
        memcpy(eth->h_source, Conf.MAC, ETH_ALEN);
        write(Conf.socket, (void *) &buffer[0],
recieved_bytes);
    }
}
```

6.7 Реализация алгоритма обработки ESP пакетов

Функция обработки ESP пакетов принимает на вход IP пакет, затем создает объект класса Cypher, который в свою использует класс GOST_28147_89 для функционирования. Данный объект используется для обеспечения функций шифрования и расшифрования данных. Класс Cypher является абстрактным, что обеспечивает возможность использования им любых шифров которые поддерживают методы ввода ключа, шифрование и расшифрование, в данном случае используется в соответствии с национальным стандартом ГОСТ 28147-89 «Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования» [11], который реализован в классе GOST_28147_89. Реализация класса Cypher представлена во фрагменте кода ниже.

```
class Cypher{
public:
    virtual void SetKey( const unsigned char * key, const
unsigned int &key_size ) = 0;
    virtual void Encrypt( const unsigned char *
open_buffer, const unsigned int open_buffer_size, unsigned char
* encrypted_buffer ) = 0;
    virtual void Decrypt( const unsigned char *
encrypted_buffer, unsigned char * open_buffer, const unsigned
int open_buffer_size ) = 0;
    virtual ~Cypher(){};
};
```

После инициализации объекта, обеспечивающего функции шифрования и расшифрования производится проверка из какой сети пришел пакет.

Если принятый пакет пришел из внутренней сети, то производится поиск шлюза подсети назначения, затем инициализируется сеансовый ключ шифрования, рассчитанный в процессе обмена ключами с данным шлюзом, затем принятый пакет зашифровывается, рассчитывается его контрольная сумма

и пакет отправляется в шлюз подсети назначения. Данные операции реализованы во фрагменте кода, представленном ниже.

```
cypher->SetKey(          (unsigned          char          *)
&Key_storage.generated_keys[route.SPI].local.session_key[0],
//Данные ключа
Key_storage.generated_keys[route.SPI].local.session_key.s
ize() * 4); //Его размер в байтах
unsigned short encrypted_bytes = htons(ip_recieved-
>tot_len) + (8 - (htons(ip_recieved->tot_len)%8));
cypher->Encrypt((const unsigned char *)ip_recieved,
encrypted_bytes, esp->enc_data); //Зашифровывание принятого ip
пакета и добавление его
send_ip->tot_len = htons(sizeof(iphdr) + sizeof(esp-
>seq_no) + sizeof(esp->spi) + encrypted_bytes);
send_ip->check = Internet_checksum( (unsigned short
*)send_ip, htons(send_ip->tot_len));
//L2
memcpy(send_eth->h_dest,          &ARP_cache[gateway_ip][0],
ETH_ALEN);
memcpy(send_eth->h_source, Conf.MAC, ETH_ALEN);
send_eth->h_proto = htons(ETH_P_IP);
write(Conf.socket,          (void          *)&send_buffer[0],
sizeof(ethhdr) + htons(send_ip->tot_len));
```

Если принятый пакет пришел на внешний интерфейс, производится проверка из какой подсети принятый пакет, инициализируется сеансовый ключ расшифрования, рассчитанный в процессе обмена ключами со шлюзом подсети из которой пришел пакет, затем пакет расшифровывается и отбрасываются лишние заголовки. После расшифрования в ARP кэше производится поиск MAC адреса узла которому адресован расшифрованный пакет, если адрес найден, пакет отправляется адресату, иначе во внутреннюю сеть отправляется ARP запрос для получения MAC адреса адресата и расшифрованный пакет добавляется в очередь пакетов с отсутствующим MAC адресом назначения.

6.8 Реализация алгоритма обмена ключами

В процессе обмена ключами используется протокол UDP, прием и прослушивание производится на порту с номером 4500.

При начале обмена ключами инициатор отправляет респонденту пакет, содержащий SPI инициатора и его случайное число, данный процесс представлен ниже во фрагменте кода.

```
kepacket->RSPIInit = uid(rd);
kepacket->GRandom_number_init = uid(rd);
write(Conf.socket, (void *)&buffer[0], sizeof(ethhdr) +
hton(ip->tot_len));
```

При получении данного пакета респондент проверяет корректность принятого пакета, сохраняет полученные значения и отправляет инициатору пакет содержащий SPI респондента и его случайное число, данные действия представлены во фрагменте кода ниже.

```
Key_storage.generated_keys[SPI].Initiators_SPI =
kepacket->RSPIInit;
Key_storage.generated_keys[SPI].Initiators_random_number
= kepacket->GRandom_number_init;
Key_storage.generated_keys[SPI].Respondents_SPI= uid(rd);
Key_storage.generated_keys[SPI].Respondents_random_number
= uid(rd);
kepacket->RSPIResp =
Key_storage.generated_keys[SPI].Respondents_SPI;
kepacket->GRandom_number_init =
Key_storage.generated_keys[SPI].Respondents_random_number;
write(Conf.socket, (void *)&buffer[0], sizeof(ethhdr) +
hton(ip->tot_len));
```

После получения данного пакета инициатор проверяет его корректность и сохраняет полученные значения.

Затем респондент и инициатор рассчитывают сеансовые ключи на полученных значениях. Процесс расчета сеансовых ключей реализуется во фрагменте кода представленном ниже.

```
Creating_a_session_keys
(Key_storage.generated_keys[SPI].local.session_key,
  Key_storage.generated_keys[SPI].outside.session_key,
  Key_storage.generated_keys[SPI].local.authentication_ke,
  Key_storage.generated_keys[SPI].outside.authentication_key,
  Key_storage.main_key,
  Key_storage.generated_keys[SPI].Initiators_random_number,
  Key_storage.generated_keys[SPI].Respondents_random_number);
```

После расчета сеансовых ключей инициатор рассчитывает ключ авторизации, при использовании которого вычисляется авторизационное число инициатора и отправляет респонденту пакет со своим авторизационным числом и Esp_spi, описанные действия реализованы в представленном ниже фрагменте кода.

```
Key_storage.generated_keys[SPI].local.authorization_number
= Calculate_authentication_number(
  Key_storage.generated_keys[SPI].Initiators_SPI,
  Key_storage.generated_keys[SPI].Respondents_SPI,      Conf.SPI,
  Key_storage.generated_keys[SPI].local.authentication_key);
for (size_t i = 0; i < Key_storage.generated_keys
[SPI].local.authorization_number.size());
  aipacket->BAuthorization[i] = Key_storage.generated_keys
[SPI].local.authorization_number[i], i++ );
write(Conf.socket, (void *)&buffer[0], sizeof(ethhdr) +
htons(ip->tot_len));
```

Приняв данный пакет респондент проверяет его корректность, сохраняет полученное авторизационное число инициатора и его Esp_spi, затем авторизационные ключи инициатора и респондента, вычисляется авторизационное число инициатора и сравнивается с полученным значением,

если они совпадают, сеанс считается установленным и вычисляется авторизационное число респондента и отправляется инициатору в пакете, который также содержит Esp_spi респондента. Данный процесс реализован в представленном ниже фрагменте кода.

```

    Key_storage.generated_keys[SPI].outside.authorization_number = Calculate_authentication_number(
        Key_storage.generated_keys[SPI].Initiators_SPI,
        Key_storage.generated_keys[SPI].Respondents_SPI,
        aipacket->GEsp_spi,
        Key_storage.generated_keys[SPI].outside.authentication_key);

    for ( size_t i = 0; i < Key_storage.generated_keys[SPI].outside.authorization_number.size(); i++ )
        if ( Key_storage.generated_keys[SPI].outside.authorization_number[i] != aipacket->BAuthorization[i] ) {
            Key_storage.generated_keys[SPI].flag = false;
            Key_storage.generated_keys[SPI].exchange = -1;
            return;
        }

    Key_storage.generated_keys[SPI].flag = true;
    Key_storage.generated_keys[SPI].local.authorization_number = Calculate_authentication_number(
        Key_storage.generated_keys[SPI].Initiators_SPI,
        Key_storage.generated_keys[SPI].Respondents_SPI,
        Conf.SPI,
        Key_storage.generated_keys[SPI].local.authentication_key);

    for ( size_t i = 0; i < Key_storage.generated_keys[SPI].local.authorization_number.size(); aipacket->BAuthorization[i] = Key_storage.generated_keys[SPI].local.authorization_number[i], i++ );

```

```
write(Config.socket, (void *)&buffer[0], sizeof(ethhdr) +
htons(ip->tot_len));
```

После получения данного пакета инициатор рассчитывает ключ авторизации респондента и вычисляет его авторизационное число, если оно совпадает с принятым сеанс считается установленным, это реализовано в представленном ниже фрагменте кода.

```
Key_storage.generated_keys[SPI].outside.authorization_number =
Calculate_authentication_number(
    Key_storage.generated_keys[SPI].Initiators_SPI,
    Key_storage.generated_keys[SPI].Respondents_SPI,
    aipacket->GEsp_spi,
    Key_storage.generated_keys[SPI].outside.authentication_key);
for ( size_t i = 0; i < Key_storage.generated_keys
[SPI].outside.authorization_number.size(); i++ )
    if ( Key_storage.generated_keys
[SPI].outside.authorization_number[i] != aipacket-
>BAuthorization[i]) {
    Key_storage.generated_keys[SPI].flag = false;
    Key_storage.generated_keys[SPI].exchange = -1;
    return;
}
Key_storage.generated_keys[SPI].flag = true;
```

Исходный код данного программного комплекса представлен в Приложении А.

7 Тестирование разработанного программного комплекса

Для тестирования программного комплекса были использованы 2 виртуальные машины под управлением ОС Linux, имеющие по 2 сетевых интерфейса, каждой машине функционирует программный комплекс эмуляции устройства шифрования (далее эмулятор). Стандартные сетевые службы, такие как ping, arp на данных виртуальных машинах были отключены так как эмулятор самостоятельно выполняет их функции.

Эмулятор 1 имеет следующие сетевые адреса:

- внутренний интерфейс – IP адрес 192.168.56.2/24;
- внешний интерфейс – IP адрес 192.168.1.56/24.

Эмулятор 2 имеет следующие сетевые адреса:

- внутренний интерфейс – IP адрес 192.168.57.2/24;
- внешний интерфейс – IP адрес 192.168.1.57/24.

Также для тестирования были созданы еще 2 виртуальные машины (далее абоненты), которые находились во внутренней сети каждого эмулятора и использовали его как шлюз по умолчанию.

Абонент 1 имеет следующую конфигурацию сетевого интерфейса:

- IP адрес – 192.168.56.3/24;
- шлюз по умолчанию – 192.168.56.2.

Абонент 2 имеет следующую конфигурацию сетевого интерфейса:

- IP адрес – 192.168.57.3/24;
- шлюз по умолчанию – 192.168.57.2.

Тестирование проводилось в сети с пропускной способностью 100 Мбит/сек, у эмуляторов был одинаковый криптографический ключ, и для визуализации передаваемых по сети пакетов использован сетевой анализатор «Wireshark».

Эмулятор 1 при запуске инициализировался параметрами файла conf1.xml, содержимое которого представлено ниже.

<CONFIGURATION>

```

<INTERFACES>
    <EXTERNAL>eth0</EXTERNAL>
    <INTERNAL>eth1</INTERNAL>
</INTERFACES>
<KEYFILE>/home/key.bin</KEYFILE>
<ROUTETABLE>
    <ROUTE>
        <NETADDR>192.168.57.0</NETADDR>
        <NETMASK>255.255.255.0</NETMASK>
        <GATEWAY>192.168.1.57</GATEWAY>
    </ROUTE>
</ROUTETABLE>
</CONFIGURATION>

```

Эмулятор 2 при запуске также инициализировался параметрами файла conf2.xml, содержимое которого представлено ниже.

