

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 100503 —

Информационная безопасность
автоматизированных систем

Специализация —

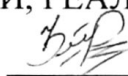
Защищенные автоматизированные
системы управления

ДИПЛОМНЫЙ ПРОЕКТ

на тему

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ УСТРОЙСТВА КРИПТОГРАФИЧЕСКОЙ
ЗАЩИТЫ ИНФОРМАЦИИ, РЕАЛИЗОВАННОЕ НА ПЭВМ

Студент

 Зиманов Дмитрий Андреевич

Руководитель

 Бубнов А.В.

Нормоконтролёр

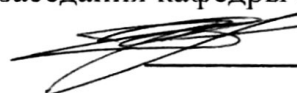
 Иванов А.П.

Рецензент

 Меримова Н.В.

Работа допущена к защите (протокол заседания кафедры от 26.06.18 № 16)

Заведующий кафедрой

 С.Л. Зефирова

Работа защищена с отметкой ОТЛИЧНО (протокол заседания ГЭК
от 27.06.18 № 3)

Секретарь ГЭК

 А.П. Иванов

Пенза 2018

 Проверка ВКР-1086110
22.06.18г

Оглавление

Введение.....	9
1 Общие сведения об устройстве криптографической защиты информации.....	11
2 Разработка алгоритмов работы модулей программного обеспечения.....	13
2.1 Разработка алгоритма работы интерфейсного модуля.....	13
2.2 Разработка алгоритма работы модулей шифрования.....	13
2.2.1 Общий принцип работы.....	13
2.2.2 Реализация шифрования.....	15
3 Разработка алгоритма взаимодействия модулей программного обеспечения.....	18
3.1 Алгоритм взаимодействия модулей программного обеспечения.....	18
3.2 Идентификация угроз безопасности информации, передаваемой при взаимодействии модулей.....	19
3.3 Выбор защитных мер.....	21
4 Разработка протокола взаимодействия интерфейсного модуля и модуля шифрования.....	24
4.1 Общие положения.....	24
4.2 Форматы кадров при обмене данными, командами, сообщениями и квитанциями.....	24
4.2.1 Общий формат.....	24
4.2.2 Структура и состав информационных (исполняемых) байт кадров.....	25
4.3 Команды от интерфейсного модуля в модуль шифрования....	26
4.4 Команды от модуля шифрования в интерфейсный модуль.....	27
4.5 Квитанции от модуля шифрования в интерфейсный модуль..	28

4.6	Квитанции от интерфейсного модуля в модуль шифрования.....	28
5	Разработка программного обеспечения устройства криптографической защиты информации.....	30
5.1	Общие сведения.....	30
5.2	Описание модулей, входящих в состав программного обеспечения.....	30
5.2.1	Описание программного обеспечения модуля шифрования «Кузнечик».....	30
5.2.2	Описание программного обеспечения модуля шифрования «Магма».....	38
5.2.3	Описание программного обеспечения технологического интерфейсного модуля.....	42
6	Разработка руководства пользователя программного обеспечения..	45
6.1	Общие сведения.....	45
6.2	Интерфейсный модуль.....	45
6.3	Модуль шифрования «Кузнечик».....	47
6.4	Модуль маршрутизации.....	49
6.5	Модуль шифрования «Магма».....	51
6.6	Модуль приемопередатчик.....	52
7	Тестирование программного обеспечения.....	54
7.1	Тестирование модуля шифрования «Кузнечик».....	54
7.2	Тестирование модуля шифрования «Магма».....	55
7.3	Тестирование квитирования сообщений.....	56
7.4	Тестирование вычисления контрольной суммы.....	57
8	Вредные психофизиологические факторы, влияющие на нервную систему специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие.....	58
	Заключение.....	60

Список используемых источников.....	61
Приложение А Алгоритм взаимодействия интерфейсный модуль-модуль шифрования.....	62
Приложение Б Алгоритм взаимодействия модуль шифрования-интерфейсный модуль.....	65
Приложение В Алгоритм взаимодействия модуль шифрования-модуль маршрутизации.....	67
Приложение Г Исходный текст программного обеспечения модуля шифрования «Кузнечик».....	69
Приложение Д Исходный текст программного обеспечения модуля шифрования «Магма».....	109
Приложение Е Исходный текст программного обеспечения интерфейсного модуля.....	129

Термины и определения

В настоящем отчете применяют следующие термины с соответствующими определениями:

администратор безопасности — привилегированный пользователь, обеспечивающий штатное функционирование средств и системы защиты на основе реализации положений политики безопасности;

атака — целенаправленные действия с целью осуществления НСД к информации или с целью создания условий для этого;

доступ к информации — ознакомление с информацией, её обработка, в частности копирование, модификация или уничтожение информации;

защита информации — деятельность, направленная на предотвращение утечки защищаемой информации, несанкционированных и непреднамеренных воздействий на защищаемую информацию;

защищаемая информация — информация, подлежащая защите в соответствии с требованиями правовых документов или требованиями, устанавливаемыми собственниками информации;

канал атаки — среда переноса между субъектом и объектом атаки действий, осуществляемых при проведении атаки;

контролируемая зона — пространство, в пределах которого осуществляется контроль за пребыванием и действиями лиц и (или) транспортных средств;

модель нарушителя — предположения о возможностях нарушителя, которые он может использовать для разработки и проведения атак, а также об ограничениях на эти возможности;

нарушитель (субъект атаки) — лицо (или иницилируемый им процесс), проводящее (проводящий) атаку;

носитель информации — физическое лицо или материальный объект, в том числе физическое поле, в которых информация находит свое отображение в

виде символов, образов, сигналов, технических решений и процессов;

обработка информация — совокупность операций сбора, накопления, ввода, вывода, приема, передачи, записи, хранения, регистрации, уничтожения, преобразования, отображения информации;

объект доступа — единица информационного ресурса автоматизированной системы, доступ к которой регламентируется правилами разграничения доступа;

правило доступа к информации (правило доступа) — совокупность правил, регламентирующих порядок и условия доступа субъекта к информации и ее носителям;

право доступа к информации (право доступа) — совокупность правил доступа к информации, установленных правовыми документами или собственником, владельцем информации;

средство защиты информации — техническое, программное средство, вещество и (или) материал, предназначенные или используемые для защиты информации;

субъект доступа — пользователь автоматизированной системы или иницилируемый им процесс, действия которого регламентируются правилами разграничения доступа.

Обозначения и сокращения

В настоящем отчете применяют следующие сокращения:

АПД — аппаратура передачи данных;

АО — акционерное общество;

ВКР — выпускная квалификационная работа;

ЗМ — защитные меры;

ИА — информационные активы;

ИБ — информационная безопасность;

ИВ — имитовставка;

КОИ — криптоопасная информация;

КИ — ключевая информация;

МИ — модуль интерфейсный;

ММАР — модуль маршрутизатора;

МСКЗИ — модуль средства криптографической защиты информации;

МШ — модуль шифрования;

НДВ — недеklarированные возможности;

НСД — несанкционированный доступ;

ОС — операционная система;

ПД — передача данных;

ПО — программное обеспечение;

ПП — приемопередатчик;

ПЭВМ — персональная электронная вычислительная машина;

СЗИ — средство защиты информации;

СКЗИ — средство криптографической защиты информации;

ТЗ — техническое задание;

У — угрозы;

CRC — cyclic redundancy check (циклический избыточный код);

FIFO — first in, first out;

IP — internet protocol;

IV — синхропосылка;

UDP — user datagram protocol (протокол пользовательских датаграмм).

Введение

На сегодняшний день информация является активом, к которому проявляется интерес различных сторон по той или иной причине. Предприятия, владеющие большим количеством конфиденциальной информации, тратят значительные средства на их сохранность, потому что злоумышленники каждый день прогрессируют и находят новые способы хищения данных. Для того чтобы злоумышленники не добились желаемого, используются различные средства защиты информации, в частности использование криптографических методов в информационных системах.

Использование таких методов также является актуальным потому, что расширилось использование компьютерных сетей, в частности глобальной сети Интернет, по которым передаются большие объемы информации государственного, военного, коммерческого и частного характера, не допускающего возможность доступа к ней посторонних лиц.

Актуальность темы ВКР обусловлена тем, что на сегодняшний день остро стоит вопрос о защите информации от несанкционированного доступа, поэтому использование различных средств защиты информации, в частности использование криптографических методов является востребованным во многих предприятиях.

Целью данного дипломного проекта является разработка программного обеспечения устройства криптографической защиты информации, реализованного на ПЭВМ.

В устройстве криптографической защиты информации можно выделить несколько модулей. В рамках дипломного проекта была проведена программная реализация интерфейсного модуля и модуля шифрования. Модуль шифрования должен выполнять двухконтурное шифрование по двум алгоритмам блочного шифрования, приведенных в ГОСТ Р 34.12-2015 с режимами работы, приведенными в ГОСТ Р 34.13-2015.

Была произведена разработка алгоритмов работы модулей программного обеспечения устройства криптографической защиты информации (интерфейсного модуля, модуля шифрования), алгоритма взаимодействия модулей программного обеспечения, протоколов взаимодействия модулей программного обеспечения, руководства пользователя программного обеспечения. Произведен обзор вредных психофизиологических факторов, влияющих на нервную систему специалиста по защите информации.

Было проведено тестирование модулей программного обеспечения по контрольным примерам, приведенным в ГОСТ Р 34.13-2015.

1 Общие сведения о программной реализации устройства криптографической защиты информации

Программный комплекс состоит из пяти модулей:

- интерфейсного модуля;
- модуля шифрования (с алгоритмом шифрования ГОСТ 34.12-2015, 64 бита);
- модуля маршрутизатора (ММАР);
- модуля шифрования (с алгоритмом шифрования ГОСТ 34.12-2015, 128 бит);
- приемопередатчика (ПП).

Модули взаимодействуют между собой через сетевой протокол UDP посредством информационно-логического протокола. Взаимодействие модулей представлено на рисунке 1.

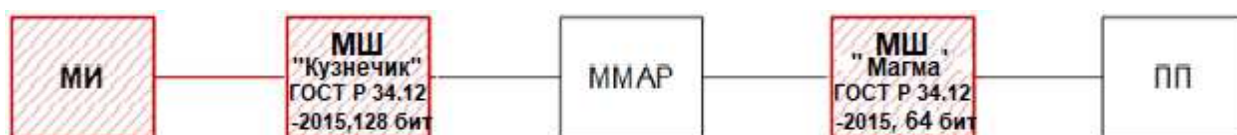


Рисунок 1 — Взаимодействие модулей программного обеспечения

В рамках дипломного проекта выполняется реализация интерфейсного модуля (МИ), модулей шифрования (МШ) и протокола взаимодействия между МИ и МШ (на рисунке 1 выделено красным).

МИ разбивает исходные данные на блоки. В соответствии с протоколом взаимодействия модулей МИ и МШ сформированные блоки из МИ передаются в МШ.

МШ «Кузнечик» полностью зашифровывает принятые подблоки данных и добавляет к ним данные, необходимые для их расшифрования. В соответствии с протоколом взаимодействия модулей МШ и ПП сформированные подблоки из МШ передаются в ПП. Для реализации шифрования применяется блочный шифр «Кузнечик» в режиме гаммирования, с выработкой имитовставки в

соответствии с ГОСТ Р 34.13-2015.

ММАР разбивает каждый принятый блок на подблоки. В соответствии с протоколом взаимодействия модулей ММАР и МСКЗИ сформированные подблоки из ММАР передаются в МШ «Кузнечик».

МШ «Магма» полностью зашифровывает принятые блоки данных и добавляет к ним данные, необходимые для их расшифрования. В соответствии с протоколом взаимодействия модулей МШ и ММАР, сформированные блоки из МШ передаются в ММАР. Для реализации шифрования применяется блочный шифр «Магма» в режиме гаммирования, с выработкой имитовставки в соответствии с ГОСТ Р 34.13-2015.

Модуль ПП предназначен для ретрансляции данных и формирования ошибок при передаче по каналу связи.

Модули ММАР и ПП не разрабатываются в рамках ВКР, и предоставляются руководителем дипломного проектирования в АО «ПНИЭИ» для тестирования МИ и МШ.

2 Разработка алгоритмов работы модулей программного обеспечения

2.1 Разработка алгоритма работы интерфейсного модуля

МИ разбивает исходные данные на блоки, кратные 16 байтам. Максимальный размер блока 1536 байт. К каждому блоку добавляется заголовок №1, где указывается номер блока, размер блока и общее количество блоков исходных данных. В соответствии с протоколом взаимодействия модулей МИ и МШ сформированные блоки из МИ передаются в МШ, как показано на рисунке 2.

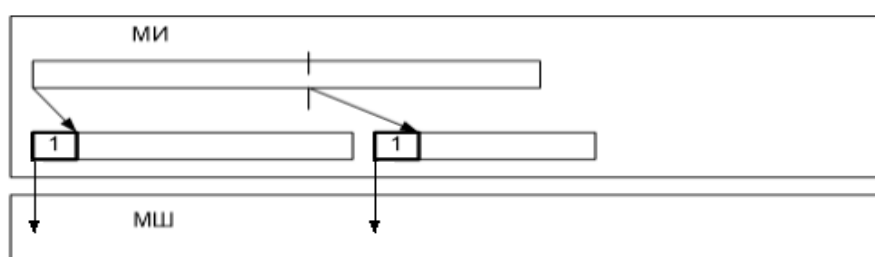


Рисунок 2 — Работа интерфейсного модуля

2.2 Разработка алгоритма работы модулей шифрования

2.2.1 Общий принцип работы

МШ полностью зашифровывает принятые блоки данных из МИ и ММАР и добавляет к ним заголовок №2 или №4, в котором хранится информация, необходимая для их расшифрования. Формат заголовка №2 представлен в таблице 1.

Таблица 1 — Формат заголовка №2

№ п/п	Назначение полей преамбулы	Размер поля, бит
1	Синхропосылка	128
2	Криптономер отправителя	16
3	Номер ключа	16
4	Время	16
5	Тип пакета	8
6	Резерв	8
7	Имитовставка на пакет данных	32
8	CRC32	32
9	Всего	256

Формат заголовка №4 представлен в таблице 2.

Таблица 2 — Формат заголовка №4

№ п/п	Назначение полей преамбулы	Размер поля, бит
1	Синхропосылка	64
2	Криптономер отправителя	16
3	Номер ключа	16
4	Время	16
5	Тип пакета	8
6	Резерв	104
7	CRC32	32
8	Всего	256

2.2.2 Реализация шифрования

Для реализации шифрования применяются алгоритмы блочных шифров с длинами блоков 64 («Магма») и 128 («Кузнечик») бит в соответствии с ГОСТ Р 34.12-2015, в режиме гаммирования, с выработкой имитовставки в соответствии с ГОСТ Р 34.13-2015.

Режим гаммирования имеет два параметра:

— s – целочисленная величина, определяющая длину блока открытого текста. Для алгоритма «Кузнечик» параметр s не может превосходить значение 128, а для алгоритма «Магма» – 64;

— IV – уникальная несекретная двоичная последовательность (инициализационный вектор, синхропосылка), определяемая для каждого открытого текста и ключа. Для алгоритма «Кузнечик» размер синхропосылки IV равен 64 битам, а для алгоритма «Магма» размер синхропосылки IV равен 32 битам.

Режим заключается в разбиении открытого текста на блоки длины s и в итеративном сложении полученных блоков открытого текста с усеченным до длины s результатом зашифрования блока, состоящего из синхропосылки IV и очередного значения счетчика. Процедура зашифрования в режиме гаммирования представлена на рисунке 3.

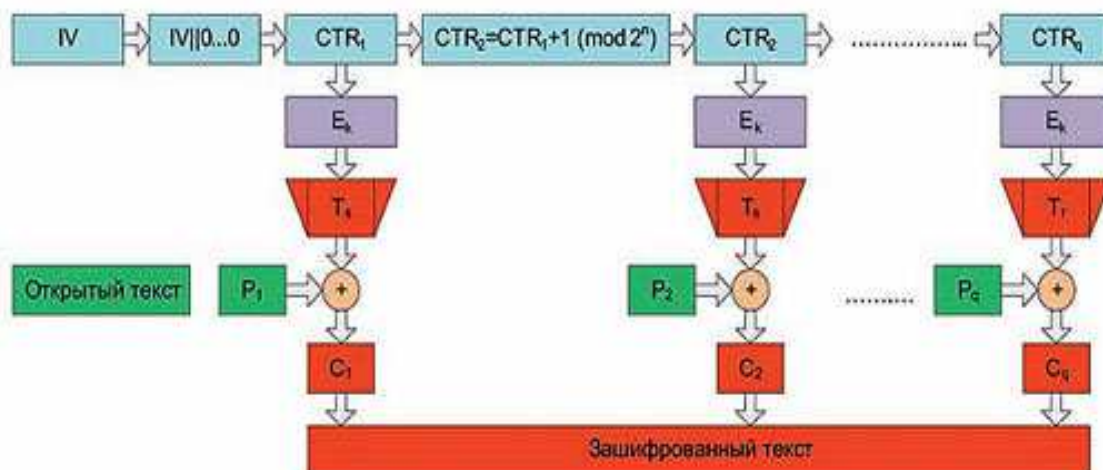


Рисунок 3 — Зашифрование в режиме гаммирования

Процедура расшифрования в режиме гаммирования заключается в прибавлении той же последовательности, только уже к зашифрованному тексту. Данная процедура представлена на рисунке 4.

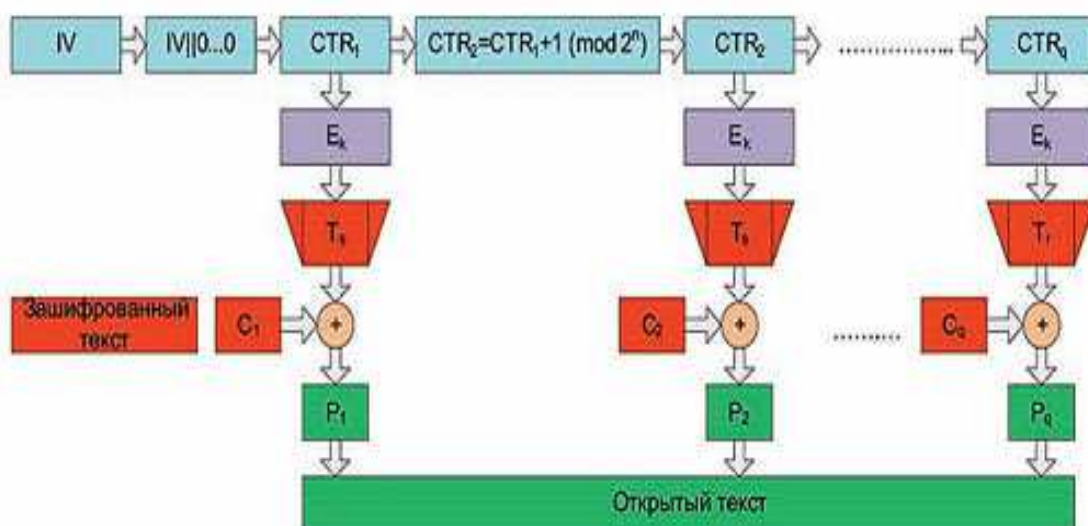


Рисунок 4 — Расшифрование в режиме гаммирования

У режима гаммирования нет недостатка режима простой замены (в режиме простой замены в зашифрованном тексте могут встречаться повторяющиеся блоки, свидетельствующие об одинаковых блоках открытого текста, что очевидным образом раскрывает его структуру). Режим гаммирования также допускает распараллеливание, что существенно повышает его производительность. Например, первый поток осуществляет преобразование «CRT1, CRT2, ..., CRT_r», с помощью которых осуществляется шифрование блоков открытого текста – «P1, P2, ..., P_r», второй поток осуществляет преобразование «CRT_{r+1}, CRT_{r+2}, ..., CRT_{2r}», с помощью которых осуществляется шифрование блоков открытого текста – «P_{r+1}, P_{r+2}, ..., P_{2r}», и т.д. Зачастую в режиме гаммирования параметр s выбирают равным длине блока алгоритма блочного шифрования, то есть для алгоритма «Кузнечик» s равен 128, а для алгоритма «Магма» – s равен 64, в этом случае нет необходимости производить усечение. Также стоит отметить, что синхропосылка, используемая в режиме гаммирования, может быть передана в открытом виде получателю информации, т.е. на нее не налагаются требования по обеспечению конфиденциальности. Стоит отметить, что в режиме гаммирования при зашифровании и расшифровании используется только прямое криптографическое преобразование E_k , в соответствии с ГОСТ Р 34.12-2015.

Процедура вычисления значения имитовставки похожа на процедуру зашифрования в режиме простой замены с зацеплением при m равным n и инициализации начального заполнения регистра сдвига значением 0_n : на вход алгоритма шифрования подается результат покомпонентного сложения очередного блока текста и результата зашифрования на предыдущем шаге. Основное отличие заключается в процедуре обработки последнего блока: на вход алгоритма блочного шифрования подается результат покомпонентного сложения последнего блока, результата зашифрования на предыдущем шаге

и одного из производных ключей. Конкретный производный ключ выбирается в зависимости от того, является ли последний блок исходного сообщения полным или нет.

Процедура выработки имитовставки, представлена на рисунке 5.

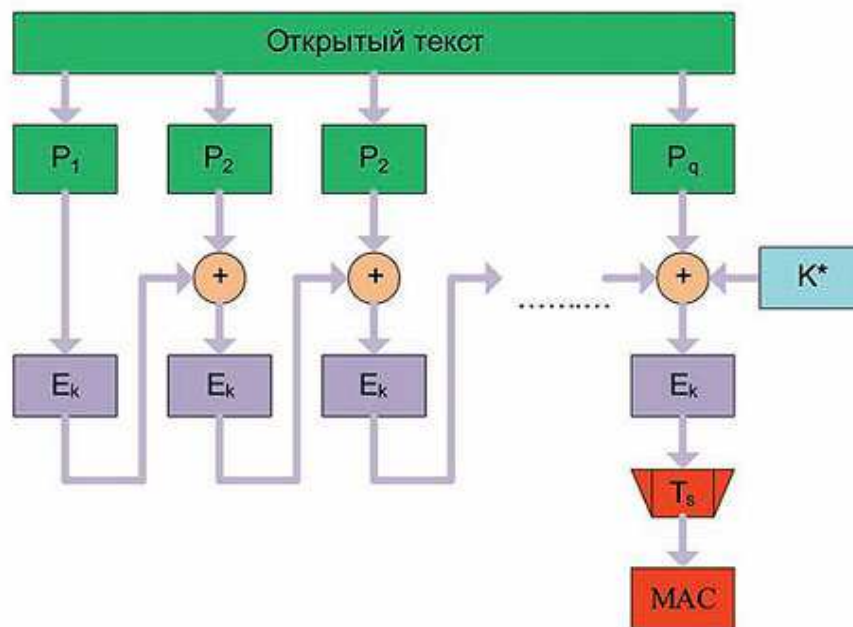


Рисунок 5 — Процедура выработки имитовставки

Значением имитовставки является результат применения процедуры усечения к выходу алгоритма шифрования при обработке последнего блока.

3 Разработка алгоритма взаимодействия модулей программного обеспечения

3.1 Алгоритм взаимодействия модулей программного обеспечения

МИ разбивает исходные данные на блоки, кратные 16 байтам. Максимальный размер блока 1536 байт. К каждому блоку добавляется заголовок №1, где указывается номер блока, размер блока и общее количество блоков исходных данных. В соответствии с протоколом взаимодействия модулей МИ и МШ сформированные блоки из МИ передаются в МШ.

МШ «Кузнечик» полностью зашифровывает принятые блоки данных и добавляет к ним заголовок №2, в котором хранится информация, необходимая для их расшифрования. В соответствии с протоколом взаимодействия модулей МШ и ММАР, сформированные блоки из МШ передаются в ММАР. Для реализации шифрования применяется блочный шифр «Кузнечик» в режиме гаммирования, с выработкой имитовставки в соответствии с ГОСТ Р 34.13-2015.

ММАР разбивает каждый принятый блок на подблоки по 96 байт. К каждому полученному подблоку добавляет заголовок №3, где указывается номер подблока, размер подблока и общее количество подблоков принятого блока. В соответствии с протоколом взаимодействия модулей ММАР и МСКЗИ сформированные подблоки из ММАР передаются в МШ «Кузнечик».

МШ «Магма» полностью зашифровывает принятые подблоки данных и добавляет к ним заголовок №4, в котором хранится информация, необходимая для их расшифрования. В соответствии с протоколом взаимодействия модулей МШ и ПП сформированные подблоки из МШ передаются в ПП. Для реализации шифрования применяется блочный шифр «Магма» в режиме гаммирования, в соответствии с ГОСТ Р 34.13-2015.

Модуль ПП предназначен для ретрансляции данных и формирование ошибок при передаче по каналу связи.

Взаимодействие модулей на передачу представлено на рисунке 6.

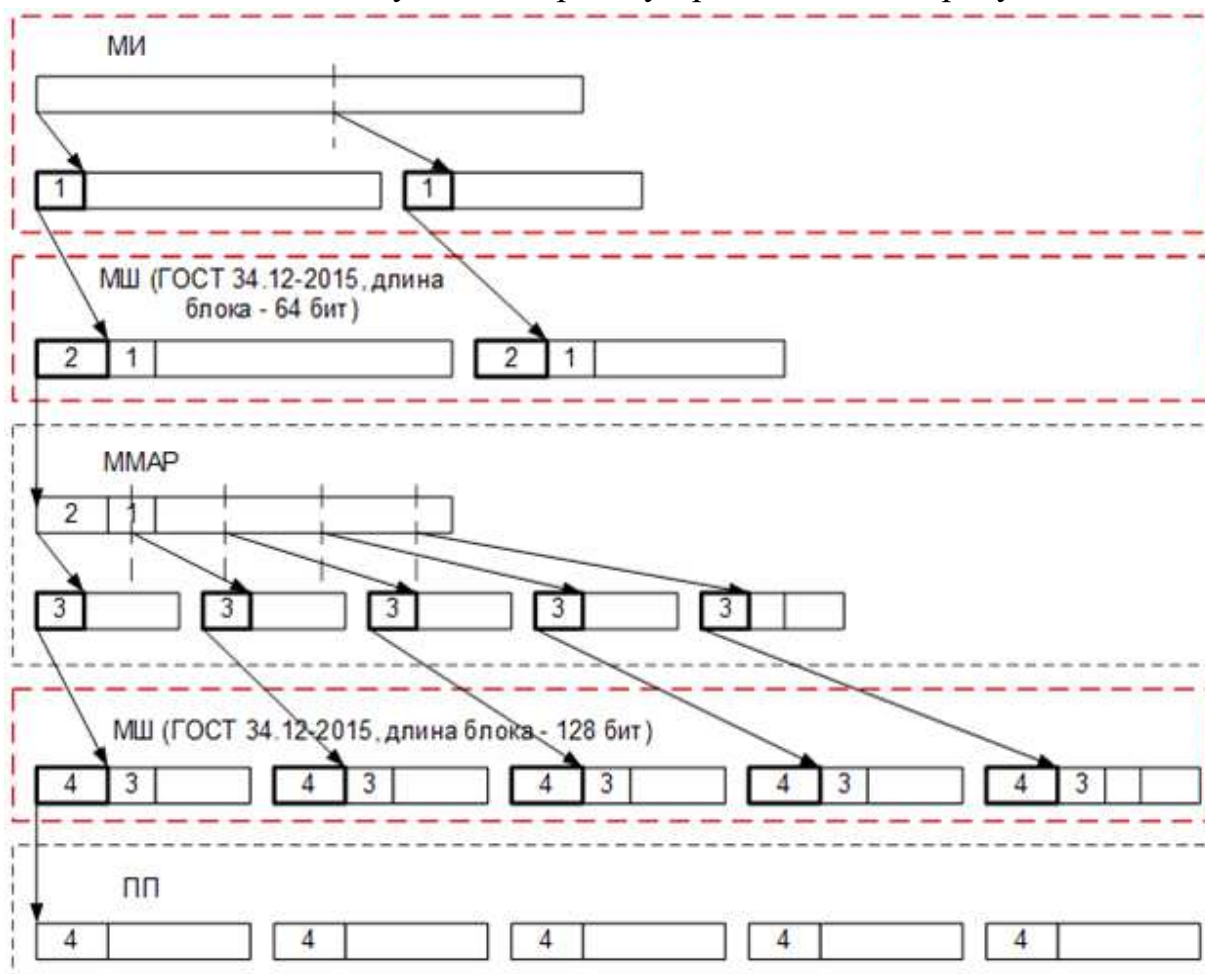


Рисунок 6 - Взаимодействие модулей

Приём аналогичен передаче в обратном направлении.

Алгоритм взаимодействия МИ-МШ представлен в приложении А.

Алгоритм взаимодействия МШ-МИ представлен в приложении Б.

Алгоритм взаимодействия МШ-ММАР представлен в приложении В.

3.2 Идентификация угроз безопасности информации, передаваемой при взаимодействии модулей

Под угрозой безопасности обрабатываемой информации понимается возможность возникновения опасного события, обусловленного совокупностью факторов и условий.

Опасными событиями считаются события, приводящие к нарушению заданных для объекта защиты характеристик безопасности

(конфиденциальности, доступности, целостности и т.д.).

Под условиями существования угрозы безопасности обрабатываемой информации понимаются определенные обстоятельства, свойства или особенности функционирования программного обеспечения устройства криптографической защиты информации, обеспечивающие возможность реализации угрозы.

Были идентифицированы информационные активы (ИА), подлежащие защите от угроз безопасности:

- ИА1 — абонентская информация;
- ИА2 — заголовочная информация сообщений;
- ИА3 — аутентификационная информация пользователей.

Были идентифицированы угрозы (У) информационной безопасности устройства криптографической защиты информации:

- У1 — несанкционированное ознакомление нарушителя с абонентской информацией, содержащей охраняемые сведения, с аутентифицирующей и ключевой информацией;

- У2 — целенаправленное навязывание ложных сообщений с абонентской информацией по каналу связи. К ложным сообщениям будем относить:

- сообщения, которые содержат ложную информацию в информационных частях сообщений (полях данных);

- сообщения, содержащие ложную информацию об отправителе или получателе в адресной части сообщения;

- задержанные и повторенные сообщения (составные части сообщений).

- У4 — переадресация абонентской информации (попадание информации одного абонента к другому абоненту, которому она не предназначена) в результате сбоя или преднамеренных действий нарушителя,

направленных на искажение полей адресов в абонентских сообщениях или маршрутноадресной информации.

Таким образом, для защищаемой информации необходимо обеспечить следующие характеристики безопасности:

- конфиденциальность. Данная характеристика должна быть обеспечена в отношении следующих объектов защиты:

- абонентских данных;
- полей адресных данных в абонентских сообщениях;
- данных аутентификации пользователей и технических средств;

- целостность (защищенность от несанкционированной модификации). Данная характеристика должна быть обеспечена в отношении следующих объектов защиты:

- абонентских данных;
- заголовков в абонентских сообщениях;
- доступность (защищенность от блокирования штатного доступа к системным ресурсам устройства криптографической защиты информации). Данная характеристика должна быть обеспечена в отношении абонентских данных, обрабатываемых в технических средствах.

Существование любой из приведенных выше основных угроз У1 - У4 может быть обусловлено, в общем случае, различными факторами, так, например, навязывание ложных сообщений возможно как из-за наличия недекларированных возможностей.

3.3 Выбор защитных мер

Были идентифицированы следующие защитные меры:

- квитирование с трехкратным повторением. Для противодействия утрате передаваемых в рамках протокола сообщений необходимо использовать

квитирование. При использовании квитирования правильность передачи каждого пакета должна подтверждаться квитанцией получателя. Таким образом отправитель будет оповещён о принятии сообщения получателем, в случае, если ответной квитанции от получателя нет — прервёт сеанс связи.

Для того, чтобы можно было организовать повторную передачу искажённых пакетов отправитель нумерует их. Для каждого пакета отправитель ожидает от приёмника так называемую положительную квитанцию — служебное сообщение, извещающее о том, что исходный пакет был получен и данные в нём оказались корректными. Время этого ожидания ограничено — при отправке каждого пакета передатчик запускает таймер, и если по его истечению положительная квитанция не получена, то пакет считается утерянным. В некоторых протоколах приёмник, в случае получения пакета с искаженными данными должен отправить отрицательную квитанцию — явное указание того, что данный кадр нужно передать повторно.

— имитозащита. Для противодействия модификации передаваемых в рамках протокола сообщений необходимо обеспечивать их имитозащиту. Получатель, таким образом, сможет точно определить, были ли внесены в полученное им сообщение какие-либо изменения. Имитозащита — это понятие, связанное с защитой от навязывания ложных сообщений путём формирования в зависимости от секретного ключа специальной дополнительной информации, называемой ИЗВ, которая передаётся вместе сообщением. Для вычисления имитовставки используется алгоритм, задающий зависимость имитовставки от каждого бита сообщения. Чем больше длина имитовставки, тем меньше вероятность того, что искажение шифртекста не будет обнаружено санкционированным (законным) получателем. Нарушитель может модифицировать шифртекст, но поскольку он не знает ключа, то не может вычислить новое значение имитовставки, соответствующее модифицированному сообщению. Вычисление ИЗВ передаваемых в рамках протокола сообщений реализуется в соответствии с ГОСТ 34.13-2015;

— контрольная сумма. Нахождение контрольной суммы, предназначенной для проверки целостности данных. Информационные (исполняемые) байты сообщения дополняются контрольной суммой. В рамках протокола реализуется алгоритм CRC16 IRDA при взаимодействии модулей и CRC32 для заголовков модулей шифрования.

4 Разработка протокола взаимодействия интерфейсного модуля и модуля шифрования

4.1 Общие положения

Протокол устанавливает информационно-логическое взаимодействие между МИ и МШ.

Для взаимодействия модулей устанавливается сетевое соединение (UDP СОКЕТ), в котором передаются команды от МИ в МШ по порту 4011, команды от МШ в МИ по порту 4111.

Между МИ в МШ передаются данные в пакетном режиме.

4.2 Форматы кадров при обмене данными, командами, сообщениями и квитанциями

4.2.1 Общий формат

Обмен информацией между интерфейсным модулем и МШ осуществляется с помощью четырех типов кадров:

- кадр 1 «Команды от МИ в МШ»;
- кадр 2 «Команды от МШ в МИ»;
- кадр 3 «Квитанции от МШ в МИ»;
- кадр 4 «Квитанции от МИ в МШ».

Каждый Кадр 1, передаваемый от МИ в МШ, квитируется Кадром 3, передаваемым от МШ в МИ. Каждый Кадр 2, передаваемый от МШ в МИ, квитируется Кадром 4, передаваемым от МИ в МШ. Устройство, передающее команду, далее называется передатчиком. Устройство, принимающее команду, далее называется приемником.

Приемник всегда подтверждает факт приема кадра передачей квитанции. Передача следующего кадра до получения квитанции запрещена. При отсутствии квитанции на кадр в течение 1 секунды или приеме отрицательной квитанции выполняется его повторная передача до двух раз. В случае

отсутствия квитанции после трёхкратной передачи приемник считается неисправным.

Все типы кадров имеют одинаковую структуру. Информационные (исполняемые) байты кадров обрамляются специальными флагами и дополняются контрольной суммой, как показано в таблице 3.

Таблица 3 — Структура кадра

Начало кадра (1 байт)	Информационные (исполняемые) байты (n байт)	Контрольная сумма (2 байта)	Конец кадра (1 байт)
0xC0	См. таблицу 3	CRC16	0xC1

Контрольная сумма рассчитывается на «Информационные (исполняемые) байты».