```

<CONFIGURATION>
    <INTERFACES>
        <EXTERNAL>eth0</EXTERNAL>
        <INTERNAL>eth1</INTERNAL>
    </INTERFACES>
    <KEYFILE>/home/key.bin</KEYFILE>
    <ROUTETABLE>
        <ROUTE>
            <NETADDR>192.168.56.0</NETADDR>
            <NETMASK>255.255.255.0</NETMASK>
            <GATEWAY>192.168.1.56</GATEWAY>
        </ROUTE>
    </ROUTETABLE>
</CONFIGURATION>

```

При инициализации каждый эмулятор выдал уведомления, представленные на рисунках с 15 по 17.

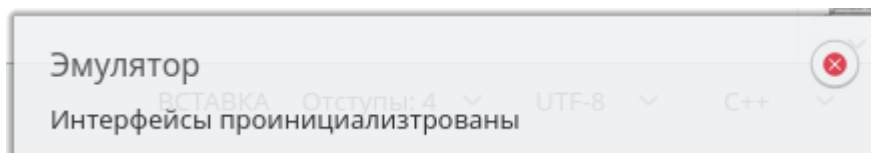


Рисунок 15 – Уведомление о успешной инициализации интерфейсов

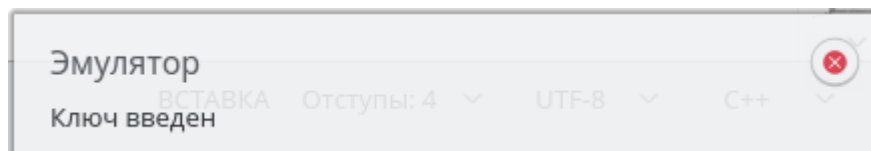


Рисунок 16 – Уведомление о успешной загрузке ключа

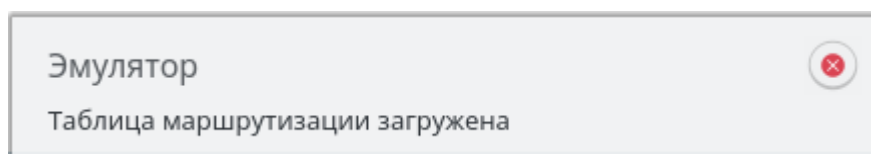


Рисунок 17 – Уведомление о успешной загрузке таблицы маршрутизации

В первую очередь была проверена обработка протокола ICMP, данная проверка проводилась на эмуляторе 1. Для этого на абонентом 1 посылались ICMP запросы на внутренний и внешний интерфейсы эмулятора 1. Результаты данного исследования представлены на рисунках 18 и 19.

No.	Time	Source	Destination	Protocol	Length	Info
→ 18	31.340197864	192.168.56.3	192.168.56.2	ICMP	98	Echo (ping) request id=0x2d19, seq=1/256, ttl=64 (reply in 19)
← 19	31.340407042	192.168.56.2	192.168.56.3	ICMP	98	Echo (ping) reply id=0x2d19, seq=1/256, ttl=63 (request in 18)
→ 20	32.342559787	192.168.56.3	192.168.56.2	ICMP	98	Echo (ping) request id=0x2d19, seq=2/512, ttl=64 (reply in 21)
← 21	32.342747903	192.168.56.2	192.168.56.3	ICMP	98	Echo (ping) reply id=0x2d19, seq=2/512, ttl=63 (request in 20)
→ 22	33.344821064	192.168.56.3	192.168.56.2	ICMP	98	Echo (ping) request id=0x2d19, seq=3/768, ttl=64 (reply in 23)
← 23	33.345114340	192.168.56.2	192.168.56.3	ICMP	98	Echo (ping) reply id=0x2d19, seq=3/768, ttl=63 (request in 22)
→ 24	34.346934993	192.168.56.3	192.168.56.2	ICMP	98	Echo (ping) request id=0x2d19, seq=4/1024, ttl=64 (reply in 25)
← 25	34.347226994	192.168.56.2	192.168.56.3	ICMP	98	Echo (ping) reply id=0x2d19, seq=4/1024, ttl=63 (request in 24)

▶ Frame 19: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0 ▶ Ethernet II, Src: PcsSyste_82:9c:5a (08:00:27:82:9c:5a), Dst: PcsSyste_fb:df:2f (08:00:27:fb:df:2f) ▶ Internet Protocol Version 4, Src: 192.168.56.2, Dst: 192.168.56.3 ▶ Internet Control Message Protocol Type: 0 (Echo (ping) reply) Code: 0 Checksum: 0x4290 [correct] [Checksum Status: Good] Identifier (BE): 11545 (0x2d19) Identifier (LE): 6445 (0x192d) Sequence number (BE): 1 (0x0001) Sequence number (LE): 256 (0x0100) [Request frame: 18]	<pre> 0000 08 00 27 fb df 2f 08 00 27 82 9c 5a 08 00 45 00 ..'...Z..E. 0010 00 54 00 01 00 00 3f 01 8a 52 c0 a8 38 02 c0 a8 .T....?.R..8... 0020 38 03 00 00 42 90 2d 19 00 01 d2 30 41 5a 00 00 8...B.-. ...0AZ.. 0030 00 00 b2 f7 0b 00 00 00 00 00 10 11 12 13 14 15 0040 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 !""#\$% 0050 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 &'()*+,-./012345 0060 36 37 67 </pre>
--	---

Рисунок 18 – Ping от абонента 1 на внутренний интерфейс эмулятора 1

Из результатов, представленных на рисунке 18 следует, что на каждый посланный абонентом 1 запрос он получал корректный ответ, значит эмулятор корректно принимает запрос, обрабатывает его и высылает ответ.

No.	Time	Source	Destination	Protocol	Length	Info
5	2.584619875	192.168.56.3	192.168.1.56	ICMP	98	Echo (ping) request id=0x2d56, seq=1/256, ttl=64 (no response found!)
6	3.592640357	192.168.56.3	192.168.1.56	ICMP	98	Echo (ping) request id=0x2d56, seq=2/512, ttl=64 (no response found!)
7	4.601146316	192.168.56.3	192.168.1.56	ICMP	98	Echo (ping) request id=0x2d56, seq=3/768, ttl=64 (no response found!)
10	5.608467354	192.168.56.3	192.168.1.56	ICMP	98	Echo (ping) request id=0x2d56, seq=4/1024, ttl=64 (no response found!)

▶ Frame 10: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0 ▶ Ethernet II, Src: PcsSyste_fb:df:2f (08:00:27:fb:df:2f), Dst: PcsSyste_82:9c:5a (08:00:27:82:9c:5a) ▶ Internet Protocol Version 4, Src: 192.168.56.3, Dst: 192.168.1.56 ▶ Internet Control Message Protocol Type: 8 (Echo (ping) request) Code: 0 Checksum: 0x0d40 [correct] [Checksum Status: Good] Identifier (BE): 11606 (0x2d56) Identifier (LE): 22061 (0x562d) Sequence number (BE): 4 (0x0004) Sequence number (LE): 1024 (0x0400) ▶ [No response seen]						
---	--	--	--	--	--	--

0000	08 00 27 82 9c 5a 08 00 27 fb df 2f 08 00 45 00	..'.Z.. '.../..E.
0010	00 54 9b 77 40 00 00 01 e4 a5 c0 a8 38 03 c0 a8	.T.w@.@.8...
0020	01 38 08 00 0d 40 2d 56 00 04 98 33 41 5a 00 00	.8...@-V ...3AZ..
0030	00 00 1a 05 0b 00 00 00 00 00 10 11 12 13 14 15	
0040	16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25	 !"#%\$
0050	26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35	&'()*+,-./012345
0060	36 37	67

Рисунок 19 – Ping от абонента 1 на внешний интерфейс эмулятора 1

Из результатов, представленных на рисунке 19 следует, что на все посланные абонентом 1 запросы он не получил ответа, это объясняется тем, что внешний интерфейс эмулятора находится в другой подсети и эмулятор не допускает обращения из внутренней сети напрямую к внешнему интерфейсу.

Для проверки обработки ICMP запросов на внешнем интерфейсе был послан ICMP запрос от узла сети с IP адресом 192.168.1.21 на внешний интерфейс эмулятора 1, результат отсылки данного запроса представлен на рисунке 20.

No.	Time	Source	Destination	Protocol	Length	Info
→	3 3.260378255	192.168.1.21	192.168.1.56	ICMP	98	Echo (ping) request id=0x5255, seq=1/256, ttl=65 (reply in 4)
←	4 3.260718923	192.168.1.56	192.168.1.21	ICMP	98	Echo (ping) reply id=0x5255, seq=1/256, ttl=64 (request in 3)
	5 4.260367998	192.168.1.21	192.168.1.56	ICMP	98	Echo (ping) request id=0x5255, seq=2/512, ttl=65 (reply in 6)
	6 4.260607318	192.168.1.56	192.168.1.21	ICMP	98	Echo (ping) reply id=0x5255, seq=2/512, ttl=64 (request in 5)
	9 5.290854588	192.168.1.21	192.168.1.56	ICMP	98	Echo (ping) request id=0x5255, seq=3/768, ttl=65 (reply in 10)
	10 5.291161132	192.168.1.56	192.168.1.21	ICMP	98	Echo (ping) reply id=0x5255, seq=3/768, ttl=64 (request in 9)
	11 6.314348252	192.168.1.21	192.168.1.56	ICMP	98	Echo (ping) request id=0x5255, seq=4/1024, ttl=65 (reply in 12)
	12 6.314476968	192.168.1.56	192.168.1.21	ICMP	98	Echo (ping) reply id=0x5255, seq=4/1024, ttl=64 (request in 11)

▶ Frame 4: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0 ▶ Ethernet II, Src: PcsSyste_c1:e0:fa (08:00:27:c1:e0:fa), Dst: IntelCor_93:b6:d0 (78:92:9c:93:b6:d0) ▶ Internet Protocol Version 4, Src: 192.168.1.56, Dst: 192.168.1.21 ▶ Internet Control Message Protocol Type: 0 (Echo (ping) reply) Code: 0 Checksum: 0x863e [correct] [Checksum Status: Good] Identifier (BE): 21077 (0x5255) Identifier (LE): 21842 (0x5552) Sequence number (BE): 1 (0x0001) Sequence number (LE): 256 (0x0100) [Request frame: 3]						
---	--	--	--	--	--	--

0000	78 92 9c 93 b6 d0 08 00 27 c1 e0 fa 08 00 45 00	x..... '.....E.
0010	00 54 00 05 00 00 40 01 f7 06 c0 a8 01 38 c0 a8	.T....@.8...
0020	01 15 00 00 86 3e 52 55 00 01 b5 32 41 5a 00 00	>RU ...2AZ..
0030	00 00 6f 0b 03 00 00 00 00 00 10 11 12 13 14 15	
0040	16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25	 !"#%\$
0050	26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35	&'()*+,-./012345
0060	36 37	67

Рисунок 20 – Ping от 192.168.1.21 на 192.168.1.56

Из результатов, представленных на рисунке 20 следует что на внешнем интерфейсе эмулятор корректно производит обработку ISMP пакетов.

Затем был протестирован обмен ключами, результаты тестирования представлены на рисунках 21 и 22.

No.	Time	Source	Destination	Protocol	Length	Info
499	44.445828081	192.168.1.56	192.168.1.57	ESP	114	ESP (SPI=0x0103484d)
500	44.454294819	192.168.1.57	192.168.1.56	ESP	114	ESP (SPI=0x0103484d)
501	44.455237242	192.168.1.56	192.168.1.57	ESP	146	ESP (SPI=0x0103484d)
502	44.461919413	192.168.1.57	192.168.1.56	ESP	146	ESP (SPI=0x0103484d)

▶ Frame 499: 114 bytes on wire (912 bits), 114 bytes captured (912 bits) on interface 0 ▶ Ethernet II, Src: PcsSyste_c1:e0:fa (08:00:27:c1:e0:fa), Dst: Azurewav_57:4a:b5 (6c:71:d9:57:4a:b5) ▶ Internet Protocol Version 4, Src: 192.168.1.56, Dst: 192.168.1.57 ▲ User Datagram Protocol, Src Port: 4500, Dst Port: 4500 Source Port: 4500 Destination Port: 4500 Length: 72 [Checksum: [missing]] [Checksum Status: Not present] [Stream index: 20] UDP Encapsulation of IPsec Packets ▶ Encapsulating Security Payload	0000 6c 71 d9 57 4a b5 08 00 27 c1 e0 fa 08 00 45 00 lq.WJ... '.....E. 0010 00 64 00 01 00 00 40 11 ca 98 c0 a8 01 38 c0 a8 .d....@.8.. 0020 01 39 11 94 11 94 00 48 00 00 01 03 48 4d 9f 21 .9.....HHM! 0030 87 0c 00 00 00 00 00 00 00 00 10 20 12 08 00 00 0040 00 00 18 19 1a 1b 1c 1d 1e 1f bb aa ff ee dd cc 0050 bb aa 28 29 2a 2b d4 73 1a 6c 68 ee 60 46 34 35 ..()*+.s .lh.`F45 0060 36 37 00 00 0c 00 01 0a 6c 69 6e 75 78 2d 67 72 67..... linux-gr 0070 79 37 y7
---	--

Рисунок 21 – Обмен ключами эмулятора 1 с эмулятором 2

No.	Time	Source	Destination	Protocol	Length	Info
128	4.776506730	192.168.1.57	192.168.1.56	ESP	114	ESP (SPI=0xd3cc3818)
129	4.777134546	192.168.1.56	192.168.1.57	ESP	114	ESP (SPI=0xd3cc3818)
130	4.785224788	192.168.1.57	192.168.1.56	ESP	146	ESP (SPI=0xd3cc3818)
131	4.786037545	192.168.1.56	192.168.1.57	ESP	146	ESP (SPI=0xd3cc3818)

▶ Frame 128: 114 bytes on wire (912 bits), 114 bytes captured (912 bits) on interface 0 ▶ Ethernet II, Src: Azurewav_57:4a:b5 (6c:71:d9:57:4a:b5), Dst: PcsSyste_c1:e0:fa (08:00:27:c1:e0:fa) ▶ Internet Protocol Version 4, Src: 192.168.1.57, Dst: 192.168.1.56 ▲ User Datagram Protocol, Src Port: 4500, Dst Port: 4500 Source Port: 4500 Destination Port: 4500 Length: 72 [Checksum: [missing]] [Checksum Status: Not present] [Stream index: 4] UDP Encapsulation of IPsec Packets ▶ Encapsulating Security Payload	0000 08 00 27 c1 e0 fa 6c 71 d9 57 4a b5 08 00 45 00 ..'...lq .WJ...E. 0010 00 64 00 01 00 00 40 11 ae 8b c0 a8 01 39 c0 a8 .d....@.9.. 0020 01 38 11 94 11 94 00 48 00 00 d3 cc 38 18 33 b6 .8.....H8.3. 0030 cf 23 00 00 00 00 00 00 00 00 10 20 12 08 00 00 .#..... 0040 00 00 18 19 1a 1b 1c 1d 1e 1f bb aa ff ee dd cc 0050 bb aa 28 29 2a 2b f6 5c 3d 27 e7 7d 21 2e 34 35 ..()*+.\ '='.}!.45 0060 36 37 30 30 0d 0a 4e 54 3a 75 75 69 64 3a 64 66 6700..NT :uuid:df 0070 35 63 5c
--	---

Рисунок 22 – Обмен ключами эмулятора 2 с эмулятором 1

Из результатов, представленных на рисунках 21 и 22 следует, что эмуляторы в соответствии с протоколом корректно обмениваются ключами, и

нет разницы в том, кто из них будет инициатор, а кто респондент так как эмулятор корректно обрабатывает полученные данные являясь либо инициатором, либо респондентом.

При успешном завершении обмена ключами эмулятор 1 выдал уведомление, представленное на рисунке 23.

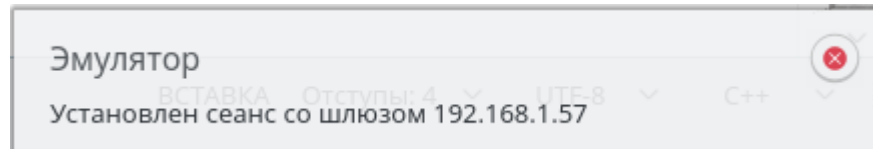


Рисунок 23 – Уведомление о установлении сеанса

После успешного обмена ключами было проведено тестирование передачи ICMP пакетов из одной подсети в другую, а именно от абонента 1 абоненту 2, результаты тестирования представлены на рисунках с 24 по 27.

No.	Time	Source	Destination	Protocol	Length	Info
358	58.223974485	192.168.56.3	192.168.57.3	ICMP	98	Echo (ping) request id=0x2eb1, seq=1/256, ttl=64 (no response found!)
364	59.232991677	192.168.56.3	192.168.57.3	ICMP	98	Echo (ping) request id=0x2eb1, seq=2/512, ttl=64 (no response found!)
370	60.240737436	192.168.56.3	192.168.57.3	ICMP	98	Echo (ping) request id=0x2eb1, seq=3/768, ttl=64 (no response found!)
378	61.248929131	192.168.56.3	192.168.57.3	ICMP	98	Echo (ping) request id=0x2eb1, seq=4/1024, ttl=64 (no response found!)
383	62.256632385	192.168.56.3	192.168.57.3	ICMP	98	Echo (ping) request id=0x2eb1, seq=5/1280, ttl=64 (no response found!)

Internet Control Message Protocol Type: 8 (Echo (ping) request) Code: 0 Checksum: 0x6ad3 [correct] [Checksum Status: Good] Identifier (BE): 11953 (0x2eb1) Identifier (LE): 45358 (0xb12e) Sequence number (BE): 1 (0x0001) Sequence number (LE): 256 (0x0100) [No response seen]						
---	--	--	--	--	--	--

0000	08 00 27 82 9c 5a 08 00	27 fb df 2f 08 00 45 00	..'.Z.. '...E.
0010	00 54 83 79 40 00 40 01	c4 d8 c0 a8 38 03 c0 a8	.T.y@.@.8..
0020	39 03 08 00 6a d3 2e b1	00 01 ed 43 41 5a 00 00	9...j... ..CAZ..
0030	00 00 67 09 0a 00 00 00	00 00 10 11 12 13 14 15	..g.....
0040	16 17 18 19 1a 1b 1c 1d	1e 1f 20 21 22 23 24 25	!"#\$%
0050	26 27 28 29 2a 2b 2c 2d	2e 2f 30 31 32 33 34 35	&'()*+,- ./012345
0060	36 37		67

Рисунок 24 – Пакеты отправляемые абонентом 1 абоненту 2

No.	Time	Source	Destination	Protocol	Length	Info
741	64.195863673	192.168.56.3	192.168.57.3	ICMP	98	Echo (ping) request id=0x2eb1, seq=1/256, ttl=64 (no response found!)
742	64.196306538	192.168.1.56	192.168.1.57	ESP	130	ESP (SPI=0x7f513a39)
753	65.205305518	192.168.56.3	192.168.57.3	ICMP	98	Echo (ping) request id=0x2eb1, seq=2/512, ttl=64 (no response found!)
754	65.205727881	192.168.1.56	192.168.1.57	ESP	130	ESP (SPI=0x7f513a39)
766	66.213547853	192.168.56.3	192.168.57.3	ICMP	98	Echo (ping) request id=0x2eb1, seq=3/768, ttl=64 (no response found!)
780	67.222747433	192.168.1.56	192.168.1.57	ESP	130	ESP (SPI=0x7f513a39)

Frame 754: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface 0 Ethernet II, Src: PcsSyste_c1:e0:fa (08:00:27:c1:e0:fa), Dst: Azurewav_57:4a:b5 (6c:71:d9:57:4a:b5) Internet Protocol Version 4, Src: 192.168.1.56, Dst: 192.168.1.57 Encapsulating Security Payload ESP SPI: 0x7f513a39 (2136029753) ESP Sequence: 4						
--	--	--	--	--	--	--

0000	6c 71 d9 57 4a b5 08 00	27 c1 e0 fa 08 00 45 00	lq.wJ.. '.....E.
0010	00 74 00 04 00 00 40 32	2f cc c0 a8 01 38 c0 a8	.t....@2 /...8..
0020	01 39 7f 51 3a 39 00 00	00 04 04 7a 94 b4 08 b4	.9.Q:9.Z....
0030	90 12 9b 44 39 fe b8 ac	40 f5 2a 9c 2d 74 4c 9e	...D9... @*.-tL.
0040	7b cf be c9 7f db 65 00	ae 74 3f 8a 71 2b 3b 22	{.....e. .t?.q+;"
0050	5e d3 53 fb 47 a4 f2 2b	b8 51 03 69 b1 3a ae 9b	^.S.G.+ .Q.i:...
0060	d4 ce d6 74 c3 e2 1d ee	8c 34 9b 1a 49 c9 68 1c	...t.... .4..I.h.
0070	02 0a c4 18 76 a0 b0 25	cf d4 05 8a 2c 8d bc db	v..%
0080	93 bd		..

Рисунок 25 – Пакеты приняты эмулятором1 и перенаправлены на эмулятор 2

Из результатов, представленных на рисунке 25 следует, что каждый принятый эмулятором 1 пакет зашифровывается и отправляется на эмулятор 2. Результат приведен на рисунке 26.

No.	Time	Source	Destination	Protocol	Length	Info
462	42.174895028	192.168.1.56	192.168.1.57	ESP	130	ESP (SPI=0x7f513a39)
463	42.175009905	192.168.56.3	192.168.57.3	ICMP	98	Echo (ping) request id=0x2eb1, seq=1/256, ttl=64 (no response found!)
476	43.182952151	192.168.1.56	192.168.1.57	ESP	130	ESP (SPI=0x7f513a39)
477	43.183071301	192.168.56.3	192.168.57.3	ICMP	98	Echo (ping) request id=0x2eb1, seq=2/512, ttl=64 (no response found!)
488	44.187260251	192.168.1.56	192.168.1.57	ESP	130	ESP (SPI=0x7f513a39)
489	44.187392114	192.168.56.3	192.168.57.3	ICMP	98	Echo (ping) request id=0x2eb1, seq=3/768, ttl=64 (no response found!)

▶ Frame 476: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface 0 ▶ Ethernet II, Src: IntelCor_93:b6:d0 (78:92:9c:93:b6:d0), Dst: PcsSyste_55:f4:4f (08:00:27:55:f4:4f) ▶ Internet Protocol Version 4, Src: 192.168.1.56, Dst: 192.168.1.57 ▶ Encapsulating Security Payload ESP SPI: 0x7f513a39 (2136029753) ESP Sequence: 4						
--	--	--	--	--	--	--

0000	08 00 27 55 f4 4f 78 92	9c 93 b6 d0 08 00 45 00	..U.Ox.E.
0010	00 74 00 04 00 00 40 32	2f cc c0 a8 01 38 c0 a8	.t....@2 /...8..
0020	01 39 7f 51 3a 39 00 00	00 04 04 7a 94 b4 08 b4	.9.Q:9... ..z....
0030	90 12 9b 44 39 fe b8 ac	40 f5 2a 9c 2d 74 4c 9e	...D9... @.*.-tl.
0040	7b cf be c9 7f db 65 00	ae 74 3f 8a 71 2b 3b 22	{.....t?.q;tj"
0050	5e d3 53 fb 47 a4 f2 2b	b8 51 03 69 b1 3a ae 9b	^S.G...+ .Q.i:...
0060	d4 ce d6 74 c3 e2 1d ee	8c 34 9b 1a 49 c9 68 1c	...t....4..I.h.
0070	02 0a c4 18 76 a0 b0 25	cf da 05 8a 2c 8d bc db	...v...%
0080	93 bd		..

Рисунок 26 – Пакеты приняты эмулятором 2 и перенаправлены абоненту 2

Из результатов, представленных на рисунке 26 следует, что полученные пакеты расшифровываются и отправляются адресату. Результат на рисунке 27.

No.	Time	Source	Destination	Protocol	Length	Info
294	52.191583950	192.168.56.3	192.168.57.3	ICMP	98	Echo (ping) request id=0x2eb1, seq=1/256, ttl=64 (no response found!)
303	53.199141864	192.168.56.3	192.168.57.3	ICMP	98	Echo (ping) request id=0x2eb1, seq=2/512, ttl=64 (no response found!)
310	54.202942780	192.168.56.3	192.168.57.3	ICMP	98	Echo (ping) request id=0x2eb1, seq=3/768, ttl=64 (no response found!)
317	55.209557726	192.168.56.3	192.168.57.3	ICMP	98	Echo (ping) request id=0x2eb1, seq=4/1024, ttl=64 (no response found!)
323	56.218133123	192.168.56.3	192.168.57.3	ICMP	98	Echo (ping) request id=0x2eb1, seq=5/1280, ttl=64 (no response found!)

▶ Frame 294: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0 ▶ Ethernet II, Src: PcsSyste_5c:d9:5f (08:00:27:5c:d9:5f), Dst: PcsSyste_a0:73:9a (08:00:27:a0:73:9a) ▶ Internet Protocol Version 4, Src: 192.168.56.3, Dst: 192.168.57.3 ▶ Internet Control Message Protocol Type: 8 (Echo (ping) request) Code: 0 Checksum: 0x6ad3 [correct] [Checksum Status: Good] Identifier (BE): 11953 (0x2eb1) Identifier (LE): 45358 (0xb12e) Sequence number (BE): 1 (0x0001) Sequence number (LE): 256 (0x0100) ▶ [No response seen] ▶ Data (56 bytes)						
--	--	--	--	--	--	--

0000	08 00 27 a0 73 9a 08 00	27 5c d9 5f 08 00 45 00	..'.s... '\...E.
0010	00 54 83 79 40 00 00 01	c4 d8 c0 a8 38 03 c0 a8	.T.y@. @.8...
0020	39 03 08 00 6a d3 2e b1	00 01 ed 43 41 5a 00 00	9...j... ..CAZ...
0030	00 00 67 09 0a 00 00 00	00 00 10 11 12 13 14 15	9...j... ..CAZ...
0040	16 17 18 19 1a 1b 1c 1d	1e 1f 20 21 22 23 24 25	..8..... !"#%&
0050	26 27 28 29 2a 2b 2c 2d	2e 2f 30 31 32 33 34 35	&()*+,- ./012345
0060	36 37		67

Рисунок 27 – Пакеты приняты абонентом 2

Из результатов, представленных на рисунках с 24 по 27 следует, что пакеты, отправленные адресатом 1 в другую подсеть были корректно маршрутизированы, прошли шифрование и расшифрование и были доставлены адресату без нарушения целостности.

Для подтверждения вывода, сделанного выше от абонента 2 абоненту 1 также были направлены ICMP пакеты, результаты тестирования представлены на рисунках с 28 по 31.

No.	Time	Source	Destination	Protocol	Length	Info
258	40.893881508	192.168.57.3	192.168.56.3	ICMP	98	Echo (ping) request id=0x2afa, seq=1/256, ttl=64 (no response found!)
263	41.893272893	192.168.57.3	192.168.56.3	ICMP	98	Echo (ping) request id=0x2afa, seq=2/512, ttl=64 (no response found!)
268	42.893264664	192.168.57.3	192.168.56.3	ICMP	98	Echo (ping) request id=0x2afa, seq=3/768, ttl=64 (no response found!)
273	43.893394361	192.168.57.3	192.168.56.3	ICMP	98	Echo (ping) request id=0x2afa, seq=4/1024, ttl=64 (no response found!)
278	44.893509958	192.168.57.3	192.168.56.3	ICMP	98	Echo (ping) request id=0x2afa, seq=5/1280, ttl=64 (no response found!)

▶ Frame 258: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
 ▶ Ethernet II, Src: PcsSyste_a0:73:9a (08:00:27:a0:73:9a), Dst: PcsSyste_5c:d9:5f (08:00:27:5c:d9:5f)
 ▶ Internet Protocol Version 4, Src: 192.168.57.3, Dst: 192.168.56.3
 ▲ Internet Control Message Protocol
 Type: 8 (Echo (ping) request)
 Code: 0
 Checksum: 0xb154 [correct]
 [Checksum Status: Good]
 Identifier (BE): 11002 (0x2afa)
 Identifier (LE): 64042 (0xfa2a)
 Sequence number (BE): 1 (0x0001)
 Sequence number (LE): 256 (0x0100)
 ▶ [No response seen]

0000	08 00 27 5c d9 5f 08 00 27 a0 73 9a 08 00 45 00	..'_...'.s...E.
0010	00 54 07 d2 00 00 40 01 40 00 c0 a8 39 03 c0 a8	.T..@. @...9...
0020	38 03 08 00 b1 54 2a fa 00 01 ea 40 41 5a 00 00	8....T*...@AZ...
0030	00 00 26 42 0b 00 00 00 00 00 10 11 12 13 14 15	..&B.....
0040	16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25	! "#\$%
0050	26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35	8'()*+,-./012345
0060	36 37	67

Рисунок 28 – Пакеты отправляемые абонентом 2 абоненту 1

No.	Time	Source	Destination	Protocol	Length	Info
367	31.898406997	192.168.57.3	192.168.56.3	ICMP	98	Echo (ping) request id=0x2afa, seq=1/256, ttl=64 (no response found!)
368	31.898575074	192.168.1.57	192.168.1.56	ESP	130	ESP (SPI=0x7f513b3a)
377	32.898313201	192.168.57.3	192.168.56.3	ICMP	98	Echo (ping) request id=0x2afa, seq=2/512, ttl=64 (no response found!)
387	33.898945147	192.168.1.57	192.168.1.56	ESP	130	ESP (SPI=0x7f513b3a)
388	33.898811943	192.168.57.3	192.168.56.3	ICMP	98	Echo (ping) request id=0x2afa, seq=3/768, ttl=64 (no response found!)
402	34.899580334	192.168.1.57	192.168.1.56	ESP	130	ESP (SPI=0x7f513b3a)

▶ Frame 368: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface 0
 ▶ Ethernet II, Src: PcsSyste_55:f4:4f (08:00:27:55:f4:4f), Dst: IntelCor_93:b6:d0 (78:92:9c:93:b6:d0)
 ▶ Internet Protocol Version 4, Src: 192.168.1.57, Dst: 192.168.1.56
 ▲ Encapsulating Security Payload
 ESP SPI: 0x7f513b3a (2136030010)
 ESP Sequence: 13

0000	78 92 9c 93 b6 d0 08 00 27 55 f4 4f 08 00 45 00	x.....'U.O..E.
0010	00 74 00 0d 00 00 40 32 4e 6e c0 a8 01 39 c0 a8	.t....@2 Nn...9..
0020	01 38 7f 51 3b 3a 00 00 00 0d 7c ab d8 fe ed 1c	.8.Q;:..
0030	a8 32 99 a6 96 9d e3 32 f2 49 5b 2d 39 28 c7 5f	.2.....2 .I[-9(.
0040	6f dd 04 94 c8 ae 86 e8 8a 8b 1c 72 f5 6d 9a f4	o.....r.m..
0050	ef 9d bd 3f b1 a3 2d 66 38 f3 c8 db e1 ae 41 f9	...?..-f 8.....A.
0060	0e f3 a3 05 7e 96 24 4d dd 9b 61 55 7a 1a f6 bb	~.\$M ..aUz...
0070	7e 1a 38 11 91 96 9a fb 92 b8 d1 b6 90 5c 9b 6f	~.8.....\..o
0080	7c ad	.

Рисунок 29 – Пакеты приняты эмулятором 2 и перенаправлены на эмулятор 1

No.	Time	Source	Destination	Protocol	Length	Info
160	11.925859774	192.168.1.57	192.168.1.56	ESP	130	ESP (SPI=0x7f513b3a)
161	11.926245506	192.168.57.3	192.168.56.3	ICMP	98	Echo (ping) request id=0x2afa, seq=1/256, ttl=64 (no response found!)
166	12.925947835	192.168.1.57	192.168.1.56	ESP	130	ESP (SPI=0x7f513b3a)
167	12.926358145	192.168.57.3	192.168.56.3	ICMP	98	Echo (ping) request id=0x2afa, seq=2/512, ttl=64 (no response found!)
178	13.926210688	192.168.1.57	192.168.1.56	ESP	130	ESP (SPI=0x7f513b3a)
179	13.926584215	192.168.57.3	192.168.56.3	ICMP	98	Echo (ping) request id=0x2afa, seq=3/768, ttl=64 (no response found!)

▶ Frame 160: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface 0
 ▶ Ethernet II, Src: Azurewav_57:4a:b5 (6c:71:d9:57:4a:b5), Dst: PcsSyste_c1:e0:fa (08:00:27:c1:e0:fa)
 ▶ Internet Protocol Version 4, Src: 192.168.1.57, Dst: 192.168.1.56
 ▲ Encapsulating Security Payload
 ESP SPI: 0x7f513b3a (2136030010)
 ESP Sequence: 13

0000	08 00 27 c1 e0 fa 6c 71 d9 57 4a b5 08 00 45 00	..'....lq .WJ...E.
0010	00 74 00 0d 00 00 40 32 4e 6e c0 a8 01 39 c0 a8	.t....@2 Nn...9..
0020	01 38 7f 51 3b 3a 00 00 00 0d 7c ab d8 fe ed 1c	.8.Q;:..
0030	a8 32 99 a6 96 9d e3 32 f2 49 5b 2d 39 28 c7 5f	.2.....2 .I[-9(.
0040	6f dd 04 94 c8 ae 86 e8 8a 8b 1c 72 f5 6d 9a f4	o.....r.m..
0050	ef 9d bd 3f b1 a3 2d 66 38 f3 c8 db e1 ae 41 f9	...?..-f 8.....A.
0060	0e f3 a3 05 7e 96 24 4d dd 9b 61 55 7a 1a f6 bb	~.\$M ..aUz...
0070	7e 1a 38 11 91 96 9a fb 92 b8 d1 b6 90 5c 9b 6f	~.8.....\..o
0080	7c ad	.

Рисунок 30 – Пакеты приняты эмулятором 1 и перенаправлены абоненту 1

No.	Time	Source	Destination	Protocol	Length	Info
458	74.897351690	192.168.57.3	192.168.56.3	ICMP	98	Echo (ping) request id=0x2afa, seq=1/256, ttl=64 (no response found!)
463	75.896963511	192.168.57.3	192.168.56.3	ICMP	98	Echo (ping) request id=0x2afa, seq=2/512, ttl=64 (no response found!)
470	76.896730577	192.168.57.3	192.168.56.3	ICMP	98	Echo (ping) request id=0x2afa, seq=3/768, ttl=64 (no response found!)
475	77.910750859	192.168.57.3	192.168.56.3	ICMP	98	Echo (ping) request id=0x2afa, seq=4/1024, ttl=64 (no response found!)
480	78.907086905	192.168.57.3	192.168.56.3	ICMP	98	Echo (ping) request id=0x2afa, seq=5/1280, ttl=64 (no response found!)
<p>▶ Frame 458: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0</p> <p>▶ Ethernet II, Src: PcsSyste_82:9c:5a (08:00:27:82:9c:5a), Dst: PcsSyste_fb:df:2f (08:00:27:fb:df:2f)</p> <p>▶ Internet Protocol Version 4, Src: 192.168.57.3, Dst: 192.168.56.3</p> <p>■ Internet Control Message Protocol</p> <p>Type: 8 (Echo (ping) request) Code: 0 Checksum: 0xb154 [correct] [Checksum Status: Good] Identifier (BE): 11002 (0x2afa) Identifier (LE): 64042 (0xfa2a) Sequence number (BE): 1 (0x0001) Sequence number (LE): 256 (0x0100)</p> <p>▶ [No response seen]</p> <p>Timestamp from icmp data: Dec 25, 2017 21:18:18.000000000 ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆ (◆◆◆◆◆)</p> <p>[Timestamp from icmp data (relative): -58.324026328 seconds]</p> <p>▶ Data (48 bytes)</p>						
<pre> 0000 08 00 27 fb df 2f 08 00 27 82 9c 5a 08 00 45 0e .T...@..Z..E. 0010 08 00 27 d2 40 00 40 01 40 80 c0 a8 39 03 c0 a8 .T...@..@...9.. 0020 38 03 08 00 b1 54 2a fa 00 01 ea 40 41 5a 00 0e 8....T"...@AZ... 0030 08 00 26 42 0b 00 00 00 00 10 11 12 13 14 15 ...&B..... 0040 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 "!#\$%&'()*+,-./01234567 0050 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35 8'()*+,-./01234567 0060 36 37 8'()*+,-./01234567 </pre>						

Рисунок 31 – Пакеты приняты абонентом 1

Из результатов, представленных на рисунках с 28 по 31 следует, что пакеты, отправленные адресатом 2 в другую подсеть были корректно смаршрутизированы, прошли шифрование и расшифрование и были доставлены адресату без нарушения целостности, и это полностью подтверждает вывод, сделанный ранее при пересылке пакетов от абонента 1 абоненту 2.

Эмулятор один раз в десять секунд сигнализирует о количестве обработанных пакетов, путем отсылки уведомлений, одно из которых представлено на рисунке 32.



Рисунок 32 – Уведомление о количестве обработанных пакетов

Исходя из полученных результатов тестирования следует вывод, что программный комплекс эмуляции устройства шифрования функционирует корректно выполняя при этом все требования технического задания.

8 Разработка руководства пользователя

8.1 Общие сведения

Программный комплекс эмуляции устройства шифрования предназначен для эмуляции основных функций устройства шифрования для IP-сети: обеспечения установления сеансов связи, шифрования и расшифрования трафика для работы в сети, построенной по протоколу IPSec.

Программный комплекс включает:

- модуль установления сеансов защищенной связи для создания сеансовых ключей, которые будут использоваться модулем шифрования/расшифрования информации;
- модуль ввода и хранения ключей шифрования для чтения главного ключа из файла, его хранение, а также хранение сеансовых ключей каждого сеанса защищенной связи;
- модуль индикации состояния для отображения параметров функционирования комплекса;
- модуль конфигурирования сеансов защищенной связи для чтения конфигурационного файла, чтение системных параметров интерфейсов внутренней и внешней сети, хранение текущей конфигурации.

В программном комплексе реализованы следующие возможности:

- работа с протоколами ARP, IP, IPSec, ICMP для обеспечения корректной работы по стандартным протоколам с другими сетевыми устройствами;
- введение и удаление ключей шифрования в программный комплекс в виде бинарного файла;
- введение, сохранение и удаление конфигурации сеансов защищенной связи для контролирования параметров программного комплекса;
- установление сеансов защищенной связи по протоколу IPSec;
- поддержка до 4096 сеансов защищенной связи одновременно;
- формирование сеансовых ключей шифрования для предотвращения возможности компрометации главного ключа;

- прием и передача зашифрованной информации через интерфейс внешней сети по протоколам IPSec для подключения данного интерфейса к коммутатору внешней сети;

- открытой информации через интерфейс внутренней сети по протоколу IP для подключения данного интерфейса к коммутатору внутренней сети;

- использование криптографического алгоритма, приведенного в национальном стандарте ГОСТ 28147-89 «Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования» [11] для шифрования и расшифрования пакетов так как данный алгоритм обеспечивает высокое быстродействие и надежную защиту информации.

8.2 Первоначальная настройка

Перед использованием программного комплекса эмуляции устройства шифрования (далее эмулятор) во внутренней сети необходимо установить значение MTU равное 1472 байта чтобы избежать потерь данных, так как внешняя сеть имеет MTU равное 1500 байт, а при обработке пакета, направляемого из внутренней сети во внешнюю он будет инкапсулирован сначала в ESP заголовок, а тот в свою очередь в заголовок IP и при этом размер исходного пакета будет увеличен на 28 байт.

Для функционирования программного комплекса необходима виртуальная машина под управлением ОС Linux имеющая два сетевых интерфейса, функционирующих в режиме моста, либо физическая машина под управлением той же операционной системы и также имеющая два сетевых адаптера.

Для поддержки до 4096 сеансов программному комплексу необходимо не менее 700 КБ оперативной памяти для хранения всех необходимых структур данных и конфигураций сеансов защищенной связи. Данное требование необходимо учитывать при конфигурировании виртуальной/физической машины перед установкой на нее данного эмулятора.

8.3 Пользовательская настройка

Для функционирования эмулятора необходимо создать конфигурационный файл формата XML, структура которого представлена ниже.