Если рассчитанная контрольная сумма не соответствует принятой, то приемник выдает квитанцию об ошибочном значении контрольной суммы.

Если хотя бы один байт (кроме флагов) не соответствует протоколу, то приемник выдает квитанцию об ошибочном кадре.

Передачик производит анализ информационных (исполняемых) байтов кадра и полей контрольной суммы на наличие специальных символов «0xC0», «0xC1» и «0x7D». При обнаружении в указанных полях кода специального символа, осуществляется вставка двух байт, то есть вставка специального символа «0x7D» и результата сложения по модулю 2 заменяемого символа и кода «0x20».

Приёмник информации производит анализ байтов и поля контрольной суммы на наличие специального символа «0x7D». При обнаружении в указанных полях специального символа, он заменяется результатом сложения по модулю 2 следующего символа и кода «0x20».

4.2.2 Структура и состав информационных (исполняемых) байт кадров

Структура и состав информационных (исполняемых) байт кадров представлено в таблице 4.

Таблица 4 — Структура информационных (исполняемых) байт кадров

Информационные (исполняемые) байты				
Признак источника и получателя информации (1 байт)	Тип кадра (1 байт)	Наименование кадра (1 байт)	Резерв (1 байт)	Данные (0 ÷ n байт)

Поле «Признак источника и получателя информации» характеризует отправителя и получателя информации – участников информационно-логического взаимодействия.

Например, для участников данного протокола информационно-логического взаимодействия (МИ и МШ) код поля «Признак источника и получателя информации» равен «0x11».

Другие возможные значения поля «Признак источника и получателя информации», которые описаны в рамках других протоколов информационно-логического взаимодействия:

- «0x22» - обмен между МШ и ММАР;
- «0x33» - обмен между ММАР и МСКЗИ;
- «0x44» - обмен между МСКЗИ и ПП.

Поле «Тип кадра» определяет один из четырех типов кадров обмена:

- «0x00» - кадр 1 «Команды от МИ в МШ»;
- «0x01» - кадр 2 «Команды от МШ в МИ»;
- «0x02» - кадр 3 «Квитанции от МШ в МИ»;
- «0x03» - кадр 4 «Квитанции от МИ в МШ».

Поле «Наименование кадра» определяет назначение команды, сообщения или квитанции.

Поле «Резерв» в данном протоколе не используется.

Поле «Данные» собственно определяет параметры данных, команд, сообщений и квитанций.

4.3 Команды от интерфейсного модуля в модуль шифрования

Общий формат кадра команд (кадр 1) представлен в таблице 5.

Таблица 5 — Кадр 1

Начало кадра (1 байт)	Признак источника и получателя информации (1 байт)	Тип кадра (1 байт)	Наименование кадра (1 байт)	Резерв (1 байт)	Данные (0 ÷ n байт)	Контрольная сумма (2 байта)	Конец кадра (1 байт)
0xC0	0x11	0x00	См. таблицу 6	0x00	См. таблицу 6	CRC16	0xC1

Наименование команд от МИ в МШ и соответствующие им коды полей «Наименование кадра» и «Данные» приведены в таблице 6.

Таблица 6 — Наименование команд от МИ в МШ

№ п/п	Назначение кадра	Наименование кадра	Размер данных, байт	Примечание
1	Начало работы с МШ	0x01	0	
2	Передать пакет данных для шифрования	0x03	20-1540	D0, D1 - номер пакета D2, D3 - размер D4-D1539 - данные
3	Завершение работы с МШ	0x09	0	

4.4 Команды от модуля шифрования в интерфейсный модуль

Общий формат кадра команд (кадр 2) представлен в таблице 7.

Таблица 7 — Кадр 2

Начало кадра (1 байт)	Признак источника и получателя информации (1 байт)	Тип кадра (1 байт)	Наименование кадра (1 байт)	Резерв (1 байт)	Данные (0 ÷ n байт)	Контрольная сумма CRC16 (2 байта)	Конец кадра (1 байт)
0xC0	0x11	0x01	См. таблицу 8	0x00	См. таблицу 8	CRC16	0xC1

Наименование команд от МИ в МШ и соответствующие им коды полей «Наименование кадра» и «Данные» приведены в таблице 8.

Таблица 8 — Наименование команд от МШ в МИ

№ п/п	Назначение кадра	Наименование кадра	Размер данных, байт	Примечание
1	Начало работы с МИ	0x01	0	Ответ на команду "Начало работы с МШ"

Продолжение таблицы 8

№ п/п	Назначение кадра	Наименование кадра	Размер данных, байт	Примечание
2	Передан пакет данных для шифрования	0x03	3	D0, D1 - номер пакета D2 - результат 1 – успешно; 0 - не успешно
3	Принят пакет данных	0x06	20-1540	D0, D1 - номер пакета D2, D3 - размер D4-D1539 - данные
4	Завершение работы с МШ	0x09	0	Ответ на команду Завершение работы с МШ

4.5 Квитанции от модуля шифрования в интерфейсный модуль

Общий формат кадра квитанций (кадр 3), представлен в таблице 9.

Таблица 9 — Кадр 3

Начало кадра (1 байт)	Признак источника и получателя информации (1 байт)	Тип кадра (1 байт)	Наименование кадра (1 байт)	Результат выполнения команды (1 байт)	Контрольная сумма (2 байта)	Конец кадра (1 байт)
0xC0	0x11	0x02	См. код поля «Наименование кадра» в таблице 6	«0x00» - кадр принят; «0x01» - ошибочное значение CRC для принятого кадра; «0x02» - неопознанная команда; «0x03» - не готов к приему; «0x04» *- готов к приему.	CRC16	0xC1

Квитанция с кодом «0x04» в поле «Результат выполнения команды» выдается МШ самостоятельно при освобождении буфера приема.

4.6 Квитанции от интерфейсного модуля в модуль шифрования

Общий формат кадра квитанций (кадр 4), представлен в таблице 10.

Таблица 10 — Кадр 4

Начало кадра (1 байт)	Признак источника и получателя информации (1 байт)	Тип кадра (1 байт)	Наименование кадра (1 байт)	Результат выполнения команды (1 байт)	Контрольная сумма CRC16 (2 байта)	Конец кадра (1 байт)
0xC0	0x11	0x03	См. код поля «Наименование кадра» в таблице 8	«0x00» - кадр принят; «0x01» - ошибочное значение CRC для принятого кадра; «0x02» - неопознанная команда;	CRC16	0xC1

5 Разработка программного обеспечения устройства криптографической защиты информации

5.1 Общие сведения

Исходный текст ПО написан на языке программирования C++.

Программа устанавливается на ПЭВМ с ОС Windows 7.

Программное обеспечение состоит из пяти модулей. Перед началом работы необходимо запустить все пять модулей и произвести их настройку.

5.2 Описание модулей, входящих в состав программного обеспечения

5.2.1 Описание программного обеспечения модуля шифрования «Кузнечик»

Исходный текст ПО МШ «Кузнечик» представлен в приложении Г.

ПО МШ «Кузнечик» обеспечивает:

- графический интерфейс пользователя;
- взаимодействие с модулями МИ и ММАР по протоколу информационно-логического взаимодействия;
- шифрование и расшифрование пакетов данных по алгоритму блочного шифрования «Кузнечик».

ПО модуля состоит из следующих частей:

- программное обеспечение, реализующее графический интерфейс пользователя;
- программные модули, реализующие алгоритмы работы функциональных узлов (интерфейс с МИ и ММАР, шифратор).

Программа состоит из программных модулей, представленных в таблице 11.

Таблица 11 — Номенклатура программных модулей имитационной модели МШ «Кузнечик»

Наименование модуля	Описание модуля
crc32.cpp	Реализует функции вычисления контрольной суммы по алгоритму CRC-32.
CRC_IRDA.cpp	Реализует функции вычисления контрольной суммы по алгоритму CRC-16.
dte.cpp	Реализует интерфейс с другими модулями программного обеспечения.
fifo.cpp	Реализует очередь данных типа FIFO.
log.cpp	Реализует ведение отладочного журнала.
main.cpp	Реализует графический интерфейс пользователя и функции верхнего уровня интерфейсов с модулями МИ и ММАР.
MSH.cpp	Реализует начальную инициализацию программы при запуске.
gostgrasshopper.cpp	Реализует основные функции шифрования по алгоритму «Кузнечик»
params.cpp	Реализует ввод дополнительных параметров работы программы.
UnitControl.cpp	Реализует отображение результатов расчета контрольного примера.

В модуле `crc32.cpp` реализованы функции, представленные в таблице 12.

Таблица 12 — Функции, реализованные в `crc32.cpp`

Наименование функции	Описание функции
<code>crc32_update_byte</code>	Осуществляет обработку очередного байта входных данных.
<code>crc32_update</code>	Осуществляет обработку массива входных данных.

Продолжение таблицы 12

Наименование функции	Описание функции
calc_csum	Возвращаемое значение контрольной суммы.
crc32_final	Возвращает инвертированное значение контрольной суммы.

В модуле CRC_IRDA.cpp реализованы функции, представленные в таблице 13.

Таблица 13 - Функции, реализованные в CRC_IRDA.cpp

Наименование функции	Описание функции
pprfcs	Осуществляет обработку массива входных данных.
crc_make	Возвращает инвертированное значение контрольной суммы.
crc_check	Осуществляет сравнение рассчитанной контрольной суммы с эталонным значением.

В модуле dte.cpp реализованы методы классов CDte, CDteMenager, представленные в таблице 14.

Таблица 14 - Функции, реализованные в dte.cpp

Наименование функции	Описание функции
CDte::Init	Осуществляет инициализацию протокола взаимодействия.
CDte::TimerStart	Запускает таймер ожидания квитанции.
CDte::TimerStop	Останавливает таймер ожидания квитанции.
CDte::TimerAction	Событие срабатывания таймера.
CDte::SendDte	Помещает данные в очередь передачи.
CDte::SendAck	Формирует и передает квитанцию.
CDte::insert	Байтстаффинг одного байта данных.

Продолжение таблицы 14

Наименование функции	Описание функции
CDte::Encode	Байтстаффинг пакета данных.
CDte::Send	Подготовка пакета к передаче.
CDte::SendData	Передача пакета.
CDte::AckProcess	Обработка принятых квитанций.
CDte::CheckFormat	Проверка формата принятого пакета.
CDte::Receive	Прием и обработка пакета в соответствии с протоколом взаимодействия.
CDte::Idle	Работа модуля в фоновом режиме.
CDte::isBusy	Возвращает признак занятости модуля.
CDteMenager::Init	Инициализация менеджера модулей интерфейсов.
CDteMenager::Add	Добавление интерфейса в менеджер.
CDteMenager::Delete	Удаление интерфейса из менеджера.
CDteMenager::Idle	Фоновая работа менеджера протоколов.
CDteMenager::Timer Action	Обработка события таймера.
CDteMenager::isBusy	Возвращает признак занятости интерфейсов.

В модуле `fifo.cpp` реализованы методы классов `CFifo`, `CFifoMenager`, представленные в таблице 15.

Таблица 15 - Функции, реализованные в `fifo.cpp`

Наименование функции	Описание функции
CFifo::Init	Инициализация очередей типа FIFO.
CFifo::Write	Запись данных в очередь
CFifo::Read	Чтение данных из очереди.
CFifo::ReadEmpty	Сброс данных без вычитывания.
CFifo::GetReadPtr	Возвращает указатель чтения из очереди.
CFifo::Count	Возвращает количество занятых элементов очереди.
CFifo::Free	Возвращает количество свободных элементов очереди.

Продолжение таблицы 15

Наименование функции	Описание функции
CFifo::Clear	Очистка очереди.
CFifo::Name	Возвращает символьное имя очереди.
CfifoMenager:: Init	Инициализация менеджера очередей.
CfifoMenager:: Find	Возвращает индекс очереди по символьному имени.
CfifoMenager:: Delete	Удаляет очередь из менеджера.
CfifoMenager:: Add	Добавляет очередь в менеджер.

В модуле log.cpp реализованы методы класса CLog, представленные в таблице 16.

Таблица 16 - Функции, реализованные в log.cpp

Наименование функции	Описание функции
CLog::Init	Инициализация кольцевого буфера отладочной информации.
CLog::Put	Помещение массива в кольцевой буфер.

В модуле main.cpp реализованы методы класса TMainForm, представленные в таблице 17.

Таблица 17 - Функции, реализованные в main.cpp

Наименование функции	Описание функции
TmainForm:: getMKIInCmdLog	Возвращает форматированную строку журнала команд в МИ
TmainForm:: getMKIOutCmdLog	Возвращает форматированную строку журнала команд от МИ
TmainForm:: getMMARInCmdLog	Возвращает форматированную строку журнала команд в ММАР
TmainForm:: getMMAROutCmdLog	Возвращает форматированную строку журнала команд от ММАР

Продолжение таблицы 17

Наименование функции	Описание функции
TMainForm::FormCreate	Обработчик события создания главного окна программы.
TMainForm::SendToMMAR	Передача пакета в ММАР через UDP-сокеты
TMainForm::SendToMKI	Передача пакета в МИ через UDP-сокеты
TMainForm::InitSockets	Инициализация UDP-сокетов
TMainForm::TimerTimer	Обработчик события таймер
TMainForm::UDPServerMKIU DPRead	Чтение данных из UDP-сокета МИ, запись в очередь.
TMainForm::CreateHeaderMKI	Инициализация заголовка пакетов в МИ
TMainForm::PipeToMKI	Чтение пакетов данных и квитанций из очереди МИ и передача в модуль канального протокола.
TMainForm::PutPipeToMKI	Запись подготовленного пакета в очередь данных в МИ
TMainForm::CreatePipeMKI	Создание очереди МИ и помещение ее в пул очередей.
TMainForm::CreateHeaderMMAR	Инициализация заголовка пакетов в ММАР
TMainForm::PipeToMMAR	Чтение пакетов данных и квитанций из очереди ММАР и передача в модуль канального протокола.
TMainForm::PutPipeToMMAR	Запись подготовленного пакета в очередь данных в ММАР
TMainForm::CreatePipeMMAR	Создание общей очереди ММАР и помещение ее в пул очередей.
TmainForm::getDataLog	Возвращает форматированную строку журнала данных.

Продолжение таблицы 17

Наименование функции	Описание функции
TmainForm::PipeTxDataToMMAR	Чтение данных из очереди ММАР и передача в модуль канального протокола.
TMainForm::CreatePipeTxData	Создание очереди данных ММАР и помещение ее в пул очередей данных.
TMainForm::CmdAction	Обработка принятых пакетов данных, команд и квитанций.
TmainForm::MKIIdle	Фоновая работа интерфейса с МИ.
TMainForm::MMARIdle	Фоновая работа интерфейса с ММАР.
TMainForm::WaitTimerTimer	Ожидание инициализации таймера.
TMainForm::N3Click	Обработка контекстного меню очистки окна журнала.
TMainForm::UDPServerMMAR UDPRead	Чтение данных из UDP-сокета ММАР, запись в очередь.
TMainForm::Button1Click	Задание ip-адреса модуля МИ.
TMainForm::SendUDPClick	Задание ip-адреса модуля ММАР.
TmainForm::ParamsButtonClick	Вывод на экран окна задания параметры режима работы.

В MSH.cpp модуле реализованы функции, представленные в таблице 18.

Таблица 18 - Функции, реализованные в MSH.cpp

Наименование функции	Описание функции
WinMain	Начальная инициализация при запуске программы.

В модуле params.cpp реализованы функции, представленные в таблице 19.

Таблица 19 - Функции, реализованные в params.cpp

Наименование функции	Описание функции
TparamsForm:: OKButtonClick	Обработчик события нажатия кнопки применения параметров режима работы.

В модуле UnitControl.cpp реализованы функции, представленные в таблице 20.

Таблица 20 - Функции, реализованные в UnitControl.cpp

Наименование функции	Описание функции
TFormControl::FormActivate	Отображение результатов тестирования по контрольному примеру.

В модуле gostgrasshopper.cpp реализованы функции, представленные в таблице 21.

Таблица 21 - Функции, реализованные в gostgrasshopper.cpp

Наименование функции	Описание функции
funcX	Побитное сложение входного вектора с раундовым ключом.
funcS	Нелинейное преобразование S
funcR	Линейное преобразование R
funcL	Линейное преобразование L
funcLSX	Базовое преобразование
funcC	Выработка итерационных констант
ExpandKey	Выработка раундовых ключей
Encrypt_14	Базовый алгоритм зашифрования
CounterGost	Режим гаммирования
ShifttLeftOne	Сдвиг бита «на один»
ExpandImitKey	Выработка вспомогательных ключей
mac	Выработка имитовставки

5.2.2 Описание программного обеспечения модуля шифрования «Магма»

Исходный текст ПО МШ «Магма» представлен в приложении Д.

ПО модуля МШ «Магма» обеспечивает:

- графический интерфейс пользователя;
- взаимодействие с модулями ММАР и ПП по протоколу информационно-логического взаимодействия;
- шифрование и расшифрование пакетов данных по алгоритму блочного шифрования «Магма».

ПО модуля состоит из следующих частей:

- программное обеспечение, реализующее графический интерфейс пользователя;
- программные модули, реализующие алгоритмы работы функциональных узлов (интерфейс с ММАР и ПП, шифратор).

Программа состоит из программных модулей, представленных в таблице 22.

Таблица 22 – Номенклатура программных модулей МШ «Магма»

Наименование модуля	Описание модуля
crc32.cpp	Реализует функции вычисления контрольной суммы по алгоритму CRC-32
CRC_IRDA.cpp	Реализует функции вычисления контрольной суммы по алгоритму CRC-16
dte.cpp	Реализует интерфейс с другими модулями имитационной модели.
fifo.cpp	Реализует очередь данных типа FIFO.
log.cpp	Реализует ведение отладочного журнала.

Продолжение таблицы 22

Наименование модуля	Описание модуля
main.cpp	Реализует графический интерфейс пользователя и функции верхнего уровня интерфейсов с модулями МИ и ММАР.
MSKZI.cpp	Реализует начальную инициализацию программы при запуске.
magma.cpp	Реализует основные функции шифрования по алгоритму «Магма».

В модуле main.cpp реализованы методы класса TMainForm, представленные в таблице 23.

Таблица 23 — Функции, реализованные в main.cpp

Наименование функции	Описание функции
TMainForm::getMMARInCmdLog	Возвращает форматированную строку журнала команд в ММАР
TMainForm::getMMAROutCmdLog	Возвращает форматированную строку журнала команд от ММАР
TMainForm::getPPInCmdLog	Возвращает форматированную строку журнала команд в ПП
TMainForm::getPPOutCmdLog	Возвращает форматированную строку журнала команд от ПП
TMainForm::FormCreate	Обработчик события создания главного окна программы.
TMainForm::SendToMMAR	Передача пакета в ММАР через UDP-сокеты
TMainForm::SendToPP	Передача пакета в ПП через UDP-сокеты

Продолжение таблицы 23

Наименование функции	Описание функции
TMainForm::InitSockets	Инициализация UDP-сокетов
TMainForm::TimerTimer	Обработчик события таймер
TmainForm::UDPServer PPUDPRead	Чтение данных из UDP-сокета ПП, запись в очередь.
TMainForm::CreateHeaderPP	Инициализация заголовка пакетов в ПП
TMainForm::PipeToPP	Чтение пакетов данных и квитанций из очереди ПП и передача в модуль канального протокола.
TMainForm::PutPipeToPP	Запись подготовленного пакета в очередь данных в ПП
TMainForm::CreatePipePP	Создание очереди ПП и помещение ее в пул очередей.
TMainForm::CreateHeaderMMAR	Инициализация заголовка пакетов в ММАР
TMainForm::PipeToMMAR	Чтение пакетов данных и квитанций из очереди ММАР и передача в модуль канального протокола.
TMainForm::PutPipeToMMAR	Запись подготовленного пакета в очередь данных в ММАР
TMainForm::CreatePipeMMAR	Создание общей очереди ММАР и помещение ее в пул очередей.
TMainForm::getDataLog	Возвращает форматированную строку журнала данных.
TMainForm::PipeTxDataToPP	Чтение данных из очереди ПП и передача в модуль канального протокола.

Наименование функции	Описание функции
TMainForm::CreatePipeTxData	Создание очереди данных ПП и помещение ее в пул очередей данных.
TMainForm::CmdAction	Обработка принятых пакетов данных, команд и квитанций.
TMainForm::PPIdle	Фоновая работа интерфейса с ПП.
TMainForm::MMARIdle	Фоновая работа интерфейса с ММАР.
TMainForm::WaitTimerTimer	Ожидание инициализации таймера.
TMainForm::N3Click	Обработка контекстного меню очистки окна журнала.
TMainForm::UDPServerMMARUDPRead	Чтение данных из UDP-сокета ММАР, запись в очередь.
TMainForm::Button1Click	Задание ip-адреса модуля ММАР.

В модуле MSKZI.cpp реализованы функции, представленные в таблице 24.

Таблица 24 — Функции, реализованные в MSKZI.cpp

Наименование функции	Описание функции
WinMain	Начальная инициализация при запуске программы.

В модуле magma.cpp реализованы функции, представленные в таблице 25.

Таблица 25 — Функции, реализованные в magma.cpp

Наименование функции	Описание функции
kboxinit	Инициализация таблицы замен.
f	Поблочная замена в соответствии с таблицей замен. Циклический сдвиг влево.

Продолжение таблицы 25

Наименование функции	Описание функции
gostcrypt	Цикл зашифрования.
gostdecrypt	Цикл расшифрования
gostcnt	Режим гаммирования

5.2.3 Описание программного обеспечения интерфейсного модуля

Исходный текст ПО МИ представлен в приложении Е.

ПО модуля МИ обеспечивает:

- графический интерфейс пользователя;
- взаимодействие с модулем МШ по протоколу информационно-логического взаимодействия;
- передачу и приём файлов данных.

ПО модуля МИ состоит из следующих частей:

- программное обеспечение, реализующее графический интерфейс пользователя;
- программные модули, реализующие алгоритмы работы функциональных узлов (интерфейс с МШ «Кузнечик»).

Программа состоит из программных модулей, представленных в таблице 26.

Таблица 26 - Номенклатура программных модулей имитационной модели МИ

Наименование модуля	Описание модуля
CRC_IRDA.cpp	Реализует функции вычисления контрольной суммы по алгоритму CRC-16
dte.cpp	Реализует интерфейс с другими модулями имитационной модели.
fifo.cpp	Реализует очередь данных типа FIFO.
log.cpp	Реализует ведение отладочного журнала.

Продолжение таблицы 26

Наименование модуля	Описание модуля
main.cpp	Реализует графический интерфейс пользователя и функции верхнего уровня интерфейсов с модулем МШ.
MKI.cpp	Реализует начальную инициализацию программы при запуске.

В модуле main.cpp реализованы методы класса TMainForm, представленные в таблице 27.

Таблица 27 – Функции, реализованные в main.cpp

Наименование функции	Описание функции
TMainForm::getInCmdLog	Возвращает форматированную строку журнала команд в МШ.
TMainForm::getOutCmdLog	Возвращает форматированную строку журнала команд от МШ.
TMainForm::FormCreate	Обработчик события создания главного окна программы.
TMainForm::SendToMSH	Передача пакета в МШ через UDP-сокеты.
TMainForm::InitSockets	Инициализация UDP-сокетов.
TMainForm::TimerTimer	Обработчик события таймер.
TmainForm::UDPServerUDPRead	Чтение данных из UDP-сокета МШ, запись в очередь.
TMainForm::CreateHeader	Инициализация заголовка пакетов в МШ.
TMainForm::PipeTo	Чтение пакетов данных и квитанций из очереди МШ и передача в модуль канального протокола.

Наименование функции	Описание функции
TMainForm::PutPipeTo	Запись подготовленного пакета в очередь данных в МШ
TMainForm::CreatePipe	Создание очереди МШ и помещение ее в пул очередей.
TMainForm::CmdAction	Обработка принятых пакетов данных, команд и квитанций.
TMainForm::ReadFile	Чтение данных из файла и помещение их в очередь на обработку
TMainForm::ProcessTxFile	Обработка данных из очереди на передачу
TMainForm::SendFile	Помещение обработанных из файла данных в очередь на передачу
TMainForm::ProcessRxFile	Обработка принятых данных от МШ
TMainForm::SendButtonClick	Обработка нажатия кнопки "Передать в МШ"

В модуле MKI.cpp реализованы функции, представленные в таблице 28.

Таблица 28 - Функции, реализованные в MKI.cpp

Наименование функции	Описание функции
WinMain	Начальная инициализация при запуске программы.

6.1 Общие сведения

Программный комплекс состоит из пяти модулей, каждый из которых представлен в виде исполняемого файла. Модули взаимодействуют между собой через сетевой протокол UDP посредством информационно-логического протокола. Перед началом работы необходимо запустить все пять модулей и произвести их настройку. Взаимодействие модулей представлено на рисунке 7.

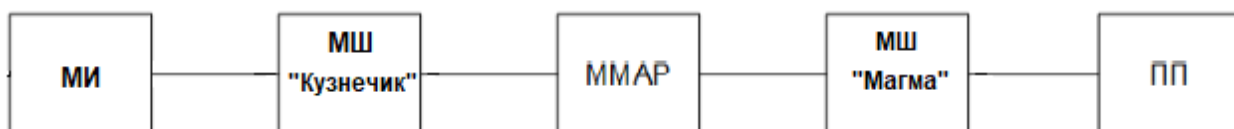


Рисунок 7— Взаимодействие модулей программного обеспечения

6.2 Интерфейсный модуль

Модуль МИ предназначен для передачи и приёма файлов. Графический интерфейс МИ представлен на рисунке 8.

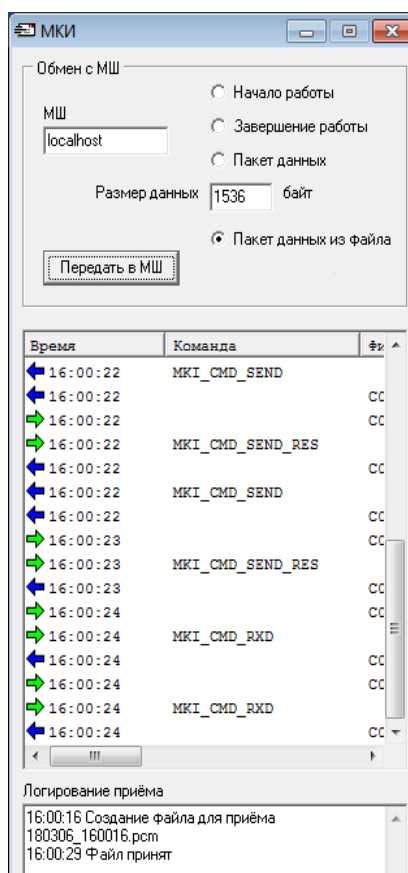


Рисунок 8 — Графический интерфейс модуля МИ

В поле МШ «Кузнечик» необходимо ввести IP адрес ПЭВМ на котором запущен модуль МШ «Кузнечик». В поле «Размер данных» необходимо ввести размер блока данных от 16 до 1536 кратно 16. Передаваемая информация в МШ «Кузнечик» будет разбита на фиксированные блоки.

Для начала работы, чтобы сбросить настройки модуля МШ «Кузнечик», в окне «Обмен данными с МШ Кузнечик» выберете команду «Начало работы» и нажмите кнопку «Передать в МШ Кузнечик». В МШ «Кузнечик» будет отправлена команда «Начало работы». Получив команду МШ «Кузнечик» сформирует и передаст ответ в МИ. Обмен команд фиксируется в окне логирования, с указанием времени и названием команды.

Для окончания работы выберете команду «Завершение работы» и нажмите кнопку «Передать в МШ Кузнечик». Получив команду МШ «Кузнечик» сформирует и передаст ответ в МИ.

Для передачи файла необходимо выбрать «Пакет данных из файла», нажать кнопку «Передать в МШ Кузнечик». После этого появится диалоговое окно для выбора файла, представленное на рисунке 9.

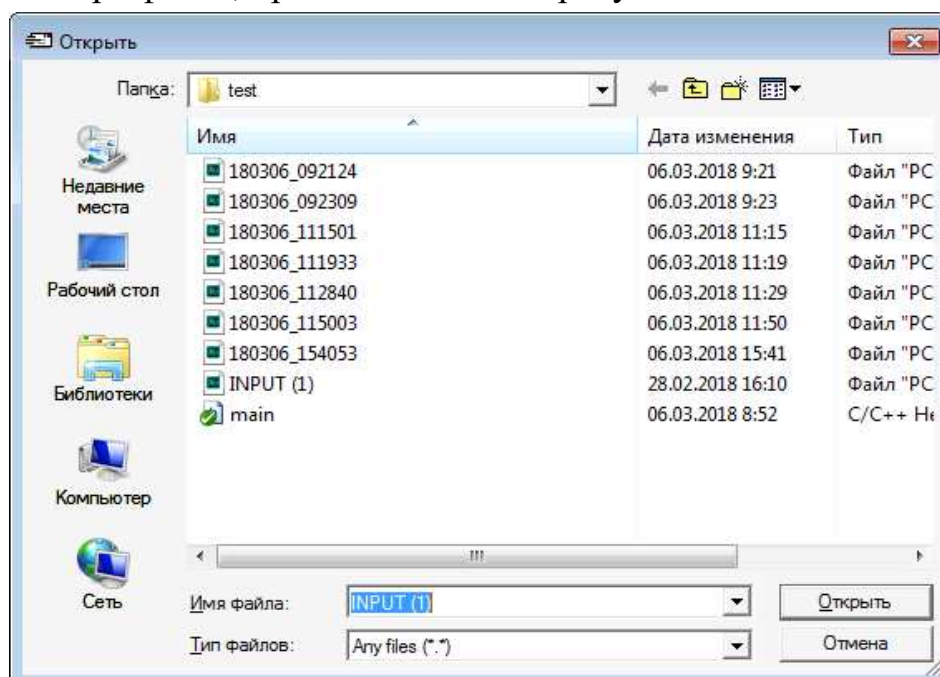


Рисунок 9 — Диалоговое окно для выбора файла модуля МИ

Выбрав файл, появится окно процесса передачи файла, представленного

на рисунке 10.

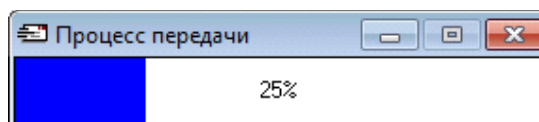


Рисунок 10 — Окно процесса передачи файла модуля МИ

Начало приёма фиксируется автоматически, автоматически определяется тип принимаемых данных и создаётся файл для приёма. Процесс приёма представлен на рисунке 11.

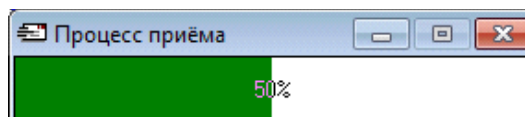


Рисунок 11 — Окно процесса приема файла модуля МШ «Кузнечик»

6.3 Модуль шифрования «Кузнечик»

Модуль МШ «Кузнечик» предназначен для шифрования и расшифрования блоков данных. Графический интерфейс МИ представлен на рисунке 12.

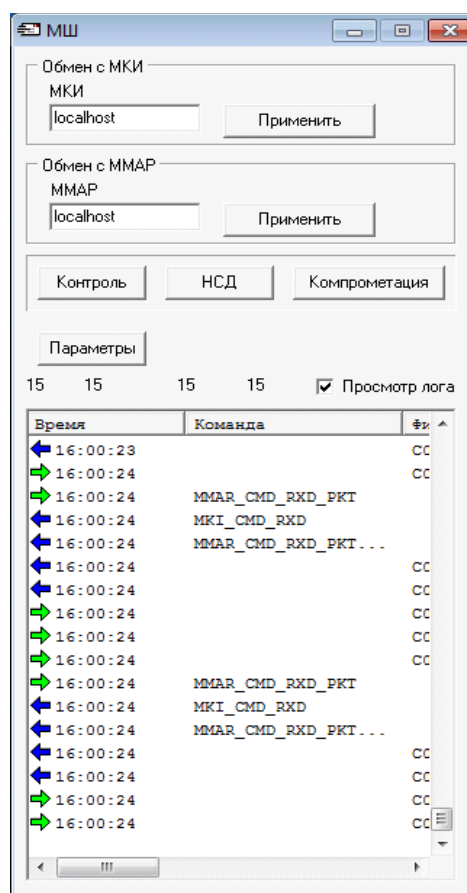


Рисунок 12 — Графический интерфейс модуля МШ «Кузнечик»

В поле «МИ» необходимо ввести IP адрес ПЭВМ на котором запущен модуль МИ и нажать кнопку «Применить». В поле «ММАР» необходимо ввести IP адрес ПЭВМ на котором запущен модуль ММАР и нажать кнопку «Применить».

Нажав кнопку «Контроль» появится окно «Контрольный пример», в котором показана информация о проверке алгоритма шифрования:

- проверка развёртывания ключа, сравнение контрольного и рассчитанного значения;
- проверка режима гаммирования, сравнение контрольного и рассчитанного значения;
- проверка выработки имитовставки, сравнение контрольного и рассчитанного значения.

Окно «Контрольный пример» представлено на рисунке 13.

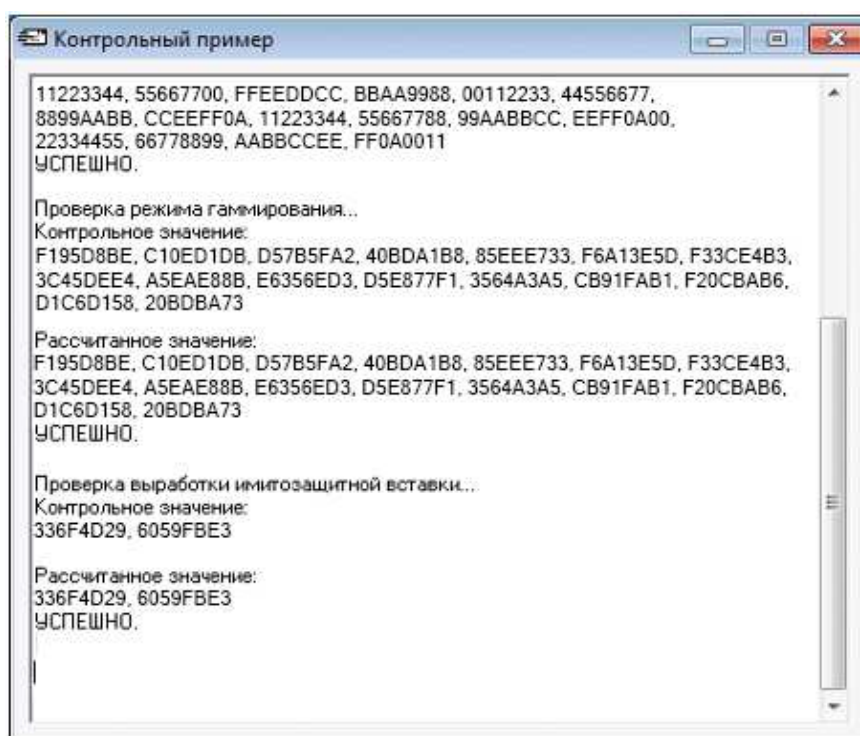


Рисунок 13 —Окно "Контрольный пример" модуля МШ «Кузнечик»

Нажав кнопку «НСД» или «Компрометация» выводится сообщение и блокируется работа модуля, как показано на рисунке 14.