```
<CONFIGURATION>
  <INTERFACES>
    <EXTERNAL></EXTERNAL>
    <INTERNAL></INTERNAL>
  </INTERFACES>
  <KEYFILE></KEYFILE>
  <ROUTETABLE>
    <ROUTE>
      <NETADDR></NETADDR>
      <NETMASK></NETMASK>
      <GATEWAY></GATEWAY>
    </ROUTE>
    ...
    <ROUTE>
      <NETADDR></NETADDR>
      <NETMASK></NETMASK>
      <GATEWAY></GATEWAY>
    </ROUTE>
  </ROUTETABLE>
</CONFIGURATION>
```

В блоке **INTERFACES** содержатся поля в которые необходимо ввести имена сетевых интерфейсов:

- в поле **EXTERNAL** вводится имя внешнего интерфейса, через который предполагается взаимодействие с другими эмуляторами;
- в поле **INTERNAL** вводится имя внутреннего интерфейса, через который предполагается взаимодействие с компьютерами абонентов;
- в поле **KEYFILE** вводится имя файла, хранящего главный ключ шифрования в бинарном виде.

В блоке ROUTETABLE хранится таблица маршрутизации. Параметры маршрута хранятся в подблоках ROUTE, которые имеют следующие поля:

- NETADDR – для ввода адреса сети;
- NETMASK – для ввода маски сети;
- GATEWAY – для ввода адреса шлюза.

Адреса, описывающие маршрут являются IP адресами формата X.X.X.X, где X – число от 0 до 255.

Пример корректно заполненной конфигурации представлен ниже.