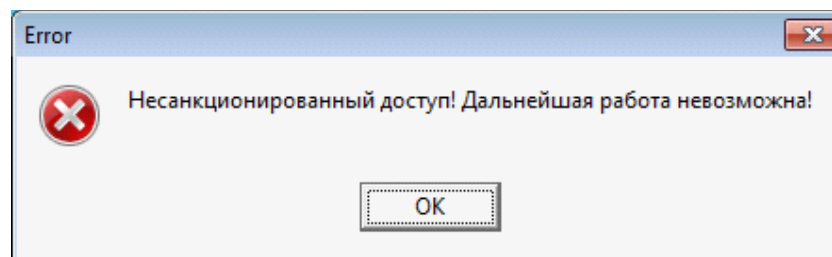


Рисунок 14 —Сообщение о блокировке работы модуля МШ «Кузнечик»

Для дальнейшей работы необходимо перезапустить все модули. Нажав кнопку «Параметры» появится окно, представленное на рисунке 15.

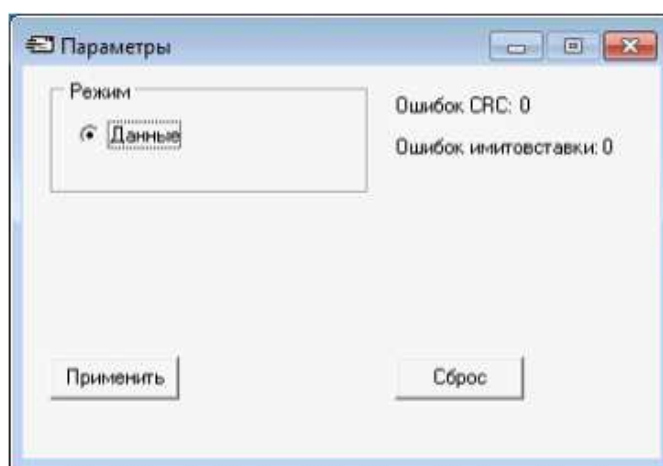


Рисунок 15 —Окно «Параметры» модуля МШ «Кузнечик»

В данной работе МШ «Кузнечик» имеет один режим работы «Данные», который представляет собой пакетный режим работы, в котором анализируется имитовставка в пакете на приёме. В случае неверной имитовставки расшифрованный пакет отбрасывается и не передаётся в МИ.

Также в окне «Параметры» ведётся статистика о принятых пакетах от ММАР, ошибках контрольной суммы заголовка, ошибках имитовставки. Статистику можно сбросить нажав кнопку «Сброс».

Обмен команд с МИ и ММАР фиксируется в окне логирования, с указанием времени и названием команды.

6.4 Модуль маршрутизации

Модуль ММАР предназначен для маршрутизации передаваемых и принимаемых блоков данных.

Графический интерфейс ММАР представлен на рисунке 16.

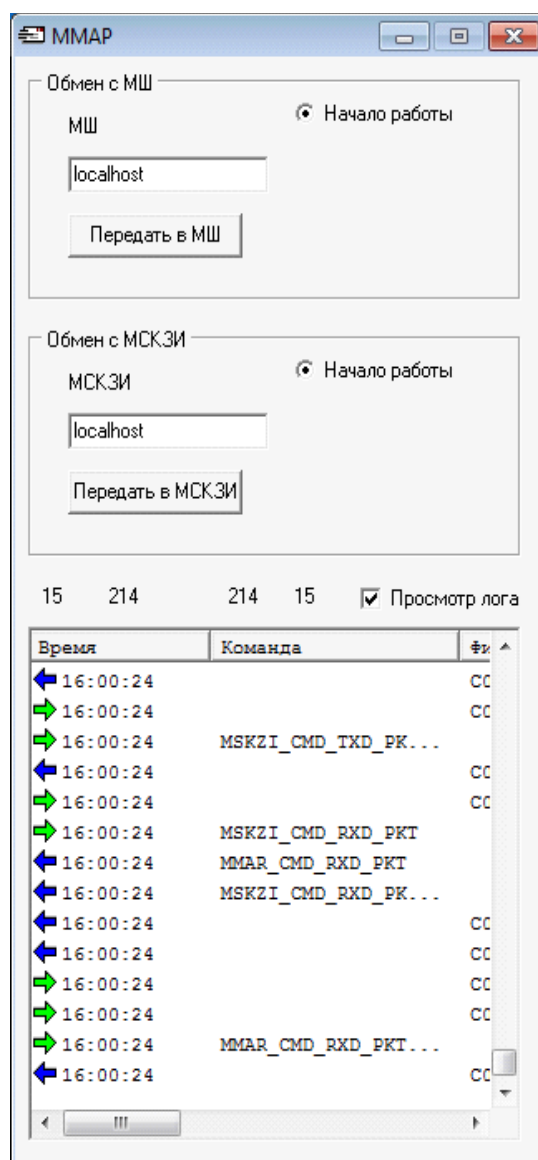


Рисунок 16 — Графический интерфейс модуля ММАР

В поле МШ «Кузнечик» необходимо ввести IP адрес ПЭВМ на котором запущен модуль МШ «Кузнечик». В поле МШ «Магма» необходимо ввести IP адрес ПЭВМ на котором запущен модуль МШ «Магма».

Для начала работы в окне «Обмен с МШ «Кузнечик»» выберите команду «Начало работы» и нажмите кнопку «Передать в МШ «Кузнечик»». В МШ «Кузнечик» будет отправлена команда «Начало работы». Получив команду МШ «Кузнечик» сформирует и передаст ответ в ММАР. В окне «Обмен с МШ «Магма»» выберите команду «Начало работы» и нажмите кнопку «Передать в МШ «Магма»». В МШ «Магма» будет отправлена команда «Начало работы». Получив команду МШ «Магма» сформирует и передаст ответ в ММАР.

Обмен команд фиксируется в окне логирования, с указанием времени и названием команды.

6.5 Модуль шифрования «Магма»

Модуль МШ «Магма» предназначен для шифрования и расшифрования блоков данных. Графический интерфейс МШ «Магма» представлен на рисунке 17.

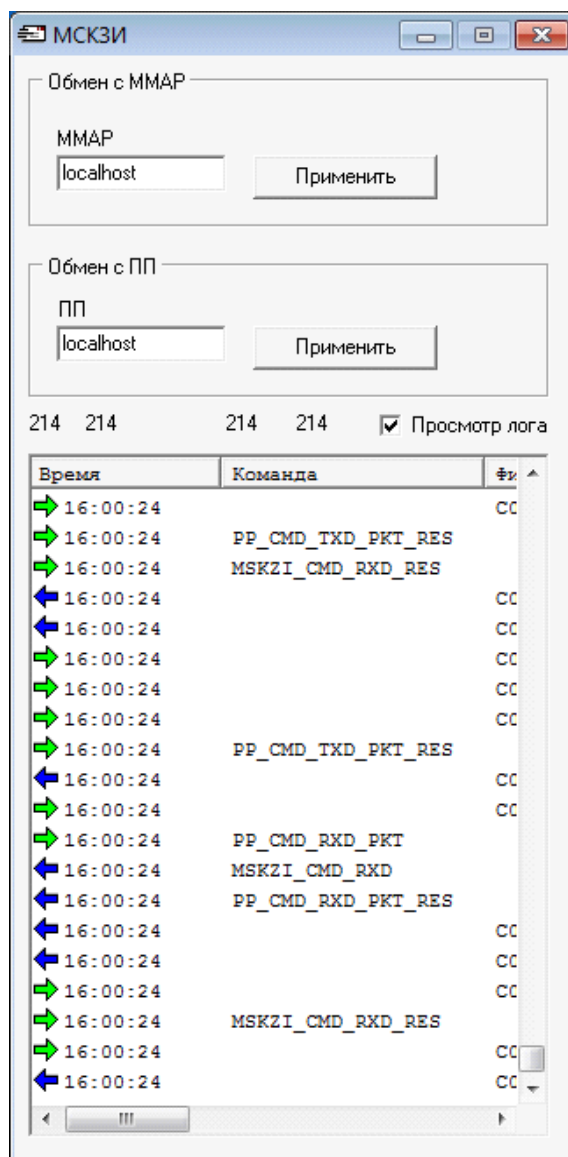


Рисунок 17 — Графический интерфейс модуля МШ «Магма»

В поле «ММАР» необходимо ввести IP адрес ПЭВМ на котором запущен модуль ММАР и нажать кнопку «Применить». В поле «ПП» необходимо ввести IP адрес ПЭВМ на котором запущен модуль ПП и нажать кнопку «Применить».

На приёме от ПП МШ «Магма» расшифровывает полученные данные и не анализирует имитовставку. Анализируется только метка времени, которая передаётся в преамбуле в открытом виде и защищена контрольной суммой.

Обмен команд фиксируется в окне логирования, с указанием времени и названием команды.

6.6 Модуль приемопередатчик

Модуль ПП предназначен для ретрансляции данных и формирование ошибок при передаче по каналу связи. Графический интерфейс ПП представлен на рисунке 18.

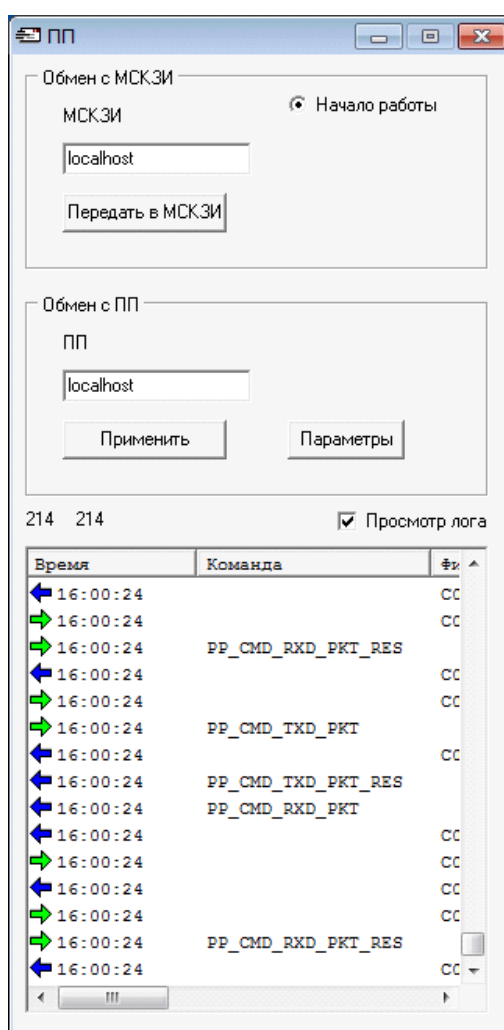


Рисунок 18 — Графический интерфейс модуля ПП

В поле «ПП» необходимо ввести IP адрес ПЭВМ на котором запущен модуль ПП и нажать кнопку «Применить». В поле МШ «Магма» необходимо

ввести IP адрес ПЭВМ на котором запущен модуль МШ «Магма».

Для начала работы в окне «Обмен с МШ Магма» выберете команду «Начало работы»и нажмите кнопку «Передать в МШ Магма». В МШ «Магма» будет отправлена команда «Начало работы» Получив команду МШ «Магма» сформирует и передаст ответ в ПП.

Нажав кнопку «Параметры» появится окно, представленное на рисунке 19.

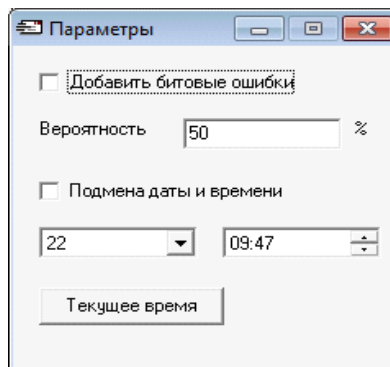


Рисунок 19 — Окно «Параметры» модуля ПП

В поле «Вероятность» можно ввести значение от 1 до 100. Установив галку «Добавить битовые ошибки» в ретранслируемые пакеты будут добавляться ошибки с выбранной вероятностью ошибок.

Выбрав дату, время и установив галку «Подмена даты и времени» в ретранслируемых пакетах будут модифицированы поле преамбулы «время», поле преамбулы «контрольная сумма». Получая пакеты от ПП, МШ «Магма» анализирует поле время в преамбуле. Если время в пакете отличается от системного в пределах пяти минут, то пакет считается скомпрометированным и отбрасывается.

Нажав кнопку «Текущее время» в полях даты и времени появится текущее системное время. Обмен команд фиксируется в окне логирования, с указанием времени и названием команды.

7 Тестирование программного обеспечения

7.1 Тестирование модуля шифрования «Кузнечик»

Целью тестирования модуля шифрования «Кузнечик» является оценка соответствия контрольному примеру, приведенному в ГОСТ Р 34.13-2015 (блочный шифр с длиной блока 128 бит).

В контрольном примере используются параметры, представленные в таблице 29.

Таблица 29 — Входные параметры

№ п/п	Входные параметры	Значения входных параметров (в шестнадцатеричном виде)
1	Ключ	8899aabbccddeeff0011223344556677fedcba98765432100123456789abcdef
2	Синхропосылка	1234567890abcef000000000000000000
3	Открытый текст	P1: 1122334455667700ffeeddccbbaa9988; P2: 00112233445566778899aabbccceeff0a; P3: 112233445566778899aabbccceeff0a00; P4: 2233445566778899aabbccceeff0a0011.

Выходные параметры представлены в таблице 30.

Таблица 30 — Выходные параметры

№ п/п	Выходные параметры	Значения выходных параметров (в шестнадцатеричном виде)
1	Шифртекст	C1: f195d8bec10ed1dbd57b5fa240bda1b8; C2: 85eee733f6a13e5df33ce4b33c45dee4; C3: a5eae88be6356ed3d5e877f13564a3a5; C4: cb91fab1f20cbab6d1c6d15820bdba73.
2	Имитовставка	336f4d296059fbe3

Таким образом, выходные параметры полностью соответствуют значениям контрольного примера.

С использованием приведенных значений, с помощью операции расшифрования воспроизводятся исходные значения открытого текста[2].

7.2 Тестирование модуля шифрования «Магма»

Целью тестирования модуля шифрования «Магма» является оценка соответствия контрольному примеру, приведенному в ГОСТ Р 34.13-2015 (блочный шифр с длиной блока 64 бита).

В контрольном примере используются параметры, представленные в таблице 31.

Таблица 31 — Входные параметры

№ п/п	Входные параметры	Значения входных параметров (в шестнадцатеричном виде)
1	Ключ	ffeeddccbbaa99887766554433221100f0f1f2f3f4f5f6f7f8f9fafbfcfdfeff
2	Синхропосылка	1234567800000000
3	Открытый текст	P1: 92def06b3c130a59; P2: db54c704f8189d20; P3: 4a98fb2e67a8024c; P4: 8912409b7b57e41.

Выходные параметры представлены в таблице 32.

Таблица 32 — Выходные параметры

№ п/п	Выходные параметры	Значения выходных параметров (в шестнадцатеричном виде)
1	Шифртекст	C1: 4e98110c97b7b93c; C2: 3e250d93d6e85d69; C3: 136d868807b2dbef; C4: 568eb680ab52a12d.

Таким образом, выходные параметры полностью соответствуют значениям контрольного примера.

С использованием приведенных значений, с помощью операции расшифрования воспроизводятся исходные значения открытого текста[2].

7.3 Тестирование квитирования сообщений

Целью тестирования квитирования передаваемых сообщений между модулями является оценка соответствия значений квитанций, описанных в протоколе взаимодействия модулей программного обеспечения с конкретным случаем работы ПО.

Для тестирования были использованы модули МИ и МШ «Кузнечик».

В качестве примера, был использован общий формат квитанций, представленный в таблице 33.

Таблица 33 — Общий формат квитанций

Начало кадра	Признак источника и получателя информации	Тип кадра	Наименование кадра	Результат выполнения команды	Контрольная сумма	Конец кадра
0xC0	0x11	0x02 0x03	0x01 0x03 0x09	«0x00» - кадр принят;	CRC16	0xC1

Результат тестирования представлен в таблице 34.

Таблица 34 — Тестирование квитирования

№ п/п	Действие	Квитанцию отправил		Значение квитанции
		МИ	МШ	
1	МИ посылает МШ команду «Начало работы с МШ»		+	C0 11 02 01 00 00 D9 13 C1
2	МШ посылает МИ команду «Начало работы с МИ»	+		C0 11 03 01 00 00 62 0F C1
3	МИ посылает МШ команду «Завершение работы с МШ»		+	C0 11 02 09 00 00 1B D5 C1
4	МИ посылает МШ команду «Начало работы с МШ» (три раза, так как МШ не включен)	—	—	—

Все значения квитанций соответствуют описанным в протоколе взаимодействия форматам.

7.4 Тестирование вычисления контрольной суммы

Целью тестирования контрольных сумм является оценка соответствия найденной контрольной суммы с контрольным примером, рассчитанным для заданных строк[6].

Для CRC16 образующий полином — «0x8408».

Расчет CRC16 производится с начальным значением «0xFFFF», а результат необходимо инвертировать. Для строки «123456789» результат CRC16 равен значению «0x29B1», что соответствует контрольному примеру.

Для CRC32 образующий полином — «0x4C11DB7» .

Расчет CRC32 производится с начальным значением 0xFFFF, а результат необходимо инвертировать. Для строки «123456789» результат CRC32 равен значению «0xCB43926», что соответствует контрольному примеру.

8 Вредные психофизиологические факторы, влияющие на нервную систему специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие

Психофизиологические опасные и вредные факторы, влияющие на нервную систему специалиста по защите информации по характеру действия подразделяются на:

- физические перегрузки;
- нервно-психические перегрузки.

Физические перегрузки подразделяются на:

- статические;
- динамические.

Нервно-психические перегрузки подразделяются на:

- умственное перенапряжение;
- перенапряжение анализаторов;
- монотонность труда;
- эмоциональные перегрузки.

Один и тот же опасный и вредный фактор по природе своего действия может относиться одновременно к различным группам.

К статическим факторам прежде всего относится вынужденное положение тела во время работы, длительное напряжение отдельных групп мышц. При длительном стоячем положении может возникнуть плоскостопие, варикозное расширение вен, изменения позвоночника (кифоз). При вынужденном сидячем положении чаще наблюдаются сколиозы и кифозы.

Профилактика сводится к механизации и автоматизации труда, рационализации рабочей позы путем совершенствования конструкций рабочих мест, производственной гимнастике, смене положения во время перерывов (работал стоя - отдыхаешь сидя и наоборот) и др.

Гиподинамия (ограничение двигательной активности) наиболее

характерна для различных видов умственного труда. Ограничение двигательной активности приводит к снижению функциональных возможностей мышечной системы: уменьшению силовых показателей, выносливости, тонуса мышц и т.д. Кроме того, гиподинамия отрицательно влияет на многие органы и системы: сердечно-сосудистую, нервную, эндокринную, обмен веществ, костно-суставной аппарат и др.

К профилактическим мероприятиям, относятся проведение производственной гимнастики, утренние зарядки, занятие физкультурой и спортом и т.д.

Постоянное напряжение, эмоциональные перегрузки неизбежно приводят к стрессам.

Признаки стресса у персонала включают в себя необычную раздражительность, другие признаки психического истощения, чрезмерное употребление алкоголя или курение, использование наркотиков.

Заключение

В процессе работы была произведена разработка алгоритмов работы модулей программного обеспечения устройства криптографической защиты информации (интерфейсного модуля, модуля шифрования), алгоритма взаимодействия модулей программного обеспечения, протокола взаимодействия модулей программного обеспечения, руководства пользователя программного обеспечения. Был произведен обзор вредных психофизиологических факторов, влияющих на нервную систему специалиста по защите информации.

В результате работы было разработано программное обеспечение устройства криптографической защиты информации и произведено тестирование программного обеспечения.

Результаты выпускной квалификационной работы были внедрены и использованы в работе сектора НИС-331 АО «ПНИЭИ».

Таким образом все пункты технического задания на ВКР были выполнены полностью.

Список используемых источников

- 1 ГОСТ Р 34.12-2015 «Информационная технология. КRYPTOграфическая защита информации. Блочные шифры»;
- 2 ГОСТ Р 34.13-2015 «Информационная технология. КRYPTOграфическая защита информации. Режимы работы блочных шифров»;
- 3 ГОСТ Р 50922-2006 «Защита информации. Основные термины и определения»;
- 4 Шнаер, Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си — Триумф, 2002. — 816 с.;
- 5 Catalogue of parametrised CRC algorithms . [Электронный ресурс] — <http://reveng.sourceforge.net/crc-catalogue/16.htm>, свободный. – Загл. с экрана.

Приложение А

(обязательное)

Алгоритм взаимодействия интерфейсный модуль-модуль шифрования

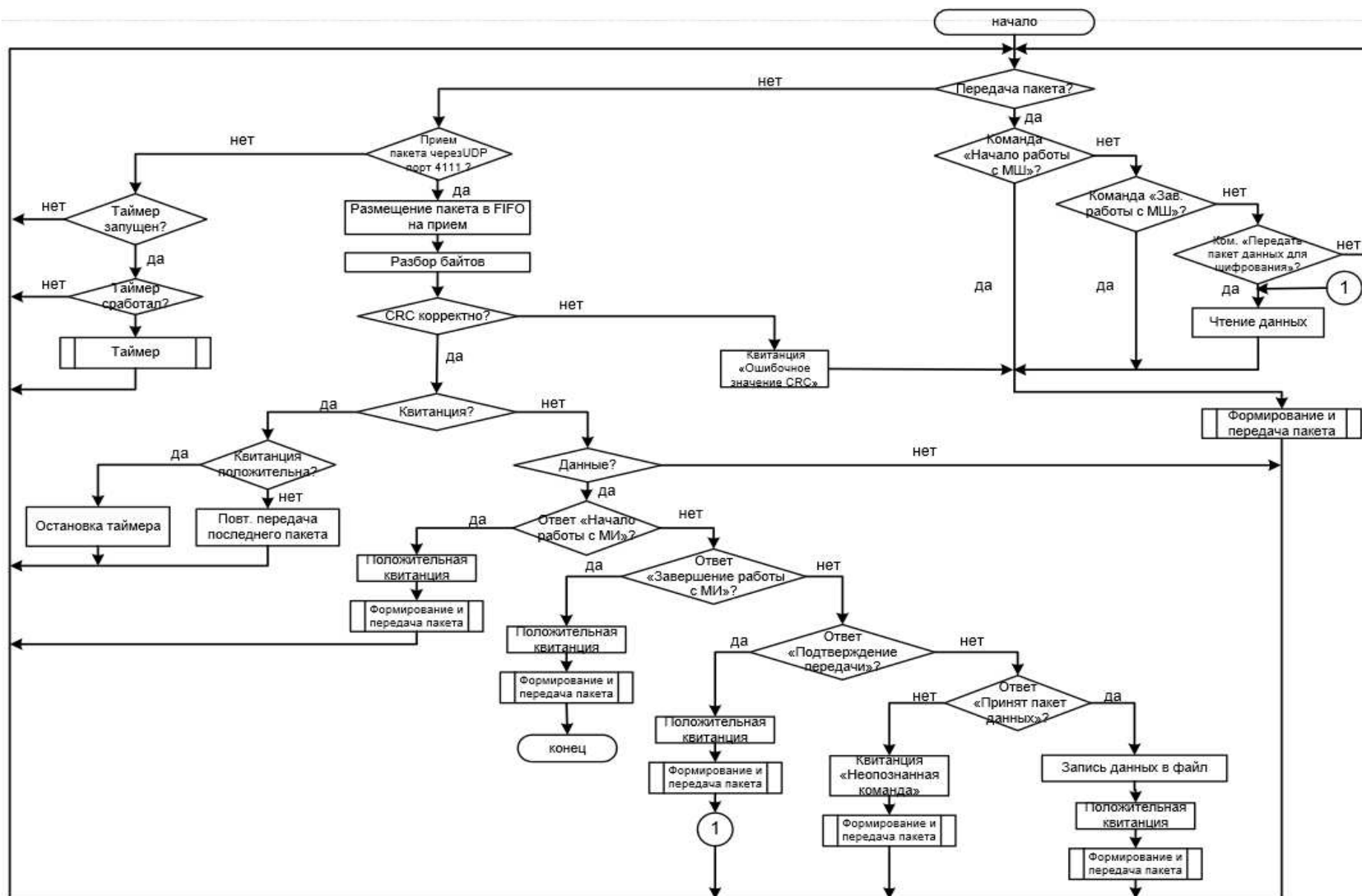


Рисунок А.1

Функция «Формирование и передача пакета»



Функция «Таймер»



Рисунок А.2

Приложение Б

(обязательное)

Алгоритм взаимодействия модуль шифрования-интерфейсный модуль



Рисунок Б.1

Приложение В

(обязательное)

Алгоритм взаимодействия модуль шифрования-модуль маршрутизации

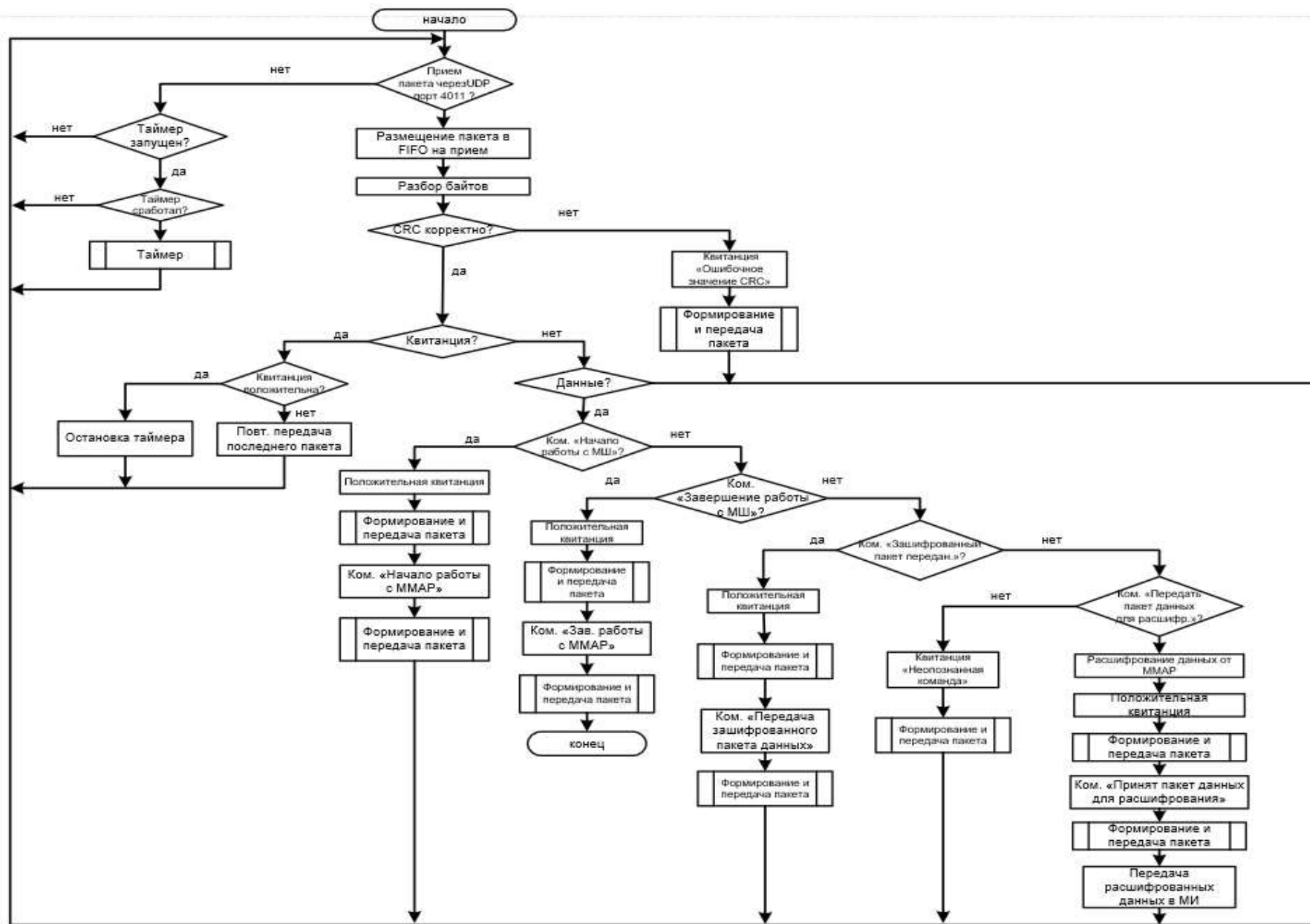


Рисунок В.1

Приложение Г

(обязательное)

Исходный текст программного обеспечения модуля шифрования «Кузнечик»

Файл «crc32.cpr»:

```
#include "crc32.h"
/** Вычисленная заранее таблица CRC32 */
static unsigned crc32_table[256] = {
    0x00000000, 0x77073096, 0xee0e612c, 0x990951ba,
    0x076dc419, 0x706af48f, 0xe963a535, 0x9e6495a3,
    0x0edb8832, 0x79dcb8a4, 0xe0d5e91e, 0x97d2d988,
    0x09b64c2b, 0x7eb17cbd, 0xe7b82d07, 0x90bf1d91,
    0x1db71064, 0x6ab020f2, 0xf3b97148, 0x84be41de,
    0x1adad47d, 0x6ddde4eb, 0xf4d4b551, 0x83d385c7,
    0x136c9856, 0x646ba8c0, 0xfd62f97a, 0x8a65c9ec,
    0x14015c4f, 0x63066cd9, 0xfa0f3d63, 0x8d080df5,
    0x3b6e20c8, 0x4c69105e, 0xd56041e4, 0xa2677172,
    0x3c03e4d1, 0x4b04d447, 0xd20d85fd, 0xa50ab56b,
    0x35b5a8fa, 0x42b2986c, 0xdbbbc9d6, 0xacbcf940,
    0x32d86ce3, 0x45df5c75, 0xdcd60dcf, 0xabd13d59,
    0x26d930ac, 0x51de003a, 0xc8d75180, 0xbfda06116,
    0x21b4f4b5, 0x56b3c423, 0xcfba9599, 0xb8bda50f,
    0x2802b89e, 0x5f058808, 0xc60cd9b2, 0xb10be924,
    0x2f6f7c87, 0x58684c11, 0xc1611dab, 0xb6662d3d,
    0x76dc4190, 0x01db7106, 0x98d220bc, 0xefd5102a,
    0x71b18589, 0x06b6b51f, 0x9fbfe4a5, 0xe8b8d433,
    0x7807c9a2, 0x0f00f934, 0x9609a88e, 0xe10e9818,
    0x7f6a0dbb, 0x086d3d2d, 0x91646c97, 0xe6635c01,
    0x6b6b51f4, 0x1c6c6162, 0x856530d8, 0xf262004e,
    0x6c0695ed, 0x1b01a57b, 0x8208f4c1, 0xf50fc457,
    0x65b0d9c6, 0x12b7e950, 0x8bb8eb8ea, 0xfcb9887c,
    0x62dd1ddf, 0x15da2d49, 0x8cd37cf3, 0xfbd44c65,
    0x4db26158, 0x3ab551ce, 0xa3bc0074, 0xd4bb30e2,
    0x4adfa541, 0x3dd895d7, 0xa4d1c46d, 0xd3d6f4fb,
    0x4369e96a, 0x346ed9fc, 0xad678846, 0xda60b8d0,
    0x44042d73, 0x33031de5, 0xaa0a4c5f, 0xdd0d7cc9,
    0x5005713c, 0x270241aa, 0xbe0b1010, 0xc90c2086,
    0x5768b525, 0x206f85b3, 0xb966d409, 0xce61e49f,
    0x5edef90e, 0x29d9c998, 0xb0d09822, 0xc7d7a8b4,
    0x59b33d17, 0x2eb40d81, 0xb7bd5c3b, 0xc0ba6cad,
    0xedb88320, 0x9abfb3b6, 0x03b6e20c, 0x74b1d29a,
    0xead54739, 0x9dd277af, 0x04db2615, 0x73dc1683,
    0xe3630b12, 0x94643b84, 0x0d6d6a3e, 0x7a6a5aa8,
    0xe40ecf0b, 0x9309ff9d, 0x0a00ae27, 0x7d079eb1,
    0xf00f9344, 0x8708a3d2, 0x1e01f268, 0x6906c2fe,
    0xf762575d, 0x806567cb, 0x196c3671, 0x6e6b06e7,
    0xfed41b76, 0x89d32be0, 0x10da7a5a, 0x67dd4acc,
```

```

0xf9b9df6f, 0x8ebeeff9, 0x17b7be43, 0x60b08ed5,
0xd6d6a3e8, 0xa1d1937e, 0x38d8c2c4, 0x4fdff252,
0xd1bb67f1, 0xa6bc5767, 0x3fb506dd, 0x48b2364b,
0xd80d2bda, 0xaf0a1b4c, 0x36034af6, 0x41047a60,
0xdf60efc3, 0xa867df55, 0x316e8eef, 0x4669be79,
0xcb61b38c, 0xbc66831a, 0x256fd2a0, 0x5268e236,
0xcc0c7795, 0xbb0b4703, 0x220216b9, 0x5505262f,
0xc5ba3bbe, 0xb2bd0b28, 0x2bb45a92, 0x5cb36a04,
0xc2d7ffa7, 0xb5d0cf31, 0x2cd99e8b, 0x5bdeae1d,
0x9b64c2b0, 0xec63f226, 0x756aa39c, 0x026d930a,
0x9c0906a9, 0xeb0e363f, 0x72076785, 0x05005713,
0x95bf4a82, 0xe2b87a14, 0x7bb12bae, 0x0cb61b38,
0x92d28e9b, 0xe5d5be0d, 0x7cdcefb7, 0x0bdbdf21,
0x86d3d2d4, 0xf1d4e242, 0x68ddb3f8, 0x1fda836e,
0x81be16cd, 0xf6b9265b, 0x6fb077e1, 0x18b74777,
0x88085ae6, 0xff0f6a70, 0x66063bca, 0x11010b5c,
0x8f659eff, 0xf862ae69, 0x616bffd3, 0x166ccf45,
0xa00ae278, 0xd70dd2ee, 0x4e048354, 0x3903b3c2,
0xa7672661, 0xd06016f7, 0x4969474d, 0x3e6e77db,
0xaed16a4a, 0xd9d65adc, 0x40df0b66, 0x37d83bf0,
0xa9bcae53, 0xdeb9ec5, 0x47b2cf7f, 0x30b5ffe9,
0xbdbdf21c, 0xcabac28a, 0x53b39330, 0x24b4a3a6,
0xbad03605, 0xcdd70693, 0x54de5729, 0x23d967bf,
0xb3667a2e, 0xc4614ab8, 0x5d681b02, 0x2a6f2b94,
0xb40bbe37, 0xc30c8ea1, 0x5a05df1b, 0x2d02ef8d
};

static unsigned crc32_update_byte(unsigned crc, unsigned char c) {
    return ((crc >> 8) ^ crc32_table[(crc ^ c) & 0xff]);
}

unsigned crc32_update(unsigned crc, const unsigned char* buf,
unsigned n) {
    if (((unsigned)buf & 3) == 0) {
        /* Оптимизированная обработка при выравненном адресе */
        while (n >= 4) {
            unsigned word = *((unsigned*)buf);
            crc = crc32_update_byte(crc, (word >> 0) & 0xff);
            crc = crc32_update_byte(crc, (word >> 8) & 0xff);
            crc = crc32_update_byte(crc, (word >> 16) & 0xff);
            crc = crc32_update_byte(crc, (word >> 24) & 0xff);
            buf += 4;
            n -= 4;
        }
    }
    /* Обработка хвоста */
    while (n) {
        crc = crc32_update_byte(crc, *buf++);
        n--;
    }
    return crc;
}

unsigned calc_csum(unsigned char* data, unsigned size) {