```
<CONFIGURATION>
  <INTERFACES>
    <EXTERNAL>eth0</EXTERNAL>
    <INTERNAL>eth1</INTERNAL>
  </INTERFACES>
  <KEYFILE>/home/key.bin</KEYFILE>
  <ROUTETABLE>
    <ROUTE>
      <NETADDR>192.168.57.0</NETADDR>
      <NETMASK>255.255.255.0</NETMASK>
      <GATEWAY>192.168.1.57</GATEWAY>
    </ROUTE>
    <ROUTE>
      <NETADDR>192.168.22.0</NETADDR>
      <NETMASK>255.255.255.0</NETMASK>
      <GATEWAY>192.168.1.22</GATEWAY>
    </ROUTE>
    <ROUTE>
      <NETADDR>192.168.5.0</NETADDR>
      <NETMASK>255.255.255.0</NETMASK>
      <GATEWAY>192.168.1.5</GATEWAY>
    </ROUTE>
  </ROUTETABLE>
</CONFIGURATION>
```

8.4 Работа с программой

После создания конфигурационного файла и файла с главным криптографическим ключом производится запуск эмулятора путем ввода имени программы эмулятора в консоль из-под учетной записи с правами администратора, так как без данных прав будет невозможно сетевое взаимодействие на канальном уровне.

Если эмулятор будет запущен с соблюдением описанных выше требований, и при инициализации не будет ошибок, тогда эмулятор сразу же начинает функционировать периодически уведомляя пользователя о своей работоспособности, а именно – посылая уведомления о успешных инициализации, чтении главного ключа и загрузке таблице маршрутизации, установлении сеанса защищенной связи со шлюзом, адрес которого будет указан в уведомлении, также каждые 10 секунд будет выводиться количество обработанных пакетов.

8.5 Сообщения об ошибках

В эмуляторе не предусмотрен вывод описания ошибок ни в консоль, ни в графическом виде, но ошибки могут возникнуть только на этапе инициализации эмулятора, а именно – запуск программы из-под учетной записи непривилегированного пользователя, отсутствующих или некорректно введенных параметров в файл конфигурации (сетевые интерфейсы, файл с ключом и т.д), при приеме некорректно сформированных пакетов, они будут отброшены.

Если эмулятору не удастся проинициализировать параметры, он совершит аварийное завершение, иначе уведомит пользователя об успешной инициализации.

9 Вредные психофизиологические факторы, влияющие на опорно-двигательный аппарат специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие

Вредными психофизиологическими факторами называются факторы, которые во время работы на производстве оказывают на рабочего негативное влияние, вследствие чего, в определенных условиях, приводят к заболеванию или снижению трудоспособности. Например, физические и эмоциональные перегрузки, умственное перенапряжение, монотонность труда [12].

Основная часть работ, выполняемых специалистом по защите информации, связана с компьютером. Он является основным инструментом при решении большинства задач. Соответственно работник проводит большую часть своего рабочего времени в сидячем положении, находясь за компьютером, что характеризует монотонность труда. При этом работник занят интеллектуальной деятельностью, что объединяет работы, связанные с приемом и обработкой информации, требующие преимущественно напряжения внимания, памяти, а также активации процессов мышления, эмоциональной сферы. Все это характеризуется малой физической активностью и большим психологическим напряжением.

Для продолжительного умственного труда в сидячем положении характерна гипокинезия, связанная со значительным снижением двигательной активности, что приводит к ухудшению реактивности организма и повышению эмоционального напряжения [13].

Также к вредным психофизиологическим факторам, влияющим на опорно-двигательный аппарат, относится неправильно обустроенное или неудобное рабочее место специалиста по защите информации. Из-за данного фактора подвержены опасности мышцы, суставы и кости. Со временем из организма уходит большинство полезных веществ, а новые не поступают из-за нарушений обменных процессов. Работа в сидячем положении оказывает негативное влияние на позвоночник и осанку. В первую очередь страдает шейный отдел, при

работе человек часто не замечает, что шея принимает неправильное положение и постепенно может развиваться остеохондроз. Кроме того, позвоночник еще является мышечной тканью и по мере длительного нахождения в сидячем положении мышцы начинают атрофироваться. Могут появиться боли в других группах мышц, запястьях, предплечьях, что также обусловлено длительным нахождением работника в одном и том же положении.

В соответствии с разделом 8 «Классификация опасных и вредных производственных факторов, обладающих свойствами психофизиологического воздействия на организм человека» [14] национального стандарта ГОСТ 12.0.003-2015 «Система стандартов по безопасности труда. Опасные и вредные производственные факторы. Классификация», длительная работа в сидячем положении относится к статическим физическим перегрузкам, связанным с рабочей позой работника.

Для того, чтобы избежать проблем, связанных с опорно-двигательной системой, специалисту по защите информации необходимо соблюдать правильный режим труда и отдыха, и, во время своего рабочего дня делать короткие перерывы и выполнять простую гимнастику, либо пешие прогулки. Простые упражнения для рук и кистей, например, протянуть руки вперед на ширину плеч, медленно сгибать и разгибать пальцы, затем из того же положения медленно сгибать и разгибать руки в запястьях, вращение кистями по кругу, также будут полезны и помогут предотвратить проблемы с опорно-двигательным аппаратом.

Комплекс упражнений для позвоночника способствует его расслаблению, улучшению периферического кровообращения, сохранению правильной осанки, оптимальному перераспределению мышечной нагрузки при работе и помогают преодолеть последствия гиподинамии.

Антропометрическая совместимость предполагает учет размеров тела человека, возможность обзора внешнего пространства, положения оператора в процессе работы [15]. Рабочее место специалиста по защите информации должно быть спроектировано таким образом, чтобы обеспечивалось наилучшее

размещение рабочих инструментов, не допускалось общего дискомфорта и снижало утомляемость. Рабочая поза сотрудника должна быть естественной. Составной частью рабочего места в положении сидя является рабочее кресло сотрудника. Кресло должно соответствовать антропометрическим данным человека. Целесообразно применять кресла с регулируемыми параметрами (высотой, углом наклона спинки), чтобы настроить его под конкретного человека [15]. Такие устройства как, клавиатура и мышь, должны находиться в соответствующих зонах досягаемости. Монитор также должен находиться в соответствующих зонах визуального поля человека и в пределах оптимальных углов обзора. На рисунке 1 изображено рекомендуемое правильное расположение работника на его рабочем месте перед компьютером.

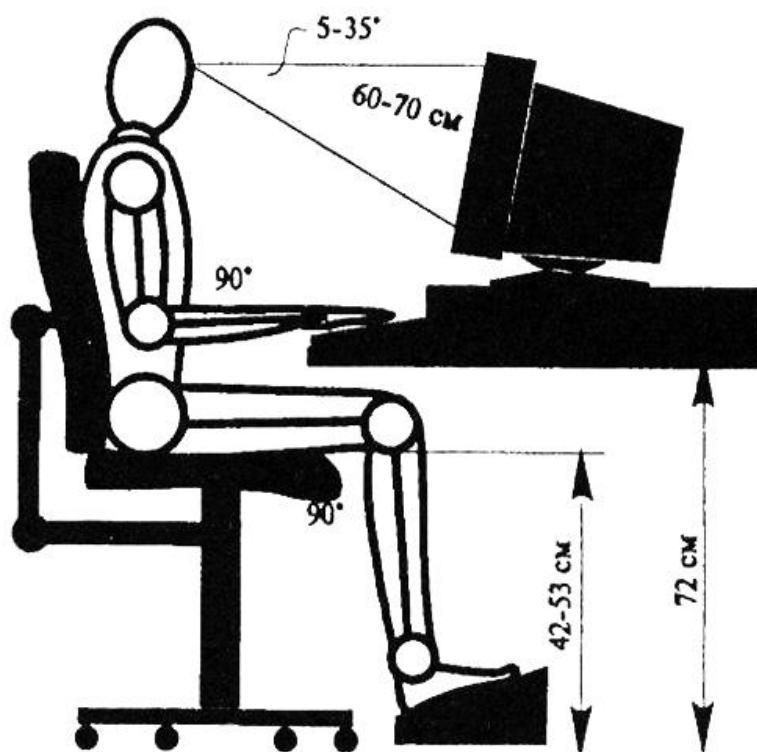


Рисунок 1 – Правильная поза работника на рабочем месте

Следуя простому разграничению трудового дня на рабочее время и перерывы для отдыха, а также правильной организации рабочего места, специалист по защите информации сможет снизить влияние вредных психофизиологических факторов на опорно-двигательный аппарат.

Заключение

В процессе выполнения дипломного проекта был разработан программный комплекс, предназначенный для эмуляции основных функций устройства шифрования для IP-сети: обеспечения установления сеансов связи, шифрования и расшифрования трафика для работы в сети, построенной по протоколу IPSec. Данный комплекс включает модули позволяющие: обеспечивать работу с протоколами ARP, IP, IPSec, введение и удаление ключей шифрования в программный комплекс в виде бинарного файла, введение, сохранение и удаление конфигурации сеансов защищенной связи для контролирования параметров программного комплекса, установление сеансов защищенной связи по протоколу IPSec, поддержка до 4096 сеансов защищенной связи одновременно, формирование сеансовых ключей шифрования, прием и передача зашифрованной информации через интерфейс внешней сети по протоколам IPSec, прием и передача открытой информации через интерфейс внутренней сети по протоколу IP, использование криптографического алгоритма, в соответствии с национальным стандартом ГОСТ 28147-89 «Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования», для шифрования и расшифрования пакетов.

В результате проведения тестирования данного программного комплекса было успешно проверено выполнение всех условий технического задания, это позволяет сделать вывод о том, что данный программный комплекс возможно применять для тестирования взаимодействия устройства с внешним оборудованием в условиях ограниченного количества изготовленных образцов, так как для изменений аппаратной части устройства затрачивается гораздо больше средств по сравнению с программной, которая позволяет сначала отладить все функции устройства программно, используя эмулятор, перед производством специализированного аппаратного обеспечения.

Список используемых источников

1. Методы и средства защиты информации [Электронный ресурс]: Екимов Д.А.. URL: <http://dfe.petrstu.ru/koi/posob/security/index.html>, – Загл. с экрана. (дата обращения: 20.12.2017)
2. RFC 791. Internet Protocol. Protocol Specification [Электронный ресурс]: J. Postel. URL: <http://www.networksorcery.com/enp/rfc/rfc791.txt>, – Загл. с экрана. (дата обращения: 23.12.2017)
3. IPSec – протокол защиты сетевого трафика на IP-уровне [Электронный ресурс]: С. Коротыгин. URL: <http://ixbt.com/comm/ipsecure.shtml#Тoc509471935>, – Загл. с экрана. (дата обращения: 20.12.2017)
4. IP-сети - Протоколы информационной безопасности [Электронный ресурс]: URL: http://ru.bmstu.wiki/IP-сети_-_Протоколы_информационной_безопасности (дата обращения: 20.12.2017)
5. RFC 4302. IP Authentication Header [Электронный ресурс]: S. Kent, URL: <http://www.networksorcery.com/enp/rfc/rfc4302.txt>, – Загл. с экрана. (дата обращения: 23.12.2017)
6. RFC 4303. IP Encapsulating Security Payload (ESP) [Электронный ресурс]: S. Kent, URL: <http://www.networksorcery.com/enp/rfc/rfc4303.txt>, – Загл. с экрана. (дата обращения: 23.12.2017)
7. Internet Key Exchange Protocol Version 2 (IKEv2) [Электронный ресурс]: URL: <http://tools.ietf.org/html/rfc7296> (дата обращения: 21.12.2017)
8. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile [Электронный ресурс]: URL: <http://tools.ietf.org/html/rfc5280> (дата обращения: 21.12.2017)
9. The TLS Protocol Version 1.0 [Электронный ресурс]: T. Dierks. URL: <http://tools.ietf.org/html/rfc2246> (дата обращения: 21.12.2017)
10. HMAC: Keyed-Hashing for Message Authentication [Электронный ресурс]: URL: <http://tools.ietf.org/html/rfc2104> (дата обращения: 21.12.2017)

11. ГОСТ 28147-89. Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования. – Введ. 01-07-1990 – М.: ИПК Издательство стандартов, 1996, – 26 с.

12. Безопасность жизнедеятельности [Текст]: Учеб. пособие для вузов /Под ред. проф. Л.А. Муравья. – 2-е изд., – М.: ЮНИТИ-ДАНА, 2010. – 431с.

13. Безопасность жизнедеятельности [Текст]: Учебник /В.Ю. Микрюков. – Изд. 2-е. – Ростов: Феникс, 2007. – 557с. : ил.

14. ГОСТ 12.0.003-2015. Система стандартов по безопасности труда. Опасные и вредные производственные факторы. Классификация. – Введ. 01-03-2017 – М.: Стандартиформ, 2016, – 9 с.

15. Охрана труда [Текст]: учебник /В.А. Девисилов. – 4-е изд. – М.: ФОРУМ, 2009. – 496с.: ил.

Приложение А
(обязательное)

Исходный код программного комплекса эмуляции устройства шифрования

Исходный код файла main.cpp

```
#include <algorithm>
#include <iostream>
#include <utility>
#include <fstream>
#include <vector>
#include <random>
#include <string>
#include <map>

#include <cstring>
#include <cstdlib>
#include <climits>
#include <ctime>

#include <sys/socket.h>
#include <linux/types.h>
#include <linux/if_ether.h>
#include <linux/if_packet.h>
#include <netinet/in.h>
#include <linux/ip.h>
#include <linux/if_arp.h>
#include <linux/icmp.h>
#include <linux/if.h>
#include <linux/udp.h>
#include <arpa/inet.h>

#include <unistd.h>
#include <sys/ioctl.h>

#include "Authentication.hpp"
#include "Cypher.hpp"
#include "GOST_28147_89.hpp"

#include <libnotify/notify.h>
#include "tinyxml2.h"
using namespace tinyxml2;

struct IConf {
    unsigned char MAC[6];
    unsigned int IP;
    unsigned int netmask;
    unsigned int netaddr;

    int socket;
    sockaddr_ll s_ll;

    unsigned int SPI;
    unsigned short IP_count;

    ~IConf(){
        close(socket);
    }
};

struct Key_bundle {

    struct Keys {

        struct Key_set {

            std::vector <unsigned long int> session_key;
            std::vector <unsigned long int> autentification_key;
```

```

std::vector <unsigned long int> authorization_number;
};

bool flag;          //Флаг существования ключей

Key_set local;      //Ключи текущего эмулятора
Key_set outside;    //Ключи внешнего эмулятора с которым обмениваются ключами текущий

int exchange;       //номер пакета обмена, -1 означает что обмен ключами еще не начинался, либо
завершен с ошибкой

unsigned long long Initiators_random_number;
unsigned long long Initiators_SPI;

unsigned long long Respondents_random_number;
unsigned long long Respondents_SPI;

Keys() {
    flag = false;
    exchange = -1;
    Initiators_random_number = 0;
}
};

std::vector <unsigned long int> main_key;
unsigned long int number;
unsigned long long int serial_number;

std::map < unsigned int, Keys > generated_keys;//unsigned int -> SPI;
};

struct Route {
    unsigned int netaddr;
    unsigned int netmask;
    unsigned int gateway;

    unsigned int SPI;

    unsigned int ESP_count;

    Route (unsigned int _netaddr, unsigned int _netmask, unsigned int _gateway)
    : netaddr(_netaddr), netmask(_netmask), gateway(_gateway) {
        SPI = _netaddr + _netmask + _gateway;
        ESP_count = 0;
    }
};

void Initialization (Key_bundle &Key_storage, std::vector < Route > &Route_table,
                    std::map < unsigned int, std::vector <unsigned char> > & ARP_cache,
                    IConf &Internal, IConf &External, int &sock_listener, std::vector <unsigned
char> &buffer );

inline bool Subnet_check (unsigned int ip, unsigned int subnetmask, unsigned int subnet);

inline unsigned short Internet_checksum (unsigned short * buffer, int size);

void ARP_packet_processing (std::map < unsigned int, std::vector <unsigned char> > &
ARP_cache,
                          IConf& Internal, IConf& External, std::vector< unsigned char >& buffer,
                          short unsigned int opcode, unsigned int ip = 0);

inline void Add_data_to_ARP_cache (std::map < unsigned int, std::vector <unsigned char> > &
ARP_cache, unsigned int ip, unsigned char * MAC);

```

```

void IP_packet_processing (Key_bundle &Key_storage,
    std::vector < std::pair < std::clock_t, std::vector < unsigned char > > >
    &packet_queue,
    std::map < unsigned int, std::vector < unsigned char > > &ARP_cache, std::vector
    < Route > &Route_table,
    IConf& Internal, IConf& External, std::vector< unsigned char > &buffer, int
    const recieved_bytes);

void ICMP_packet_processing (std::map < unsigned int, std::vector < unsigned char > > &
    ARP_cache,
    IConf& Conf, std::vector< unsigned char >& buffer, int const
    recieved_bytes);

void Key_exchange (Key_bundle &Key_storage,
    std::map < unsigned int, std::vector < unsigned char > > & ARP_cache,
    IConf& Conf, std::vector< unsigned char >& buffer, int const recieved_bytes,
    unsigned int SPI = 0, bool start = false, unsigned int respondentsIP = 0);

void ESP_packet_processing (Key_bundle &Key_storage, std::vector < Route > &Route_table,
    std::vector < std::pair < std::clock_t, std::vector < unsigned char > > >
    &packet_queue,
    std::map < unsigned int, std::vector < unsigned char > > & ARP_cache,
    IConf& Conf, std::vector< unsigned char >& buffer, int const recieved_bytes,
    bool make = false, unsigned int gateway_ip = 0);

int main() {

    IConf Internal;
    IConf External;
    int listener;

    Key_bundle Key_storage;
    std::map < unsigned int, std::vector < unsigned char > > ARP_cache;
    std::vector < std::pair < std::clock_t, std::vector < unsigned char > > > packet_queue;
    std::vector < Route > Route_table;
    std::vector < unsigned char > buffer (ETH_FRAME_LEN);

    ethhdr * ethernet_header = (ethhdr *)&buffer[0];
    arphdr * ARP_header = (arphdr *)&buffer[sizeof(ethhdr)];
    iphdr * ip_header = (iphdr *)&buffer[sizeof(ethhdr)];

    notify_init("Init");
    Initialization(Key_storage,Route_table, ARP_cache, Internal, External, listener,
    buffer);
    External.SPI = External.IP + Internal.netaddr + Internal.netmask;

    size_t timer = std::clock();

    for ( int rec = 0;;) {

        //TODO вывод обработанных пакетов!!!!!!!!!!!!!!!!!!!!
        rec = recvfrom(listener, (char *)&buffer[0], ETH_FRAME_LEN, 0, NULL, NULL);

        //Если пакет исходящий, в начало цикла
        if ( (rec < 0) ||
            memcmp(ethernet_header->h_source, Internal.MAC, ETH_ALEN) == 0 ||
            memcmp(ethernet_header->h_source, External.MAC, ETH_ALEN) == 0 ) {

            buffer.clear();
            continue;
        }

        if(((std::clock() - timer) / 1000) > 10) {

```

```

        std::string message("обработано ");
        message.append(std::to_string( (Internal.IP_count + External.IP_count) * 2 ));
        message.append(" пакета/ов");
        notify_notification_show(notify_notification_new ("Эмулятор",  message.c_str(),
0), 0);
        timer = std::clock();
    }

    if ( htons(ethernet_header->h_proto ) == ETH_P_ARP)
        ARP_packet_processing(ARP_cache, Internal, External, buffer, ARP_header->ar_op);

    else if ( ( memcmp(ethernet_header->h_dest, Internal.MAC, ETH_ALEN) == 0 ||
        memcmp(ethernet_header->h_dest, External.MAC, ETH_ALEN) == 0) &&
        htons(ethernet_header->h_proto) == ETH_P_IP )
        IP_packet_processing(Key_storage, packet_queue, ARP_cache, Route_table,
Internal, External, buffer, rec);

    buffer.clear();
    if ( !packet_queue.empty() ) {//Проверка очереди сообщений без MAC адреса получателя

        for ( size_t i = 0; i < packet_queue.size(); i++) {

            ip_header = (iphdr *)&packet_queue[i].second[sizeof(ethhdr)];//Заголовок
пакета из очереди

            if (((std::clock() - packet_queue[i].first) / 1000) > 1) {//Если запись в
очереди больше секунд и нет адреса

                packet_queue.erase(packet_queue.begin() + i);//Удаление
i--;
                continue;

            }

            if ( Subnet_check(ip_header->daddr, Internal.netmask, Internal.netaddr) &&
ARP_cache.find(ip_header->daddr) != ARP_cache.end() ){//MAC появился, обработка

                ethernet_header = (ethhdr *)&packet_queue[i].second[0];
                memcpy(ethernet_header->h_dest, &ARP_cache[ip_header->daddr][0],
ETH_ALEN);

                write(Internal.socket, (void *)&packet_queue[i].second[0],
packet_queue[i].second.size());

                packet_queue.erase(packet_queue.begin() + i);//Удаление
i--;
                continue;

            }

            for (Route &route: Route_table) {
                if (Subnet_check(ip_header->daddr, route.netmask, route.netaddr))
                {//Подсеть назначения найдена
                    if ( ARP_cache.find(route.gateway) != ARP_cache.end() &&
Key_storage.generated_keys[route.SPI].flag){//MAC появился, обработка

                        IP_packet_processing(Key_storage, packet_queue, ARP_cache,
Route_table, Internal, External, packet_queue[i].second, packet_queue[i].second.size());
                        packet_queue.erase(packet_queue.begin() + i);//Удаление
                        i--;
                        break;

                    }

                }

            }

        }

    }

}

```

```

    }
}

void Initialization ( Key_bundle &Key_storage, std::vector < Route > &Route_table,
                    std::map < unsigned int, std::vector <unsigned char> > & ARP_cache,
                    IConf &Internal, IConf &External, int &sock_listener, std::vector <unsigned
char> &buffer ) {

    NotifyNotification *n;
    tinyxml2::XMLDocument conf;
    conf.LoadFile("configuration.xml");

    tinyxml2::XMLElement *configuration, *interfaces, *keyfile, *routetable;

    configuration = conf.FirstChildElement("CONFIGURATION");
    if (!configuration) exit(EXIT_FAILURE);

    interfaces = configuration->FirstChildElement("INTERFACES");
    if (!interfaces) exit(EXIT_FAILURE);

    std::string ifaces[2];//инициализация имен интерфейсов
    ifaces[0].append(interfaces->FirstChildElement("EXTERNAL")->GetText(),
strlen(interfaces->FirstChildElement("EXTERNAL")->GetText()));
    ifaces[1].append(interfaces->FirstChildElement("INTERNAL")->GetText(),
strlen(interfaces->FirstChildElement("INTERNAL")->GetText()));

    sock_listener = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
    if(sock_listener < 0) {
        exit(EXIT_FAILURE);
    }

    for( int i = 0; i < 2; i++) {

        IConf * Conf = &External;
        if ( i != 0)Conf = &Internal;

        ifreq ifCfg;
        memset(&ifCfg, 0, sizeof (ifCfg));

        memcpy(ifCfg.ifr_name, ifaces[i].c_str(), ifaces[i].length());
        if ( ioctl (sock_listener, SIOCGIFINDEX, &ifCfg) < 0 ) {
            exit(EXIT_FAILURE);
        }

        if ( ioctl (sock_listener, SIOCGIFHWADDR, &ifCfg) < 0 ) {
            exit(EXIT_FAILURE);
        } else memcpy(Conf->MAC, ifCfg.ifr_hwaddr.sa_data, sizeof(Conf->MAC));

        if ( ioctl (sock_listener, SIOCGIFADDR, &ifCfg) < 0 ) {
            exit(EXIT_FAILURE);
        } else memcpy( (void *)&Conf->IP, (void *)&((struct sockaddr_in *)&ifCfg.ifr_addr)-
>sin_addr, sizeof(Conf->IP));

        if ( ioctl (sock_listener, SIOCGIFNETMASK, &ifCfg) < 0 ) {
            exit(EXIT_FAILURE);
        } else memcpy( (void *)&Conf->netmask, (void *)&((struct sockaddr_in
*)&ifCfg.ifr_netmask)->sin_addr, sizeof(Conf->netmask));

        Conf->netaddr = Conf->IP & Conf->netmask;

        Conf->socket = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
        if(Conf->socket < 0) {
            exit(EXIT_FAILURE);
        }
    }
}

```

```

        memset((void *)&Conf->s_ll, 0, sizeof(struct sockaddr_ll));
        Conf->s_ll.sll_family = PF_PACKET;
        Conf->s_ll.sll_protocol = htons(ETH_P_ALL);
        Conf->s_ll.sll_ifindex = ifCfg.ifr_ifindex;
        Conf->s_ll.sll_pkttype = PACKET_BROADCAST;

        if(bind (Conf->socket, (struct sockaddr *) &Conf->s_ll, sizeof (struct sockaddr_ll))
< 0 ) {
            exit(EXIT_FAILURE);
        }

        Conf->IP_count = Conf->SPI = 0;

    }
    notify_notification_show(notify_notification_new ("Эмулятор", "Интерфейсы
проинициализированы", 0), 0);

    //главный ключ(загрузить из файла)
    keyfile = configuration->FirstChildElement("KEYFILE");
    std::string key_filepath(keyfile->GetText());
    std::fstream key_file ( key_filepath.c_str() , std::ios_base::in | std::ios_base::binary
);

    key_file.seekg (0, key_file.end);
    unsigned int filesize = (unsigned int) key_file.tellg();
    key_file.seekg (0, key_file.beg);

    if (filesize < ( sizeof(unsigned long int) * 8)) {
        exit(EXIT_FAILURE);
    }

    //unsigned long int main_key[8] = {0x12AB60F5, 0xFB8843C1, 0x430AC88F, 0x678DE12B,
0xABE49671, 0x375FA123, 0x5AE02F59, 0x0B540CA1};
    Key_storage.main_key.resize(8);
    key_file.read( (char *) &Key_storage.main_key[0], (sizeof(unsigned long int) * 8));
    notify_notification_show(notify_notification_new ("Эмулятор", "Ключ введен", 0), 0);

    Key_storage.number = 1;
    Key_storage.serial_number = 1;

    routetable = configuration->FirstChildElement("ROUTETABLE");
    if (!routetable) exit(EXIT_FAILURE);

    for (const tinyxml2::XMLElement * route = routetable->FirstChildElement("ROUTE"); route;
route = route->NextSiblingElement("ROUTE")) {

        const tinyxml2::XMLElement *netaddr, *netmask, *gateway;

        netaddr = route->FirstChildElement("NETADDR");
        netmask = route->FirstChildElement("NETMASK");
        gateway = route->FirstChildElement("GATEWAY");

        if ( !(netaddr && netmask && gateway) ) continue;

        Route_table.push_back(Route(inet_addr(netaddr->GetText()), inet_addr(netmask-
>GetText()), inet_addr(gateway->GetText())));
    }
    notify_notification_show(notify_notification_new ("Эмулятор", "Таблица маршрутизации
загружена", 0), 0);
}

void ARP_packet_processing ( std::map < unsigned int, std::vector <unsigned char> > &
ARP_cache,

```

```

        IConf& Internal, IConf& External, std::vector< unsigned char > &buffer,
        short unsigned int opcode, unsigned int ip) {

    ethhdr * eth = (ethhdr *)&buffer[0];
    arphdr * arp = (arphdr *)&buffer[sizeof(ethhdr)];

    unsigned char * SHA, *THA;
    unsigned int * SPA, * TPA;