```

```
return crc32_final(crc32_update(CRC32_INIT_VALUE, data, size));
}
```

Файл «CRC_IRDA.cpp»:

```
File:          CRC_IRDA.c
Description: Cyclic Redundancy Check  protocol
*/
#include <vcl.h>
#pragma hdrstop

#include "CRC_IRDA.h"
#define CRC_CORRECT 0xF0B8
#pragma package(smart_init)

const unsigned short fcstab[256] = {
    0x0000, 0x1189, 0x2312, 0x329b, 0x4624, 0x57ad, 0x6536, 0x74bf,
    0x8c48, 0x9dc1, 0xaf5a, 0xbed3, 0xca6c, 0xdbe5, 0xe97e, 0xf8f7,
    0x1081, 0x0108, 0x3393, 0x221a, 0x56a5, 0x472c, 0x75b7, 0x643e,
    0x9cc9, 0x8d40, 0xbfdb, 0xae52, 0xdaed, 0xcb64, 0xf9ff, 0xe876,
    0x2102, 0x308b, 0x0210, 0x1399, 0x6726, 0x76af, 0x4434, 0x55bd,
    0xad4a, 0xbcc3, 0x8e58, 0x9fd1, 0xeb6e, 0xfae7, 0xc87c, 0xd9f5,
    0x3183, 0x200a, 0x1291, 0x0318, 0x77a7, 0x662e, 0x54b5, 0x453c,
    0xbdc b, 0xac42, 0x9ed9, 0x8f50, 0xfbef, 0xea66, 0xd8fd, 0xc974,
    0x4204, 0x538d, 0x6116, 0x709f, 0x0420, 0x15a9, 0x2732, 0x36bb,
    0xce4c, 0xdfc5, 0xed5e, 0xfcd7, 0x8868, 0x99e1, 0xab7a, 0xbaf3,
    0x5285, 0x430c, 0x7197, 0x601e, 0x14a1, 0x0528, 0x37b3, 0x263a,
    0xdec d, 0xcf44, 0xfddf, 0xec56, 0x98e9, 0x8960, 0xb bfb, 0xaa72,
    0x6306, 0x728f, 0x4014, 0x519d, 0x2522, 0x34ab, 0x0630, 0x17b9,
    0xef4e, 0xfec7, 0xcc5c, 0xdd d5, 0xa96a, 0xb8e3, 0x8a78, 0x9bf1,
    0x7387, 0x620e, 0x5095, 0x411c, 0x35a3, 0x242a, 0x16b1, 0x0738,
    0xffcf, 0xee46, 0xdcdd, 0xcd54, 0xb9eb, 0xa862, 0x9af9, 0x8b70,
    0x8408, 0x9581, 0xa71a, 0xb693, 0xc22c, 0xd3a5, 0xe13e, 0xf0b7,
    0x0840, 0x19c9, 0x2b52, 0x3adb, 0x4e64, 0x5fed, 0x6d76, 0x7cff,
    0x9489, 0x8500, 0xb79b, 0xa612, 0xd2ad, 0xc324, 0xf1bf, 0xe036,
    0x18c1, 0x0948, 0x3bd3, 0x2a5a, 0x5ee5, 0x4f6c, 0x7df7, 0x6c7e,
    0xa50a, 0xb483, 0x8618, 0x9791, 0xe32e, 0xf2a7, 0xc03c, 0xd1b5,
    0x2942, 0x38cb, 0x0a50, 0x1bd9, 0x6f66, 0x7eef, 0x4c74, 0x5dfd,
    0xb58b, 0xa402, 0x9699, 0x8710, 0xf3af, 0xe226, 0xd0bd, 0xc134,
    0x39c3, 0x284a, 0x1ad1, 0x0b58, 0x7fe7, 0x6e6e, 0x5cf5, 0x4d7c,
    0xc60c, 0xd785, 0xe51e, 0xf497, 0x8028, 0x91a1, 0xa33a, 0xb2b3,
    0x4a44, 0x5bcd, 0x6956, 0x78df, 0x0c60, 0x1de9, 0x2f72, 0x3efb,
    0xd68d, 0xc704, 0xf59f, 0xe416, 0x90a9, 0x8120, 0xb3bb, 0xa232,
    0x5ac5, 0x4b4c, 0x79d7, 0x685e, 0x1ce1, 0x0d68, 0x3ff3, 0x2e7a,
    0xe70e, 0xf687, 0xc41c, 0xd595, 0xa12a, 0xb0a3, 0x8238, 0x93b1,
    0x6b46, 0x7acf, 0x4854, 0x59dd, 0x2d62, 0x3ceb, 0x0e70, 0x1ff9,
    0xf78f, 0xe606, 0xd49d, 0xc514, 0xb1ab, 0xa022, 0x92b9, 0x8330,
    0x7bc7, 0x6a4e, 0x58d5, 0x495c, 0x3de3, 0x2c6a, 0x1ef1, 0x0f78
};
unsigned char array[] = {0x00,0x00,0x00,0x00};
```

```

unsigned short crc_irda=0xffff;

unsigned short pppfcs(unsigned short fcs,unsigned char *cp,
unsigned short len)
{
    while(len--)fcs = (0x00FF & (fcs >> 8)) ^ fcstab[(fcs ^ *cp+
+) & 0x00FF];
    return fcs;
}
// вычисление CRC
unsigned short crc_make(unsigned char *cp, unsigned short len)
{
    return (0xFFFF ^ pppfcs(0xffff,cp,len));
}
// проверка CRC
unsigned short crc_check(unsigned char *cp, unsigned short len)
{
    unsigned short crc;
    crc=pppfcs(0xffff,cp,len);
    if (crc == CRC_CORRECT) return (1); else return (0);
}

```

Файл «dte.cpp»:

```

#include "dte.h"
#include "crc_irda.h"
#include "string.h"

#define DTE_SOF      0x00C0
#define DTE_EOF      0x00C1
#define DTE_CE       0x007D
#define DTE_XORRY    0x0020

// инициализация канального протокола
void CDte::Init(CFifo* fifo_tx, CFifo* fifo_rx, unsigned
wait_time_ack, unsigned cnt_repeate, dte_sign_format_t** format,
unsigned format_size, unsigned char* log, unsigned log_size)
{
    memset(&m_dte_params, 0, sizeof(m_dte_params));

    is_timer_running = false;
    m_timer_count = 0;
    m_timer_wait = 0;

    m_fifo_tx = fifo_tx;
    m_fifo_rx = fifo_rx;

    m_time_wait_ack = wait_time_ack;
    m_cnt_repeate = cnt_repeate;

    memset(&m_dte_rxd, 0, sizeof(m_dte_rxd));
}

```

```

        memset(&m_stat, 0, sizeof(m_stat));

        for (int i=0; i<format_size; i++){
            m_formats[i] = format[i];
        }
        m_format_size = format_size;

m_dte_ack.header.type = DTE_TYPE_ACK_OUT;
m_dte_ack.header.format = 0;
m_dte_ack.header.result = 0;

        m_timer_isr = false;

        m_log.Init(log, log_size);
    }

// запуск таймера
void CDte::TimerStart(unsigned delay)
{
    is_timer_running = true;
    m_timer_wait = delay;
}

// остановка таймера
void CDte::TimerStop()
{
    m_dte_params.count = 0;
    m_dte_params.is_wait_ack = false;
    m_dte_params.size = 0;
    is_timer_running = false;
    m_timer_count = 0;
    m_timer_wait = 0;
}

// флаг срабатывания таймера
void CDte::TimerAction()
{
    m_timer_isr = true;
}

// поместить данные в очередь
unsigned CDte::SendDte(unsigned char* data, unsigned size)
{
    if (m_fifo_tx->Free() < size){
        return 0;
    }
    m_fifo_tx->Write(data, size);
    return size;
}

// сформировать квитанцию

```

```

void CDte::SendAck(dte_header_t* header, dte_ack_t type)
{
    m_dte_ack.header = *header;
    m_dte_ack.header.result = type;
    m_dte_ack.header.type = DTE_TYPE_ACK_OUT;
    m_dte_ack.data = 0;
    Send((unsigned char*)&m_dte_ack, sizeof(m_dte_ack));
}

// байтстаффинг байта
unsigned char* CDte::insert(unsigned char src, unsigned char* dst)
{
    if ((src == DTE_SOF) || (src == DTE_EOF) || (src ==
DTE_CE)) {
        *dst++ = DTE_CE;
        *dst++ = src ^ DTE_XORRY;
    }else{
        *dst++ = src;
    }
    return dst;
}

// байтстаффинг данных
unsigned CDte::Encode(unsigned char* src, unsigned char* dst,
unsigned int src_size)
{
    unsigned char* dst_begin;
    unsigned short crc;
    dst_begin = dst;
    crc = crc_make (src, src_size);

    *dst++ = DTE_SOF;
    while (src_size--) {
        dst = insert(*src++, dst);
    }
    dst = insert((char)(crc & 0xff), dst);
    dst = insert((char)(crc >> 8), dst);
    *dst++ = DTE_EOF;
    return (dst - dst_begin);
}

// подготовка к передачи
unsigned CDte::Send(unsigned char* data, unsigned size)
{
    size = Encode(data, txd_buf, size);
    return SendDte(txd_buf, size);
}

// передача данных
unsigned CDte::SendData(unsigned char* data, unsigned size)
{

```

```

        if (m_dte_params.is_wait_ack){
            return 0;
        }
        m_dte_params.is_wait_ack = true;
        m_dte_params.size = size;
        memcpy(m_dte_params.buf, data, size);
        TimerStart(m_time_wait_ack);
        m_log.Put(data, sizeof(dte_header_t));
        m_stat.send++;
        return Send(data, size);
    }
    // обработка квитанций
    void CDte::AckProcess(dte_format_ack_t* rxd_ack)
    {
        dte_header_t* header = (dte_header_t*)m_dte_params.buf;
        switch (rxd_ack->header.result){
            case DTE_ACK_OK:
                if (rxd_ack->header.format == header->format){
                    TimerStop();
                    m_dte_params.count = 0;
                    m_dte_params.size = 0;
                    m_dte_params.is_wait_ack = false;
                }
                m_stat.ack_succsess++;
            break;
            case DTE_ACK_CRC_ERR:
                m_stat.ack_crc_error++;
            break;
            case DTE_ACK_FORMAT_ERR:
                m_stat.ack_format_error++;
            break;
            case DTE_ACK_BUSY:
            break;
            case DTE_ACK_READY:
            break;
            default:
                m_stat.ack_format_error++;
            break;
        };
    }

    // проверка принятого формата данных
    bool CDte::CheckFormat(unsigned char sign, unsigned char format)
    {
        int i=0;
        dte_sign_format_t* s = NULL;
        while (i<m_format_size){
            if (m_formats[i]->sign == sign){
                s = m_formats[i];
                break;
            }
        }
    }

```

```

        i++;
    }
    if (!s){
        return false;
    }
    for (int i=0; i<s->size; i++){
        if (s->is_format[i] == format){
            return true;
        }
    }
    return false;
}
// обработка принятых данных
unsigned CDte::Receive(unsigned char* buf)
{
    unsigned res = 0;
    unsigned char byte;
    while (m_fifo_rx->Count()) {
        m_fifo_rx->Read(&byte, 1);
        if (byte == DTE_SOF){
            m_dte_rxd.buf_start = true;
            m_dte_rxd.buf_ptr = 0;
        } else if (m_dte_rxd.buf_start) {
            if (byte == DTE_CE){
                m_dte_rxd.is_xor = true;
            } else if (byte == DTE_EOF){
                if (!crc_check (m_dte_rxd.buf, m_dte_rxd.buf_ptr)){
                    m_stat.crc_error++;

SendAck((dte_header_t*)m_dte_rxd.buf, DTE_ACK_CRC_ERR);
                } else {
                    dte_header_t* header =
(dte_header_t*)m_dte_rxd.buf;
                    if (header->type !=
DTE_TYPE_CMD_IN && header->type != DTE_TYPE_ACK_IN) {
                        SendAck(header,
DTE_ACK_FORMAT_ERR);
                        m_stat.format_error
r++;
                    } else if (header->type ==
DTE_TYPE_ACK_IN) {

AckProcess((dte_format_ack_t*)m_dte_rxd.buf);
                    } else {
if (CheckFormat(header->sign, header->format) == false){
m_stat.format_error++;
SendAck(header, DTE_ACK_FORMAT_ERR);

} else {
m_stat.succsess++;
SendAck(header, DTE_ACK_OK);

```

```

memcpy(buf, m_dte_rxd.buf, m_dte_rxd.buf_ptr-2);
m_log.Put(buf, sizeof(dte_header_t));
res = m_dte_rxd.buf_ptr-2;
}
}

m_dte_rxd.buf_start = false;
m_dte_rxd.buf_ptr = 0;
} else {
if (m_dte_rxd.is_xor){
    byte ^= DTE_XORRY;
m_dte_rxd.is_xor = false;
}
m_dte_rxd.buf[m_dte_rxd.buf_ptr++] = byte;
if (m_dte_rxd.buf_ptr >= IRDA_BUF_SIZE){
m_dte_rxd.buf_start = m_dte_rxd.is_xor = false;
m_dte_rxd.buf_ptr = 0;
break;
}
}
}

return res;
}
// фоновая обработка
void CDte::Idle()
{
    if (!m_timer_isr){
        return;
    }
    m_timer_isr = false;
    if (!is_timer_running){
        return;
    }
    m_timer_count++;
    if (m_timer_count < m_timer_wait){
        return;
    }
    if (m_dte_params.count >= m_cnt_repeate) {
        TimerStop();
        m_dte_params.count = 0;
        m_dte_params.is_wait_ack = false;
        m_dte_params.size = 0;
    } else {
        Send(m_dte_params.buf, m_dte_params.size);
        TimerStart(m_time_wait_ack);
        m_dte_params.count++;
    }
}

bool CDte::isBusy()
{

```

```

        if (is_timer_running){
            return true;
        }
        if (m_fifo_rx->Count()){
            return true;
        }
        if (m_fifo_tx->Count()){
            return true;
        }
        return false;
    }
    // инициализация менеджера
    void CDteMenager::Init()
    {
        m_count = 0;
    }
    // добавление в менеджер
    bool CDteMenager::Add(CDte* dte)
    {
        if (m_count < MAX_DTE){
            m_dte[m_count++] = dte;
            return true;
        }
        return false;
    }
    // удаление из менеджера
    bool CDteMenager::Delete(CDte* dte)
    {
        int i=0;
        while (i<m_count){
            if (m_dte[i] == dte){
                m_dte[i] = NULL;
                for (int j=i; j<m_count-1; j++){
                    m_dte[j] = m_dte[j+1];
                }
                m_count--;
                return true;
            }
            i++;
        }
        return false;
    }
    // фоновая обработка
    void CDteMenager::Idle()
    {
        for (int i=0; i<m_count; i++){
            m_dte[i]->Idle();
        }
    }
    // обработчик от таймера
    void CDteMenager::TimerAction()

```

```

{
    for (int i=0; i<m_count; i++){
        m_dte[i]->TimerAction();
    }
}
bool CDteMenager::isBusy()
{
    for (int i=0; i<m_count; i++){
        if (m_dte[i]->isBusy()){
            return true;
        }
    }
    return false;
}

```

Файл «fifo.cpp»:

```

#include "fifo.h"
#include "string.h"

// инициализация фифо
void CFifo::Init(char* name, unsigned char* buffer, unsigned size)
{
    memset(&m_fifo, 0, sizeof(m_fifo));
    strcpy(m_fifo.name, name);
    m_fifo.buffer = buffer;
    m_fifo.rd = m_fifo.wr = m_fifo.count = 0;
    m_fifo.size = size;
}

// запись в фифо
unsigned CFifo::Write(unsigned char* buffer, unsigned sizeb)
{
    if (m_fifo.count + sizeb <= m_fifo.size){
        unsigned i=0, size = sizeb;
        if (m_fifo.wr + size > m_fifo.size){
            memcpy(&m_fifo.buffer[m_fifo.wr], buffer,
m_fifo.size - m_fifo.wr);
            i += m_fifo.size - m_fifo.wr;
            size -= m_fifo.size - m_fifo.wr;
            m_fifo.wr = 0;
        }
        memcpy(&m_fifo.buffer[m_fifo.wr], &buffer[i],
size);

        m_fifo.wr += size;
        if (m_fifo.wr >= m_fifo.size){
            m_fifo.wr -= m_fifo.size;
        }
        m_fifo.count += sizeb;
        return sizeb;
    }
    return 0;
}

```

```

// чтение из фифо
unsigned CFifo::Read(unsigned char* buffer, unsigned sizeb)
{
    if (m_fifo.count){
        if (m_fifo.count < sizeb){
            sizeb = m_fifo.count;
        }
        unsigned i=0, size = sizeb;
        if (m_fifo.rd + size > m_fifo.size){
            memcpy(buffer, &m_fifo.buffer[m_fifo.rd],
m_fifo.size - m_fifo.rd);
            i += m_fifo.size - m_fifo.rd;
            size -= m_fifo.size - m_fifo.rd;
            m_fifo.rd = 0;
        }
        memcpy(&buffer[i], &m_fifo.buffer[m_fifo.rd],
size);

        m_fifo.rd += size;
        if (m_fifo.rd >= m_fifo.size){
            m_fifo.rd -= m_fifo.size;
        }
        m_fifo.count -= sizeb;
        return sizeb;
    }
    return 0;
}

// вычитывание из фифо
unsigned CFifo::ReadEmpty(unsigned size)
{
    if (m_fifo.count >= size){
        m_fifo.rd += size;
        if (m_fifo.rd >= m_fifo.size){
            m_fifo.rd -= m_fifo.size;
        }
        m_fifo.count -= size;
        return size;
    }
    return 0;
}

unsigned char* CFifo::GetReadPtr()
{
    return &m_fifo.buffer[m_fifo.rd];
}

// количество данных в фифо
unsigned CFifo::Count()
{
    return m_fifo.count;
}