    SHA = &buffer [ sizeof(ethhdr) + sizeof(arphdr) ]; //sender hardware address
    SPA = (unsigned int *) &buffer [ sizeof(ethhdr) + sizeof(arphdr) + 6]; //sender IP
address
    THA = &buffer [ sizeof(ethhdr) + sizeof(arphdr) + 6 + 4]; //target hardware address
    TPA = (unsigned int *) &buffer [ sizeof(ethhdr) + sizeof(arphdr) + 6 + 4 + 6]; //target
IP address

    IConf * Conf = nullptr;

    if ( htons(opcode) == ARPOP_REQUEST ) { //Processing recieved ARP request

        if ( *TPA == Internal.IP ) Conf = &Internal;
        else if ( *TPA == External.IP ) Conf = &External;

        if ( Conf == nullptr ) {
            buffer.clear();
            return;
        }

        memcpy(eth->h_dest, eth->h_source, ETH_ALEN);
        memcpy(eth->h_source, Conf->MAC, ETH_ALEN);

        arp->ar_op = htons(ARPOP_REPLY);

        memcpy(THA, SHA, ETH_ALEN);
        memcpy(TPA, SPA, sizeof(*TPA));

        memcpy(SHA, &Conf->MAC[0], ETH_ALEN);
        memcpy(SPA, (void *)&Conf->IP, sizeof(*SPA));

        write(Conf->socket, (void *)&buffer[0], 60);

    }else if ( ((opcode == ARPOP_REQUEST) || (opcode == ARPOP_REPLY)) && ip ) { //Make and
send ARP request and gratuitous arp

        if ( Subnet_check(ip, Internal.netmask, Internal.netaddr) ) Conf = &Internal;
        else if ( Subnet_check(ip, External.netmask, External.netaddr) ) Conf = &External;

        if ( Conf == nullptr ) {
            buffer.clear();
            return;
        }

        unsigned char broadcast[6] = { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };

        memcpy(eth->h_dest, broadcast, ETH_ALEN);
        memcpy(eth->h_source, Conf->MAC, ETH_ALEN);
        eth->h_proto = htons(ETH_P_ARP);

        arp->ar_hrd = htons(ETH_P_802_3);
        arp->ar_pro = htons(ETH_P_IP);
        arp->ar_hln = ETH_ALEN;
        arp->ar_pln = 4;
        arp->ar_op = htons(opcode);

```

```

        memcpy(SHA, &Conf->MAC[0], ETH_ALEN);
        memcpy(SPA, (void *)&Conf->IP, sizeof(*SPA));

        memcpy(THA, broadcast, ETH_ALEN);
        memcpy(TPA, &ip, sizeof(*TPA));

        write(Conf->socket, (void *)&buffer[0], 60);

    } else if ( htons(opcode) == ARPOP_REPLY ) { //Processing recieved ARP reply

        if ( (*SPA == Internal.IP) || ( *SPA == External.IP) ) {
            buffer.clear();
            return;
        }

        Add_data_to_ARP_cache(ARP_cache, *SPA, SHA);
    }

    buffer.clear();
}

inline void Add_data_to_ARP_cache ( std::map < unsigned int, std::vector <unsigned char> >
& ARP_cache, unsigned int ip, unsigned char * MAC ) {

    ARP_cache[ip].resize(ETH_ALEN);
    std::copy(MAC, MAC + ETH_ALEN, ARP_cache[ip].begin());
}

inline bool Subnet_check(unsigned int ip, unsigned int subnetmask, unsigned int subnet){

    return ((ip & subnetmask) == subnet) ? true : false;
}

inline unsigned short Internet_checksum( unsigned short * buffer, int size) {

    unsigned long cksum = 0;

    while (size > 1) {
        cksum += *buffer++;
        size -= sizeof(unsigned short);
    }

    if (size) cksum += *(unsigned char*)buffer;

    cksum = (cksum >> 16) + (cksum & 0xffff);
    cksum += (cksum >>16);

    return (unsigned short)(~cksum);
}

void IP_packet_processing ( Key_bundle &Key_storage, std::vector < std::pair < std::clock_t,
std::vector < unsigned char > > > &packet_queue,
                        std::map < unsigned int, std::vector <unsigned char> > & ARP_cache,
std::vector < Route > &Route_table,
                        IConf& Internal, IConf& External, std::vector< unsigned char >& buffer, int
const recieved_bytes) {

    ethhdr * eth = (ethhdr *)&buffer[0];
    iphdr * ip = (iphdr *)&buffer[sizeof(ethhdr)];

    unsigned short ip_chksum = ip->check;

```

```

        ip->check = 0;
        if ( ip_chksum == Internet_checksum( (unsigned short *)ip, recieved_bytes -
        (sizeof(ethhdr)) ) ) {

            Add_data_to_ARP_cache(ARP_cache, ip->saddr, eth->h_source); //Обновление арп кэша
            адресами входящего пакета

            if (ip->daddr == Internal.IP || ip->daddr == External.IP) { //Пакет для эмулятора

                if (ip->protocol == IPPROTO_ESP && ip->protocol != IPPROTO_UDP ) {

                    ESP_packet_processing (Key_storage, Route_table, packet_queue,
                    ARP_cache, Internal, buffer, recieved_bytes);

                } else if (ip->protocol == IPPROTO_ICMP) { //ICMP

                    ip->ttl -= 1; // To all ip packets on gateway
                    ip->frag_off = 0;

                    if ( ip->daddr == Internal.IP ) {
                        ip->daddr = ip->saddr;
                        ip->saddr = Internal.IP;
                        ip->id = htons( ++Internal.IP_count );
                        ICMP_packet_processing(ARP_cache, Internal, buffer, recieved_bytes);
                    } else if ( ip->daddr == External.IP ) {

                        ip->daddr = ip->saddr;
                        ip->saddr = External.IP;
                        ip->id = htons( ++External.IP_count );
                        ICMP_packet_processing(ARP_cache, External, buffer, recieved_bytes);
                    }
                } else if (ip->protocol == IPPROTO_UDP && ip->daddr == External.IP) {
                    //Ключевой обмен

                    udphdr * udp = (udphdr *)&buffer[sizeof(ethhdr) + (ip->ihl * 4)];
                    if (udp->dest == htons(4500) && udp->source == htons(4500))
                        for (Route &route: Route_table)
                            if (ip->saddr, route.gateway)
                                Key_exchange (Key_storage, ARP_cache, External, buffer,
                                recieved_bytes, route.SPI);
                }
            } else if (Subnet_check(ip->saddr, Internal.netmask, Internal.netaddr)) { //Пакет
            идет из внутренней сети во внешнюю
                for (Route &route: Route_table) {
                    if (Subnet_check(ip->daddr, route.netmask, route.netaddr)) { //Подсеть
                    назначения найдена
                        if (Key_storage.generated_keys[route.SPI].flag) { //Со шлюзом данной
                        подсети установлен сеанс

                            ESP_packet_processing (Key_storage, Route_table, packet_queue, ARP_cache,
                            External, buffer, recieved_bytes, true, route.gateway);

                        } else {

                            //Постановка пакета в очередь
                            packet_queue.push_back ( std::make_pair( std::clock(),
                            std::vector<unsigned char>( buffer.begin(), buffer.begin() + recieved_bytes ) ) );

                            if (ARP_cache.find(route.gateway) == ARP_cache.end()) { //Проверка
                            мак адреса шлюза

                                ARP_packet_processing(ARP_cache, External, External, buffer,
                                ARPOP_REQUEST, route.gateway);
                            }
                        }
                    }
                }
            }
        }
    }

```

```

        return;
    }

    if (Key_storage.generated_keys[route.SPI].exchange < 0) { //если
обмен ключами не начат, начало обмена

        Key_storage.generated_keys[route.SPI].exchange = 0; //обмен начат
        Key_exchange (Key_storage, ARP_cache, External, buffer, 0,
route.SPI, true, route.gateway);
    }
    }
    break;
}
}
}

buffer.clear();
}

void ICMP_packet_processing (std::map < unsigned int, std::vector <unsigned char> > &
ARP_cache,
        IConf& Conf, std::vector< unsigned char >& buffer, int const recieved_bytes)
{
    ethhdr * eth = (ethhdr *)&buffer[0];
    iphdr * ip = (iphdr *)&buffer[sizeof(ethhdr)];
    icmp_hdr * icmp = (icmp_hdr *)&buffer[sizeof(ethhdr) + (ip->ihl * 4)];

    unsigned short icmp_chksum = icmp->checksum;
    icmp->checksum = 0;
    if(icmp_chksum == Internet_checksum( (unsigned short *)icmp, recieved_bytes -
(sizeof(ethhdr)+(ip->ihl * 4)) ) ) {
        if(icmp->type == ICMP_ECHO ) {

            icmp->type = ICMP_ECHOREPLY;

            //calculate checksum's
            icmp->checksum = Internet_checksum( (unsigned short *)icmp, recieved_bytes -
(sizeof(ethhdr) + (ip->ihl * 4)) );
            ip->check = Internet_checksum( (unsigned short *)ip, recieved_bytes -
(sizeof(ethhdr)) );

            //L2
            memcpy(eth->h_dest, eth->h_source, ETH_ALEN);
            memcpy(eth->h_source, Conf.MAC, ETH_ALEN);

            write(Conf.socket, (void *)&buffer[0], recieved_bytes);
        }
    }

    buffer.clear();
}

void Key_exchange (Key_bundle &Key_storage,
        std::map < unsigned int, std::vector <unsigned char> > & ARP_cache,
        IConf& Conf, std::vector< unsigned char >& buffer, int const recieved_bytes,
        unsigned int SPI, bool start, unsigned int respondentsIP) {

    const unsigned long long int Vendor = 0xAABBCCDDEEFFAABB;
    const unsigned char Protocol_version = 32;

    ethhdr * eth = (ethhdr *)&buffer[0];
    iphdr * ip = (iphdr *)&buffer[sizeof(ethhdr)];
    udphdr * udp = (udphdr *)&buffer[sizeof(ethhdr) + sizeof(iphdr)];

```

```

KEPacket * kepacket = (KEPacket*)&buffer[sizeof(ethhdr) + sizeof(iphdr) + sizeof(udp)];

std::random_device rd;
std::uniform_int_distribution<long long int> uid(0, LLONG_MAX);

if (start && respondentsIP) {

    memset(ip, 0, sizeof(iphdr));
    memset(udp, 0, sizeof(udphdr));

    //инициализация полей для пакета номер 1
    kepacket->RFlag = 8;
    kepacket->RMessage_number = 0;
    kepacket->RVersion = Protocol_version;
    kepacket->RSPIInit = uid(rd);
    kepacket->RSPIResp = 0;
    kepacket->GRandom_number_init = uid(rd);
    kepacket->YVendor = Vendor;

    //Копирование номера ключа и его серии
    memcpy(kepacket->BNumber_key_and_series, &Key_storage.number,
sizeof(Key_storage.number));
    memcpy(kepacket->BNumber_key_and_series + sizeof(Key_storage.number),
&Key_storage.serial_number, sizeof(Key_storage.serial_number));

    //Сохранение временных значений
    Key_storage.generated_keys[SPI].Initiators_SPI = kepacket->RSPIInit;
    Key_storage.generated_keys[SPI].Initiators_random_number = kepacket-
>GRandom_number_init;

    //заполнение ethernet заголовка
    memcpy(eth->h_dest, &ARP_cache[respondentsIP][0], ETH_ALEN);
    memcpy(eth->h_source, Conf.MAC, ETH_ALEN);
    eth->h_proto = htons(ETH_P_IP);
    //заполнение udp заголовка
    udp->dest = htons(4500);
    udp->source = htons(4500);
    udp->len = htons(sizeof(KEPacket));
    //Заполнение ip заголовка и отправка
    ip->daddr = respondentsIP;
    ip->saddr = Conf.IP;
    ip->id = htons(++Conf.IP_count);
    ip->ttl = 64;
    ip->protocol = IPPROTO_UDP;
    ip->version = 4;
    ip->ihl = 5;
    ip->tot_len = htons(sizeof(iphdr) + sizeof(udphdr) + sizeof(KEPacket));
    ip->check = Internet_checksum( (unsigned short *)ip, htons(ip->tot_len));

    write(Conf.socket, (void *)&buffer[0], sizeof(ethhdr) + htons(ip->tot_len));
    return;
}

if ((kepacket->RVersion == Protocol_version)&&(kepacket->YVendor == Vendor)) {

    if (kepacket->RMessage_number == 0) {
        if (Key_storage.generated_keys[SPI].exchange == -1) {
            unsigned long int temp_key_number = 0;
            unsigned long long int temp_key_serial_number = 0;
            //Копирование номера ключа и его серии
            memcpy(&temp_key_number, kepacket->BNumber_key_and_series,
sizeof(temp_key_number));
            memcpy(&temp_key_serial_number, kepacket->BNumber_key_and_series +
sizeof(temp_key_number), sizeof(temp_key_serial_number));

```

```

        if(temp_key_number != Key_storage.number && temp_key_serial_number !=
Key_storage.serial_number) return;

        Key_storage.generated_keys[SPI].exchange = 0; //Был принят нулевой пакет

        //Создание spi и случайного числа инициатора
        Key_storage.generated_keys[SPI].Initiators_SPI = kepacket->RSPIInit;
        Key_storage.generated_keys[SPI].Initiators_random_number = kepacket-
>GRandom_number_init;

        //сборка ответного пакета
        kepacket->RFlag = 32;
        kepacket->RMessage_number = 1;
        kepacket->RSPIResp = Key_storage.generated_keys[SPI].Respondents_SPI;
        kepacket->GRandom_number_init =
Key_storage.generated_keys[SPI].Respondents_random_number;

        //перенаправление и отправка пакета
        ip->daddr = ip->saddr;
        ip->saddr = Conf.IP;
        ip->tttl = 64;
        ip->id = htons(++Conf.IP_count);
        ip->tot_len = htons( sizeof(iphdr) + sizeof(udphdr) + sizeof(KEPacket) );
        ip->check = Internet_checksum( (unsigned short *)ip, htons(ip->tot_len));

        memcpy(eth->h_dest, &ARP_cache[ip->daddr][0], ETH_ALEN);
        memcpy(eth->h_source, Conf.MAC, ETH_ALEN);

        //отправка пакета с номером 1
        write(Conf.socket, (void *)&buffer[0], sizeof(ethhdr) + htons(ip->tot_len));

        //Расчет ключей
        Creating_a_session_keys ( Key_storage.generated_keys[SPI].local.session_key,
//Respondents_session_key
                                Key_storage.generated_keys[SPI].outside.session_key,
//Initiators_session_key
                                Key_storage.generated_keys[SPI].local.authentication_key,
//Respondents_authentication_key
                                Key_storage.generated_keys[SPI].outside.authentication_key,
//Initiators_authentication_key
                                Key_storage.main_key, //key
                                Key_storage.generated_keys[SPI].Initiators_random_number,
//Initiators_Random_number
                                Key_storage.generated_keys[SPI].Respondents_random_number);
//Respondents_Random_number

    }
} else if (kepacket->RMessage_number == 1) {
    if (Key_storage.generated_keys[SPI].exchange == 0) {

        unsigned long int temp_key_number = 0;
        unsigned long long int temp_key_serial_number = 0;
        //Копирование номера ключа и его серии
        memcpy(&temp_key_number, kepacket->BNumber_key_and_series,
sizeof(temp_key_number));
        memcpy(&temp_key_serial_number, kepacket->BNumber_key_and_series +
sizeof(temp_key_number), sizeof(temp_key_serial_number));

        if(temp_key_number != Key_storage.number && temp_key_serial_number !=
Key_storage.serial_number) return;

        Key_storage.generated_keys[SPI].exchange = 1; //Был принят пакет с номером 1

```

```

        //Расчет ключей
        Creating_a_session_keys (
Key_storage.generated_keys[SPI].outside.session_key, //Respondents_session_key
        Key_storage.generated_keys[SPI].local.session_key,
//Initiators_session_key
        Key_storage.generated_keys[SPI].outside.authentication_key,
//Respondents_authentication_key
        Key_storage.generated_keys[SPI].local.authentication_key,
//Initiators_authentication_key
        Key_storage.main_key, //key
        Key_storage.generated_keys[SPI].Initiators_random_number,
//Initiators_Random_number
        Key_storage.generated_keys[SPI].Respondents_random_number);
//Respondents_Random_number

        //Расчет авторизационного числа
        Key_storage.generated_keys[SPI].local.authorization_number =
Calculate_authentication_number(

Key_storage.generated_keys[SPI].Initiators_SPI, //Initiators_SPI

Key_storage.generated_keys[SPI].Respondents_SPI, //Respondents_SPI
        Conf.SPI, //Initiators_Esp_spi,

Key_storage.generated_keys[SPI].local.authentication_key); //Initiators_authentication_key

        //Сборка и пакета с номером 2
        AIPacket * aipacket = (AIPacket*)&buffer[sizeof(ethhdr) + sizeof(iphdr) +
sizeof(udp)];
        *aipacket = * new AIPacket();
        aipacket->RSPIInit = Key_storage.generated_keys[SPI].Initiators_SPI;
        aipacket->RSPIResp = Key_storage.generated_keys[SPI].Respondents_SPI;
        aipacket->RFlag = 8;
        aipacket->GEsp_spi = Conf.SPI;
        aipacket->RMessage_number = 2;
        aipacket->RVersion = Protocol_version;
        aipacket->YVendor = Vendor;

        //Копирование авторизационного числа инициатора
        for ( size_t i = 0; i <
Key_storage.generated_keys[SPI].local.authorization_number.size(); aipacket->BAuthorization[i]
= Key_storage.generated_keys[SPI].local.authorization_number[i], i++ );

        //перенаправление и отправка пакета
        ip->daddr = ip->saddr;
        ip->saddr = Conf.IP;
        ip->tttl = 64;
        ip->id = htons(++Conf.IP_count);
        ip->tot_len = htons( sizeof(iphdr) + sizeof(udphdr) + sizeof(AIPacket) );
        ip->check = Internet_checksum( (unsigned short *)ip, htons(ip->tot_len));

        memcpy(eth->h_dest, &ARP_cache[ip->daddr][0], ETH_ALEN);
        memcpy(eth->h_source, Conf.MAC, ETH_ALEN);

        //отправка пакета с номером 2
        write(Conf.socket, (void *)&buffer[0], sizeof(ethhdr) + htons(ip->tot_len));
    }
} else if (kepacket->RMessage_number == 2) {
    if (Key_storage.generated_keys[SPI].exchange == 1) {
        Key_storage.generated_keys[SPI].exchange = 2; //Был принят пакет с номером 2

        AIPacket * aipacket = (AIPacket*)&buffer[sizeof(ethhdr) + sizeof(iphdr) +
sizeof(udp)];

```

```

        //Расчет авторизационного числа
        Key_storage.generated_keys[SPI].outside.authorization_number =
Calculate_authentication_number(

Key_storage.generated_keys[SPI].Initiators_SPI, //Initiators_SPI

Key_storage.generated_keys[SPI].Respondents_SPI, //Respondents_SPI
        aipacket->GEsp_spi, //Initiators_Esp_spi,

Key_storage.generated_keys[SPI].outside.authentication_key);
//Initiators_authentication_key

        for ( size_t i = 0; i <
Key_storage.generated_keys[SPI].outside.authorization_number.size(); i++ )
            if ( Key_storage.generated_keys[SPI].outside.authorization_number[i] !=
aipacket->BAuthorization[i]) {
                Key_storage.generated_keys[SPI].flag = false;
                Key_storage.generated_keys[SPI].exchange = -1;
                return;
            }
        Key_storage.generated_keys[SPI].flag = true; //сеанс установлен
        //Вывод сообщения о установленном сеансе
        std::string message("установлен сеанс со шлюзом ");
        struct in_addr addr;
        addr.s_addr = ip->saddr;
        message.append(inet_ntoa(addr));
        notify_notification_show(notify_notification_new ("Эмулятор",
message.c_str(), 0), 0);

        //Сборка пакета с номером 3
        aipacket->GEsp_spi = Conf.SPI;
        aipacket->RMessage_number = 3;
        //Расчет авторизационного числа
        Key_storage.generated_keys[SPI].local.authorization_number =
Calculate_authentication_number(

Key_storage.generated_keys[SPI].Initiators_SPI, //Initiators_SPI

Key_storage.generated_keys[SPI].Respondents_SPI, //Respondents_SPI
        Conf.SPI, //Respondents_Esp_spi,

Key_storage.generated_keys[SPI].local.authentication_key);
//Respondents_authentication_key

        //Копирование авторизационного числа инициатора
        for ( size_t i = 0; i <
Key_storage.generated_keys[SPI].local.authorization_number.size(); aipacket->BAuthorization[i]
= Key_storage.generated_keys[SPI].local.authorization_number[i], i++ );

        //перенаправление и отправка пакета
        ip->daddr = ip->saddr;
        ip->saddr = Conf.IP;
        ip->tttl = 64;
        ip->id = htons(++Conf.IP_count);
        ip->tot_len = htons( sizeof(iphdr) + sizeof(udphdr) + sizeof(AIPacket) );
        ip->check = Internet_checksum( (unsigned short *)ip, htons(ip->tot_len));

        memcpy(eth->h_dest, &ARP_cache[ip->daddr][0], ETH_ALEN);
        memcpy(eth->h_source, Conf.MAC, ETH_ALEN);

        //отправка пакета с номером 3
        write(Conf.socket, (void *)&buffer[0], sizeof(ethhdr) + htons(ip->tot_len));
    }
} else if (kepacket->RMessage_number == 3) {

```

```

        if (Key_storage.generated_keys[SPI].exchange == 2) {
            Key_storage.generated_keys[SPI].exchange = 3; //Был принят пакет с номером 3

            AIPacket * aipacket = (AIPacket*)&buffer[sizeof(ethhdr) + sizeof(iphdr) +
sizeof(udp)];

            //Расчет авторизационного числа
            Key_storage.generated_keys[SPI].outside.authorization_number =
Calculate_authentication_number(
Key_storage.generated_keys[SPI].Initiators_SPI, //Initiators_SPI

Key_storage.generated_keys[SPI].Respondents_SPI, //Respondents_SPI
aipacket->GEsp_spi, //Initiators_Esp_spi,
Key_storage.generated_keys[SPI].outside.authentication_key);
//Initiators_authentication_key

            for ( size_t i = 0; i <
Key_storage.generated_keys[SPI].outside.authorization_number.size(); i++ )
                if ( Key_storage.generated_keys[SPI].outside.authorization_number[i] !=
aipacket->BAuthorization[i]) {
                    Key_storage.generated_keys[SPI].flag = false;
                    Key_storage.generated_keys[SPI].exchange = -1;
                    return;
                }
            Key_storage.generated_keys[SPI].flag = true; //сеанс установлен

            //Вывод сообщения о установленном сеансе
            std::string message("установлен сеанс со шлюзом ");
            struct in_addr addr;
            addr.s_addr = ip->saddr;
            message.append(inet_ntoa(addr));
            notify_notification_show(notify_notification_new ("Эмулятор",
message.c_str(), 0), 0);
        }
    }
}

void ESP_packet_processing (Key_bundle &Key_storage, std::vector < Route > &Route_table,
std::vector < std::pair < std::clock_t, std::vector < unsigned char > > >
&packet_queue,
std::map < unsigned int, std::vector < unsigned char > > & ARP_cache,
IConf& Conf, std::vector< unsigned char >& buffer, int const recieved_bytes,
bool make, unsigned int gateway_ip) {

    iphdr * ip_recieved = (iphdr *)&buffer[sizeof(ethhdr)];

    std::vector< unsigned char > send_buffer(ETH_DATA_LEN);

    ethhdr * send_eth = (ethhdr *)&send_buffer[0];
    iphdr * send_ip = (iphdr *)&send_buffer[sizeof(ethhdr)];

    ip_esp_hdr * esp = (ip_esp_hdr *)&buffer[sizeof(ethhdr) + sizeof(iphdr)];

    Cypher * cypher = new GOST_28147_89;//

    if (make) {
        for (Route &route: Route_table) { //Проверка подсети в которую необходимо отправить
пакет
            if (gateway_ip == route.gateway) {

                ip_recieved->check = 0;

```

```

        ip_recieved->check = Internet_checksum( (unsigned short *)ip_recieved,
        htons(ip_recieved->tot_len));

        //Заполнение ip заголовка
        memset(send_ip, 0, sizeof(iphdr));
        send_ip->daddr = gateway_ip;
        send_ip->saddr = Conf.IP;
        send_ip->id = htons(++Conf.IP_count);
        send_ip->ttl = 64;
        send_ip->protocol = IPPROTO_ESP;
        send_ip->version = 4;
        send_ip->ihl = 5;

        //Заполнение ESP заголовка
        esp = (ip_esp_hdr *)&send_buffer[sizeof(ethhdr) + sizeof(iphdr)];
        esp->spi = Conf.SPI;
        esp->seq_no = htonl((unsigned long int)Conf.IP_count);

        //инициализация ключа
        cypher->SetKey( (unsigned char *)
        &Key_storage.generated_keys[route.SPI].local.session_key[0], //Данные ключа
        Key_storage.generated_keys[route.SPI].local.session_key.size() *
        4); //Его размер в байтах

        unsigned short encrypted_bytes = htons(ip_recieved->tot_len) + (8 -
        (htons(ip_recieved->tot_len)%8));

        cypher->Encrypt((const unsigned char *)ip_recieved, encrypted_bytes, esp-
        >enc_data); //Зашифровывание принятого ip пакета и добавление его

        send_ip->tot_len = htons(sizeof(iphdr) + sizeof(esp->seq_no) + sizeof(esp-
        >spi) + encrypted_bytes);
        send_ip->check = Internet_checksum( (unsigned short *)send_ip,
        htons(send_ip->tot_len));

        //L2
        memcpy(send_eth->h_dest, &ARP_cache[gateway_ip][0], ETH_ALEN);
        memcpy(send_eth->h_source, Conf.MAC, ETH_ALEN);
        send_eth->h_proto = htons(ETH_P_IP);

        write(Conf.socket, (void *)&send_buffer[0], sizeof(ethhdr) + htons(send_ip-
        >tot_len));
    }
} else {
    for (Route &route: Route_table) { //Проверка подсети из которой пришел пакет
        if (ip_recieved->saddr == route.gateway) {

            //инициализация ключа
            cypher->SetKey( (unsigned char *)
            &Key_storage.generated_keys[route.SPI].outside.session_key[0], //Данные ключа
            Key_storage.generated_keys[route.SPI].outside.session_key.size()
            * 4); //Его размер в байтах

            unsigned int eb = recieved_bytes - (sizeof(ethhdr) + ip_recieved->ihl*4 +
            sizeof(esp->seq_no) + sizeof(esp->spi));

            cypher->Decrypt((const unsigned char *) (esp->enc_data), (unsigned char
            *)send_ip, eb); //L2
            memcpy(send_eth->h_source, Conf.MAC, ETH_ALEN);
            send_eth->h_proto = htons(ETH_P_IP);

```

```

        if (ARP_cache.find(send_ip->daddr) == ARP_cache.end()) {
            ARP_packet_processing(ARP_cache, Conf, Conf, buffer, ARPOP_REQUEST, send_ip->daddr);
            packet_queue.push_back ( std::make_pair( std::clock(),
std::vector<unsigned char>( send_buffer.begin(), send_buffer.begin() + htons(send_ip->tot_len) + sizeof(ethhdr))));
        } else {
            memcpy(send_eth->h_dest, &ARP_cache[send_ip->daddr][0], ETH_ALEN);
            write (Conf.socket, &send_buffer[0], htons(send_ip->tot_len) +
sizeof(ethhdr));
        }
    }
}
}
}

```

Исходный код файла Authentication.hpp

```

#pragma once
#include <vector>
#include <algorithm>

unsigned long int GOST_MAC ( std :: vector < unsigned char > &, std::vector < unsigned long
int > & );

void Creating_a_session_keys ( std::vector < unsigned long int > &, std::vector < unsigned
long int > &,
                                std::vector < unsigned long int > &,
std::vector < unsigned long int > &,
                                std::vector < unsigned long int > &,
unsigned long long int, unsigned long long int );

std::vector <unsigned long int > Calculate_authentication_number( unsigned long long int,
unsigned long long int, unsigned long long int, std::vector < unsigned long int > &, bool =
false );

#pragma pack(push, 1)
struct KEPacket {
    unsigned long long int RSPIInit;
    unsigned long long int RSPIResp;
    unsigned char RNext_Payload;
    unsigned char RVersion;
    unsigned char RExchange_Type;
    unsigned char RFlag;
    unsigned int RMessage_number;
    unsigned int RMessage_length;
    unsigned char YNext_Payload;
    unsigned char YFlags;
    unsigned short YLen;
    unsigned long long int YVendor;
    unsigned char GNext_Payload;
    unsigned char GFlags;
    unsigned short GLen;
    unsigned long long int GRandom_number_init;
    unsigned char BNext_Payload;
    unsigned char BFlags;
    unsigned short BLen;
    unsigned int BType_and_reserve;//первый байт тип 2-4 резерв
    unsigned char BNumber_key_and_series[12];

    KEPacket();
};

```

```

struct AIPacket {
    unsigned long long int RSPIInit;
    unsigned long long int RSPIResp;
    unsigned char RNext_Payload;
    unsigned char RVersion;
    unsigned char RExchange_Type;
    unsigned char RFlag;
    unsigned int RMessage_number;
    unsigned int RMessage_length;
    unsigned char YNext_Payload;
    unsigned char YFlags;
    unsigned short YLen;
    unsigned long long int YVendor;
    unsigned char GNext_Payload;
    unsigned char GFlags;
    unsigned short GLen;
    unsigned char GP_more;
    unsigned char GP_reserved;
    unsigned short GP_len;
    unsigned char GP_num;
    unsigned char GP_id;
    unsigned char GP_spi_size;
    unsigned char GP_n_trans;
    unsigned int GEsp_spi;
    unsigned char GT_more;
    unsigned char GT_reserved1;
    unsigned short GT_len;
    unsigned char GT_type;
    unsigned char GT_reserved2;
    unsigned short GT_id;
    unsigned char BNext_Payload;
    unsigned char BFlags;
    unsigned short BLen;
    unsigned int BMethod_and_reserve;//первый байт метод 2-4 резерв
    unsigned long int BAuthorization[4];

    AIPacket();
};
#pragma pack(pop)

```

Исходный код файла Authentication.cpp

```

#include "Authentication.hpp"

KEPacket::KEPacket(){

    unsigned char *ptr = (unsigned char*) this;
    for ( int i = 0; i < sizeof(KEPacket); *(ptr+i)=0, i++ );
    ptr = nullptr;

    this->RNext_Payload = 43;
    this->RVersion = 32;
    this->RExchange_Type = 34;
    this->RMessage_length = 68;
    this->YNext_Payload = 40;
    this->YFlags = 128;
    this->YLen = 12;
    this->GNext_Payload = 35;
    this->GFlags = 128;
    this->GLen = 12;
    this->BFlags = 128;
    this->BLen = 16;
    this->BType_and_reserve = 0xB000;//11<<12
}

```

```

AIPacket::AIPacket(){
    unsigned char *ptr = (unsigned char*) this;
    for ( int i = 0; i < sizeof(AIPacket); *(ptr+i)=0, i++ );
    ptr = nullptr;

    this->RNext_Payload = 43;
    this->RVersion = 32;
    this->RExchange_Type = 34;
    this->RMessage_length = 68;
    this->YNext_Payload = 40;
    this->YFlags = 128;
    this->YLen = 12;
    this->GNext_Payload = 39;
    this->GFlags = 128;
    this->GLen = 24;
    this->GP_len = 20;
    this->GP_num = 1;
    this->GP_id = 3;
    this->GP_spi_size = 4;
    this->GP_n_trans = 1;
    this->GT_len = 8;
    this->GT_type = 1;
    this->GT_id = 1024;

    ///////////////////////////////////
    this->BFlags = 128;
    this->BLen = 24;
    this->BMethod_and_reserve = 0x2000;//2<<12
}

std::vector< unsigned long int > Calculate_authentication_number( unsigned long long int
Initiators_SPI, unsigned long long int Respondents_SPI, unsigned long long int Esp_spi,
std::vector< unsigned long int > & key, bool flag ) {
    std::vector< unsigned char > temp ( 24, 0 );
    std::vector< unsigned long int > result;

    for( size_t i = 0; i < sizeof(unsigned long long int); i++ )
        temp [ i ] = (unsigned char)( Initiators_SPI >> ( ( sizeof ( unsigned long long int
) * 8 ) - ( ( i + 1 ) * 8 ) ) );

    for( size_t i = 0, j = 8; i < sizeof(unsigned long long int); i++, j++ )
        temp [ j ] = (unsigned char) ( Respondents_SPI >> ( ( sizeof ( unsigned long long
int ) * 8 ) - ( ( i + 1 ) * 8 ) ) );

    for( size_t i = 0, j = 16; i < sizeof(unsigned long int); i++, j++ )
        temp [ j ] = (unsigned char) ( Esp_spi >> ( ( sizeof ( unsigned long int ) * 8 ) - (
( i + 1 ) * 8 ) ) );

    if ( !flag ) temp [ 23 ] = 1; //для инициатора
    else temp [ 23 ] = 5; //для респондента

    unsigned long int ICV = GOST_MAC ( temp, key ); //ICV(1)
    result.push_back ( ICV );

    std::rotate ( temp.begin(), temp.end() - 4, temp.end() );
    temp.resize(32);

    if ( !flag ) temp [ 27 ] = temp.size() + 1; //для инициатора
    else temp [ 27 ] = temp.size() + 5; //для респондента

    for ( size_t i = 0; i < 3; i++ ) {

        temp [ 0 ] = ( unsigned char ) ( ( ICV >> 24 ) & 0xff );
        temp [ 1 ] = ( unsigned char ) ( ( ICV >> 16 ) & 0xff );
    }
}

```

```

        temp [ 2 ] = ( unsigned char ) ( ( ICV >> 8 ) & 0xff );
        temp [ 3 ] = ( unsigned char ) ( ICV & 0xff );

        ICV = GOST_MAC ( temp, key ); //ICV(all)
        result.push_back ( ICV );
    }
    return result;
}

void Creating_a_session_keys ( std::vector < unsigned long int > & Respondents_session_key,
std::vector < unsigned long int > & Initiators_session_key,
std::vector < unsigned long int > &
Respondents_authentication_key, std::vector < unsigned long int > &
Initiators_authentication_key,
std::vector < unsigned long int > & main_key, unsigned long long
int Initiators_random_number, unsigned long long int Respondents_random_number) {

    std::vector <unsigned char> temp ( 24, 0 );

    for( size_t i = 0; i < sizeof(unsigned long long int); i++ )
        temp [ i ] = (unsigned char)( Respondents_random_number >> ( ( sizeof ( unsigned
long long int ) * 8 ) - ( ( i + 1 ) * 8 ) ) );

    for( size_t i = 0, j = 8; i < sizeof(unsigned long long int); i++, j++ )
        temp [ j ] = (unsigned char) ( Initiators_random_number >> ( ( sizeof ( unsigned
long long int ) * 8 ) - ( ( i + 1 ) * 8 ) ) );

    unsigned long int ICV = GOST_MAC ( temp, main_key ); //ICV(1)
    Initiators_session_key.push_back ( ICV );

    std::rotate ( temp.begin(), temp.end() - 4, temp.end() );

    for ( size_t i = 0; i < 31; i++ ) {

        temp [ 0 ] = ( unsigned char ) ( ( ICV >> 24 ) & 0xff );
        temp [ 1 ] = ( unsigned char ) ( ( ICV >> 16 ) & 0xff );
        temp [ 2 ] = ( unsigned char ) ( ( ICV >> 8 ) & 0xff );
        temp [ 3 ] = ( unsigned char ) ( ICV & 0xff );

        ICV = GOST_MAC ( temp, main_key ); //ICV(all)

        if ( Initiators_session_key.size() != 8 ) Initiators_session_key.push_back (
ICV );
        else if ( Respondents_session_key.size() != 8 )
Respondents_session_key.push_back ( ICV );
        else if ( Initiators_authentication_key.size() != 8 )
Initiators_authentication_key.push_back ( ICV );
        else if ( Respondents_authentication_key.size() != 8 )
Respondents_authentication_key.push_back ( ICV );

    }
}

unsigned long int GOST_MAC ( std :: vector < unsigned char > & buffer, std::vector <
unsigned long int > & key){

    unsigned int S_box[8][16] =
    {{0xC,0x4,0x6,0x2,0xA,0x5,0xB,0x9,0xE,0x8,0xD,0x7,0x0,0x3,0xF,0x1},
    {0x6,0x8,0x2,0x3,0x9,0xA,0x5,0xC,0x1,0xE,0x4,0x7,0xB,0xD,0x0,0xF},
    {0xB,0x3,0x5,0x8,0x2,0xF,0xA,0xD,0xE,0x1,0x7,0x4,0xC,0x9,0x6,0x0},
    {0xC,0x8,0x2,0x1,0xD,0x4,0xF,0x6,0x7,0x0,0xA,0x5,0x3,0xE,0x9,0xB},
    {0x7,0xF,0x5,0xA,0x8,0x1,0x6,0xD,0x0,0x9,0x3,0xE,0xB,0x4,0x2,0xC},
    {0x5,0xD,0xF,0x6,0x9,0x2,0xC,0xA,0xB,0x7,0x8,0x1,0x4,0x3,0xE,0x0},
    {0x8,0xE,0x2,0x5,0x6,0x9,0x1,0xC,0xF,0x4,0xB,0x0,0xD,0xA,0x3,0x7},
    {0x1,0x7,0xE,0xD,0x0,0x5,0x8,0x3,0x4,0xF,0xA,0x6,0x9,0xC,0xB,0x2}};

```

```

long unsigned int N[2] = { 0, 0}, MAC[2] = { 0, 0}, temp;
for( size_t i = 0; i < buffer.size(); i += 2 ) {

    //Заполнение регистров N1 и N2
    for ( int j = 0; j < 2; j++) {
        for ( int k = 0; k < 4; k++ ) {
            N[j] = N[j] | ( ( unsigned long int ) buffer [ j ] ) << (8 * k) );
        }
    }

    //16 раундов шифрования
    for ( int t = 0; t < 16; t++ ) {
        temp = 0;

        //(N1 + key_round) mod 2^32
        //Сложение блока N1 с раундовым ключом по mod 2^32
        temp = (N[0] + key[t % 8]) & 0xFFFFFFFF;

        //Подготовка блока размером 32 бита
        unsigned int Vector [ 8 ];
        for ( int n = 0; n < 8; n++) {
            //Разбиение блока на 8 4-разрядных векторов
            Vector[n] = (temp >> n * 4) & 0xF;
            //Подстановка в S-блоке
            Vector[n] = S_box[n][Vector[n]];
        }
        temp = 0;
        //Объединение выходных векторов в блок размером 32 бита
        for ( int n = 0; n < 8; n++ )
            temp = temp | ( Vector [ n ] << n * 4 );

        //Циклический сдвиг в сторону старших разрядов на 11 бит
        temp = ( temp << 11 ) | ( temp >> 21 );

        //Побитовое сложение по mod2 результата с N2
        temp = temp ^ N [ 1 ];
        //Для раундов 1 - 31 шифрования
        N [ 1 ] = N [ 0 ];
        N [ 0 ] = temp;

    }
    MAC [ 0 ] ^= N [ 0 ];
    MAC [ 1 ] ^= N [ 1 ];
}
return MAC [ 0 ];
}

```

Исходный код файла Cypher.hpp

```

#pragma once

class Cypher{
public:
    virtual void SetKey( const unsigned char * key, const unsigned int &key_size ) = 0;
    virtual void Encrypt( const unsigned char * open_buffer, const unsigned int
open_buffer_size, unsigned char * encrypted_buffer ) = 0;
    virtual void Decrypt( const unsigned char * encrypted_buffer, unsigned char *
open_buffer, const unsigned int open_buffer_size ) = 0;
    virtual ~Cypher(){};
};

```

Исходный код файла GOST_28147_89.hpp

```

#pragma once

```

```
#include "Cypher.hpp"

class GOST_28147_89 : public Cypher {
private:
    unsigned long * key;
    void GOST_ECB( const unsigned char * input, unsigned int input_size, unsigned char *
output,unsigned char work_mode);
public:
    GOST_28147_89();

    void SetKey( const unsigned char * _key, const unsigned int &key_size );
    void Encrypt( const unsigned char * open_buffer, const unsigned int open_buffer_size,
unsigned char * encrypted_buffer );
    void Decrypt( const unsigned char * encrypted_buffer, unsigned char * open_buffer,
const unsigned int open_buffer_size );

    ~GOST_28147_89();
};
```

Исходный код файла GOST_28147_89.cpp

```
#include "GOST_28147_89.hpp"

GOST_28147_89::GOST_28147_89(){
    key = new unsigned long [ 8 ];
    for ( int i = 0; i < 8; i++ )
        key [ i ] = 0;
}
//work_mode 0 - encrypt mode
//work_mode 1 - decrypt mode
void GOST_28147_89::GOST_ECB( const unsigned char * input, unsigned int input_size, unsigned
char * output,unsigned char work_mode){

    //id-tc26-gost-28147-param-Z
    unsigned int S_box[8][16] =
    {
        {0xC,0x4,0x6,0x2,0xA,0x5,0xB,0x9,0xE,0x8,0xD,0x7,0x0,0x3,0xF,0x1},
        {0x6,0x8,0x2,0x3,0x9,0xA,0x5,0xC,0x1,0xE,0x4,0x7,0xB,0xD,0x0,0xF},
        {0xB,0x3,0x5,0x8,0x2,0xF,0xA,0xD,0xE,0x1,0x7,0x4,0xC,0x9,0x6,0x0},
        {0xC,0x8,0x2,0x1,0xD,0x4,0xF,0x6,0x7,0x0,0xA,0x5,0x3,0xE,0x9,0xB},
        {0x7,0xF,0x5,0xA,0x8,0x1,0x6,0xD,0x0,0x9,0x3,0xE,0xB,0x4,0x2,0xC},
        {0x5,0xD,0xF,0x6,0x9,0x2,0xC,0xA,0xB,0x7,0x8,0x1,0x4,0x3,0xE,0x0},
        {0x8,0xE,0x2,0x5,0x6,0x9,0x1,0xC,0xF,0x4,0xB,0x0,0xD,0xA,0x3,0x7},
        {0x1,0x7,0xE,0xD,0x0,0x5,0x8,0x3,0x4,0xF,0xA,0x6,0x9,0xC,0xB,0x2}};

    long unsigned int N[2], temp;

    // Заполнение двух блоков по 32 бита
    N[0] = 0; N[1] = 0;
    for( unsigned int i = 0, k = 0; i < 2; i++)
    {
        for( unsigned int j = 0; j < 4; j++, k++)
        {
            long unsigned int s;

            if ( k < input_size ) s = ( long unsigned int ) input [ k ];
            else s = 0;

            N[i] = N[i] | ( s << 8 * j);
        }
    }
    //32 раунда шифрования
    for (int t=0; t<32; t++)
    {
```

```

temp = 0;

//(N1 + key_roud) mod 2^32
//Сложение блока N1 с раундовым ключом по mod 2^32

//Для режима зашифрования
if (!work_mode)
{
    //Для 1 - 24 раундов K_0 - K_7
    if (t < 24)
    {
        temp = (N[0] + key[t % 8]) & 0xFFFFFFFF;
    }
    //Для 25 - 32 раундов K_7 - K_0
    if (t >= 24)
    {
        temp = (N[0] + key[7 - (t % 8)]) & 0xFFFFFFFF;
    }
}
//Для режима расшифрования
if (work_mode)
{
    //Для 1 - 8 раундов K_0 - K_7
    if (t < 8)
    {
        temp = (N[0] + key[t % 8]) & 0xFFFFFFFF;
    }
    //Для 9 - 32 раундов K_7 - K_0
    if (t >= 8)
    {
        temp = (N[0] + key[7 - (t % 8)]) & 0xFFFFFFFF;
    }
}

//Подготовка блока размером 32 бита
unsigned int Vector[8];
    int n;
for (n=0; n<8; n++)
{
    //Разбиение блока на 8 4-разрядных векторов
    Vector[n] = (temp >> n * 4) & 0xF;
    //Подстановка в S-блоке
    Vector[n] = S_box[n][Vector[n]];
}
temp = 0;
for (n=0; n<8; n++)
{
    //Объединение выходных векторов в блок размером 32 бита
    temp = temp | (Vector[n] << n * 4);
}

//Циклический сдвиг в сторону старших разрядов на 11 бит
temp = (temp << 11) | (temp >> 21);

//Побитовое сложение по mod2 результата с N2
temp = temp ^ N[1];

//Для раундов 1 - 31 шифрования
if (t < 31)
{
    N[1] = N[0];
    N[0] = temp;
}
//Для последнего раунда шифрования

```

```

        if (t == 31)
        {
            N[1] = temp;
        }
    } //Цикл: 32 раунда шифрования

    //Сохранение преобразованных блоков
    for(int i = 0, k = 0; i < 2; i++)
    {
        for(int j = 0; j < 4; j++, k++)
        {
            temp = N[i] & 0xFF;
            output [ k ] = (unsigned char) temp;
            N[i] = N[i] >> 8;
        }
    }
}

void GOST_28147_89::SetKey( const unsigned char * _key, const unsigned int &key_size ){
    for(int i = 0, k = 0; i < 8; i++)
    {
        for(int j = 0; j < 4; j++, k++)
        {
            key[i] = key[i] | ( (unsigned long)_key[k] << 8 * j);
        }
    }
}

void GOST_28147_89::Encrypt( const unsigned char * open_buffer, const unsigned int
open_buffer_size, unsigned char * encrypted_buffer ){
    for ( unsigned int i = 0; i < open_buffer_size; i += 8 ){
        if ( ( open_buffer_size - i ) > 8 )
            GOST_ECB( open_buffer + i, 8, encrypted_buffer + i, 0 );
        else
            GOST_ECB( open_buffer + i, open_buffer_size - i, encrypted_buffer + i,
0 );
    }
}

void GOST_28147_89::Decrypt( const unsigned char * encrypted_buffer, unsigned char *
open_buffer, const unsigned int open_buffer_size ){
    for ( unsigned int i = 0; i < open_buffer_size; i += 8 ){
        if ( ( open_buffer_size - i ) > 8 )
            GOST_ECB( encrypted_buffer + i, 8, open_buffer + i, 1 );
        else
            GOST_ECB( encrypted_buffer + i, open_buffer_size - i, open_buffer + i,
1 );
    }
}

GOST_28147_89::~GOST_28147_89(){
    delete [] key;
    key = nullptr;
}

```

Приложение Б
(обязательное)
Протокол проверки ВКР в системе «Антиплагиат»

Протокол

проверки на оригинальность в системе «Антиплагиат.ВУЗ» выпускной квалификационной работы на тему «Программный комплекс эмуляции устройства шифрования для IP-сети» студента 6 курса С.А. Борисова специальности 10.05.02 – «Информационная безопасность телекоммуникационных систем».

Руководитель ВКР инженер 1 категории НИС-45 АО «ПНИЭИ» С.П. Хворостухин.

Отчет о проверке на заимствования

Автор: Борисов Сергей Андреевич borisov-12pt1@yandex.ru / ID: 639

Проверяющий: Фатеев Алексей Григорьевич (ibst@pnzgu.ru) / ID: 393

Организация: Пензенский гос. Университет

Отчет предоставлен сервисом «Антиплагиат» - <http://pnzgu.antiplagiat.ru>

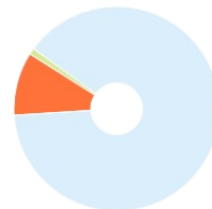
ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 34
Начало загрузки: 15.01.2018 13:23:37
Длительность загрузки: 00:00:08
Имя исходного файла: Борисов Сергей Андреевич ВКР_12ПТ1_Борисов.docx
Размер текста: 2700 кБ
Символов в тексте: 138683
Слов в тексте: 17357
Число предложений: 883

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Последний готовый отчет (ред.)
Начало проверки: 15.01.2018 22:49:44
Длительность проверки: 00:00:20
Комментарии: не указано
Модули поиска: Модуль поиска ЭБС "БиблиоРоссика", Модуль поиска ЭБС "BOOK.ru", Коллекция РГБ, Цитирование, Модуль поиска ЭБС "Университетская библиотека онлайн", Коллекция eLIBRARY.RU, Модуль поиска ЭБС "Айбукс", Модуль поиска Интернет, Модуль поиска ЭБС "Лань", Модуль поиска "ПГУ", Кольцо вузов

ЗАИМСТВОВАНИЯ	ЦИТИРОВАНИЯ	ОРИГИНАЛЬНОСТЬ
9,75%	0,38%	89,87%



16.01.2018

____ С.П. Хворостухин