```

```

// свободное место в фифо
unsigned CFifo::Free()
{
    return m_fifo.size - m_fifo.count;
}

// очистка фифо
void CFifo::Clear()
{
    m_fifo.rd = m_fifo.wr = m_fifo.count = 0;
}

// название фифо
char* CFifo::Name()
{
    return m_fifo.name;
}

// инициализация менеджера
void CFifoMenager::Init()
{
    for (int i=0; i<MAX_FIFO; i++){
        m_fifo[i] = NULL;
    }
    m_count = 0;
}

// поиск фифо
CFifo* CFifoMenager::Find(char* name)
{
    for (int i=0; i<MAX_FIFO; i++){
        if (!strcmp(m_fifo[i]->Name(), name)){
            return m_fifo[i];
        }
    }
    return NULL;
}

// удаление фифо
bool CFifoMenager::Delete(CFifo* fifo)
{
    for (int i=0; i<m_count; i++){
        if (m_fifo[i] == fifo){
            m_fifo[i] = NULL;
            for (int j=i; j<m_count-1; j++){
                m_fifo[j] = m_fifo[j+1];
            }
            m_count--;
            return true;
        }
    }
}

```

```

        return false;
    }
    // добавление фифо
    bool CFifoMenager::Add(CFifo* fifo)
    {
        if (m_count < MAX_FIFO-1){
            m_fifo[m_count++] = fifo;
            return true;
        }
        return false;
    }

```

Файл «log.cpp»:

```

#include "log.h"
#include "string.h"

extern unsigned sys_time;

void CLog::Init(unsigned char* buf, unsigned size)
{
    m_buf = buf;
    m_max_size = size;
    m_ptr = 0;
}

void CLog::Put(unsigned char* buf, unsigned size)
{
    unsigned time = 0; //sys_time;
    if (m_ptr + sizeof(time) + size >= m_max_size){
        m_ptr = 0;
    }
    memcpy(&m_buf[m_ptr], &time, sizeof(time));
    memcpy(&m_buf[m_ptr + sizeof(time)], buf, size);
    m_ptr += size + sizeof(time);
}

```

Файл «main.cpp»:

```

#include <vcl.h>
#pragma hdrstop

#include "main.h"
#include "crc32.h"
#include "params.h"
#include "breeze_behaviour.h"
#include "UnitControl.h"
//-----
#pragma package(smart_init)
#pragma link "VaClasses"
#pragma link "VaComm"
#pragma link "CGAUGES"
#pragma resource "*.dfm"
#define PORT_MKI_MSH_RX      4011
#define PORT_MSH_MKI_RX      4111

```

```

#define PORT_MSH_MMAR_RX          4122
#define PORT_MMAR_MSH_RX          4022
#define PORT_MMAR_MSKZI_RX        4033
#define PORT_MSKZI_MMAR_RX        4133
#define PORT_MSKZI_PP_RX          4144
#define PORT_PP_MSKZI_RX          4044
#define RXD_BORDER_MINUTES        5

TMainForm *MainForm;
TIdSocketHandle* MMAR_MSH_SOCKET;
TIdSocketHandle* MKI_MSH_SOCKET;

#define MKU_FIFO_TX_SIZE          2048
#define MKU_FIFO_RX_SIZE          2048
#define MKU_PIPE_TX_SIZE          30
#define MKU_TIME_WAIT_ACK         2
#define MKU_TXD_MAX_REPEAT        2
#define MKU_LOG_SIZE               (100*sizeof(CDte::dte_header_t))

unsigned char m_fifo_buffer_tx_mki[MKU_FIFO_TX_SIZE];
CFifo m_fifo_tx_mki;
unsigned char m_fifo_buffer_rx_mki[MKU_FIFO_RX_SIZE];
CFifo m_fifo_rx_mki;
unsigned char m_pipe_buffer_tx_mki[MKU_PIPE_TX_SIZE*sizeof(pipe_t)];
CFifo m_pipe_tx_mki;
unsigned char m_log_buffer_mki[MKU_LOG_SIZE];

unsigned char m_fifo_buffer_tx_mmar[MKU_FIFO_TX_SIZE];
CFifo m_fifo_tx_mmar;
unsigned char m_fifo_buffer_rx_mmar[MKU_FIFO_RX_SIZE];
CFifo m_fifo_rx_mmar;
unsigned char m_pipe_buffer_tx_mmar[MKU_PIPE_TX_SIZE*sizeof(pipe_t)];
CFifo m_pipe_tx_mmar;
unsigned char m_log_buffer_mmar[MKU_LOG_SIZE];

unsigned char m_pipe_buffer_tx_data[MKU_PIPE_TX_SIZE*sizeof(pipe_t)];
CFifo m_pipe_tx_data;
CDte m_dte_mki;
CDte m_dte_mmar;
CFifoManager hw_fifo;
CDteManager hw_dte;
bool is_mmar_busy_rx;
bool from_mki_busy_rx;
bool is_init = false;

unsigned char tmp_buffer[2048];
enum {
    MKI_CMD_BEGIN                = 0x01,
    MKI_CMD_SEND_PTP              = 0x02,
    MKI_CMD_SEND                  = 0x03,
    MKI_CMD_GET_STATE             = 0x05,
    MKI_CMD_END                   = 0x09
};

```

```

AnsiString mki_mc_cmd[] = {
    "Undefine"           ,
    "MKI_CMD_BEGIN"      ,
    "MKI_CMD_SEND_PTP"   ,
    "MKI_CMD_SEND"        ,
    "Undefine"           ,
    "MKI_CMD_GET_STATE"   ,
    "Undefine"           ,
    "Undefine"           ,
    "Undefine"           ,
    "MKI_CMD_END"
};
enum {
    MKI_CMD_BEGIN_RES      = 0x01,
    MKI_CMD_SEND_PTP_RES   = 0x02,
    MKI_CMD_SEND_RES       = 0x03,
    MKI_CMD_STATE          = 0x05,
    MKI_CMD_RXD            = 0x06,
    MKI_CMD_RXD_READY      = 0x07,
    MKI_CMD_RXD_BUSY       = 0x08,
    MKI_CMD_END_RES        = 0x09
};
AnsiString mki_cm_cmd[] = {
    "Undefine"           ,
    "MKI_CMD_BEGIN_RES"   ,
    "MKI_CMD_SEND_PTP_RES",
    "MKI_CMD_SEND_RES"    ,
    "Undefine"           ,
    "MKI_CMD_STATE"       ,
    "MKI_CMD_RXD"         ,
    "MKI_CMD_RXD_READY"   ,
    "MKI_CMD_RXD_BUSY"    ,
    "MKI_CMD_END"
};
unsigned char mki_format[] = {
    MKI_CMD_BEGIN          ,
    MKI_CMD_SEND_PTP       ,
    MKI_CMD_SEND            ,
    MKI_CMD_GET_STATE      ,
    MKI_CMD_END
};
enum {
    MMAR_CMD_BEGIN         = 0x01,
    MMAR_CMD_RXD_PKT       = 0x03,
    MMAR_CMD_TXD_PKT       = 0x08
};
AnsiString mmар_cm_cmd[] = {
    "Undefine"           ,
    "MMAR_CMD_BEGIN"     ,
    "Undefine"           ,
    "MMAR_CMD_RXD_PKT"   ,
    "Undefine"           ,
    "Undefine"
};

```

```

        "Undefine"
        "Undefine"
        "MMAR_CMD_TXD_PKT_RES"
};
enum {
    MMAR_CMD_BEGIN_RES        = 0x01,
    MMAR_CMD_RXD_PKT_RES      = 0x03,
    MMAR_CMD_TXD_PKT_RES      = 0x08
};
AnsiString mmар_mc_cmd[] = {
    "Undefine"
    "MMAR_CMD_BEGIN_RES"
    "Undefine"
    "MMAR_CMD_RXD_PKT_RES"
    "Undefine"
    "Undefine"
    "Undefine"
    "Undefine"
    "MMAR_CMD_TXD_PKT"
};
unsigned char mmар_format[] = {
    MMAR_CMD_BEGIN
    MMAR_CMD_RXD_PKT
    MMAR_CMD_TXD_PKT_RES
};
enum {
    IMAGE_TXD        = 0x00,
    IMAGE_RXD        = 0x01,
    IMAGE_RXD_FAIL   = 0x02,
    IMAGE_RXD_FAIL2  = 0x03
};
enum {
    MODE_TYPE_PACKET = 0,    // данные
};
typedef struct state_msh_t {
    unsigned short mode;
    unsigned short channel;
    unsigned char datetime[8];
    unsigned short ready;
    unsigned short count_send;
    unsigned short count_rxd_succ;
    unsigned short count_rxd_err;
}state_msh_t;
typedef struct preamble_t {
    /*unsigned iv[4];
    unsigned short crypto_num;
    unsigned short key_num;
    unsigned short time;
    unsigned char type;
    unsigned char kb_num;
    unsigned icv;
    unsigned crc;*/
    bree_preamble_t pream;

```

```

void CalcCRC(){
pream.crc = calc_csum((unsigned char*)pream.iv, sizeof(preambula_t)-
sizeof(pream.crc));
}
bool CheckCRC(){
    unsigned crc32 = calc_csum((unsigned char*)pream.iv,
sizeof(preambula_t)-sizeof(pream.crc));
    if (crc32 == pream.crc){
        return true;
    }
    return false;
}
void SaveTime(){
    TDateTime DateTime = Time().CurrentDateTime();
    pream.time = DateTime.FormatString("nn").ToInt();
// minutes
    pream.time |= (DateTime.FormatString("hh").ToInt() <<
6);    // hour
    pream.time |= (DateTime.FormatString("dd").ToInt() <<
11);    // day
}
unsigned GetTimeMinutes(){
    unsigned char nn, hh, dd;
    nn = pream.time & 0x3f;           // minutes
    hh = (pream.time >> 6) & 0x1f;    // hour
    dd = (pream.time >> 11) & 0x1f;    // day
    return ((dd*24)+(hh*60)+nn);
}
bool isBorderMinutes(unsigned minutes){
    TDateTime DateTime = Time().CurrentDateTime();
    unsigned nn = DateTime.FormatString("nn").ToInt();
// minutes
    unsigned hh = DateTime.FormatString("hh").ToInt();
// hour
    unsigned dd = DateTime.FormatString("dd").ToInt();
// day
    unsigned curr = ((dd*24)+(hh*60)+nn);
    unsigned inpkt = GetTimeMinutes();
    unsigned inpkt_min = inpkt > minutes ? inpkt-
minutes : 0;
    unsigned inpkt_max = inpkt + minutes;
    if (curr >= inpkt_min && curr <= inpkt_max) {
        return true;
    }
    return false;
}
unsigned char* Data(){
    return ((unsigned char*)&pream.crc)
+sizeof(pream.crc);
}
}preambula_t;
#define MSH_PACKET_PREAM_SIZE    sizeof(preambula_t)
#define MSH_PACKET_SIZE          1536

```

```

typedef struct msh_t {
    preambula_t pream;
    unsigned char data[MSH_PACKET_SIZE];
}msh_t;

state_msh_t m_state;

CDte::dte_sign_format_t mki_sign_format      = {CDte::DTE_SIGN_MKI_MSH,
mki_format, sizeof(mki_format)};
CDte::dte_sign_format_t      mmар_sign_format      =
{CDte::DTE_SIGN_MSH_MMAR, mmар_format, sizeof(mmар_format)};
CDte::dte_sign_format_t* mki_msh_sign_format[] = {
    &mki_sign_format
};
CDte::dte_sign_format_t* msh_mmар_sign_format[] = {
    &mmар_sign_format
};
__fastcall TMainForm::TMainForm(TComponent* Owner)
    : TForm(Owner)
{
}
bool WeAreAlone(LPSTR szName)
{
    HANDLE hMutex=CreateMutex(NULL,TRUE,szName);
    if (GetLastError()==ERROR_ALREADY_EXISTS){
        CloseHandle(hMutex);
        return false;
    }
    return true;
}
AnsiString TMainForm::getMKIInCmdLog(unsigned char format)
{
    return mki_mc_cmd[format];
}
AnsiString TMainForm::getMKIOutCmdLog(unsigned char format)
{
    return mki_cm_cmd[format];
}
AnsiString TMainForm::getMMARInCmdLog(unsigned char format)
{
    return mmар_cm_cmd[format];
}
AnsiString TMainForm::getMMAROutCmdLog(unsigned char format)
{
    return mmар_mc_cmd[format];
}
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
    if (!WeAreAlone("MyMSH")){
        Application->MessageBox("Программа уже запущена",
"Предупреждение", MB_OK);
        exit(0);
    }
}

```

```

MMAR_MSH_SOCKET = UDPClientMMAR->Bindings->Add();
MKI_MSH_SOCKET = UDPClientMKI->Bindings->Add();

UDPClientMKI->Port = PORT_MSH_MKI_RX;
UDPClientMMAR->Port = PORT_MSH_MMAR_RX;
InitSockets();
m_fifo_tx_mki.Init("MKI_FIFO_TX", m_fifo_buffer_tx_mki,
MKU_FIFO_TX_SIZE);
m_fifo_rx_mki.Init("MKI_FIFO_RX", m_fifo_buffer_rx_mki,
MKU_FIFO_RX_SIZE);
m_pipe_tx_mki.Init("MKI_PIPE_TX", m_pipe_buffer_tx_mki,
MKU_PIPE_TX_SIZE*sizeof(pipe_t));
hw_fifo.Add(&m_fifo_tx_mki);
hw_fifo.Add(&m_fifo_rx_mki);
hw_fifo.Add(&m_pipe_tx_mki);
m_dte_mki.Init(&m_fifo_tx_mki, &m_fifo_rx_mki,
MKU_TIME_WAIT_ACK, MKU_TXD_MAX_REPEAT, &mki_msh_sign_format[0], 1,
m_log_buffer_mki, MKU_LOG_SIZE);
hw_dte.Add(&m_dte_mki);
m_fifo_tx_mmar.Init("MMAR_FIFO_TX", m_fifo_buffer_tx_mmar,
MKU_FIFO_TX_SIZE);
m_fifo_rx_mmar.Init("MMAR_FIFO_RX", m_fifo_buffer_rx_mmar,
MKU_FIFO_RX_SIZE);
m_pipe_tx_mmar.Init("MMAR_PIPE_TX", m_pipe_buffer_tx_mmar,
MKU_PIPE_TX_SIZE*sizeof(pipe_t));
hw_fifo.Add(&m_fifo_tx_mmar);
hw_fifo.Add(&m_fifo_rx_mmar);
hw_fifo.Add(&m_pipe_tx_mmar);
m_dte_mmar.Init(&m_fifo_tx_mmar, &m_fifo_rx_mmar,
MKU_TIME_WAIT_ACK, MKU_TXD_MAX_REPEAT, &msh_mmar_sign_format[0], 1,
m_log_buffer_mmar, MKU_LOG_SIZE);
hw_dte.Add(&m_dte_mmar);
memset(&m_state, 0, sizeof(m_state));
m_state.ready = 1;
m_pipe_tx_data.Init("MKI_PIPE_TX_DATA", m_pipe_buffer_tx_data,
MKU_PIPE_TX_SIZE*sizeof(pipe_t));
hw_fifo.Add(&m_pipe_tx_data);
is_mmar_busy_rx = false;
from mki_busy_rx = false;
is_init = true;
m_mode = MODE_TYPE_PACKET;
CryptoInit();
NSAflag = false;
}
//-----
void TMainForm::SendToMMAR(unsigned char* data, unsigned size)
{
    UDPClientMMAR->SendBuffer(data, size);
}

void TMainForm::SendToMKI(unsigned char* data, unsigned size)
{
    UDPClientMKI->SendBuffer(data, size);
}

```

```

}
void TMainForm::InitSockets()
{
    UDPServerMKI->Active = false;
    UDPServerMMAR->Active = false;

    MMAR_MSH_SOCKET->Port = PORT_MMAR_MSH_RX;
    MKI_MSH_SOCKET->Port = PORT_MKI_MSH_RX;
    try {
        UDPServerMKI->Active = true;
        UDPServerMMAR->Active = true;
    }
    catch (...) {
        Application->MessageBox("Проверте правильность
порта", "Ошибка сокета", MB_OK);
        return;
    }
}
void __fastcall TMainForm::TimerTimer(TObject *Sender)
{
    hw_dte.TimerAction();
}
//-----
void __fastcall TMainForm::UDPServerMKIUDPRead(TObject *Sender,
    TStream *AData, TIdSocketHandle *ABinding)
{
    unsigned s = AData->Size;
    unsigned char* buf = (unsigned char*)malloc(s);
    AData->Read(buf, s);

    AnsiString t;
    if (m_fifo_rx_mki.Free() >= s) {
        m_fifo_rx_mki.Write(buf, s);
    }

    if (LogCheckBox->Checked){
        for (unsigned i=0; i<s; i++){
            t += IntToHex(buf[i], 2) + " ";
        }

        TListItem* item = LogView->Items->Add();
        item->ImageIndex = 1;
        item->Caption = FormatDateTime("hh:nn:ss", Time());
        item->SubItems->Add("");
        item->SubItems->Add(t);
        LogView->Scroll(item->GetPosition().x, item-
>GetPosition().y);
    }
    free(buf);
    hw_dte.Idle();
}
void TMainForm::CreateHeaderMKI(CDte::dte_header_t* header, unsigned
char format)

```

```

{
    header->sign = CDte::DTE_SIGN_MKI_MSH;
    header->type = CDte::DTE_TYPE_CMD_OUT;
    header->format = format;
    header->result = 0;
}
void TMainForm::PipeToMKI()
{
    if (m_pipe_tx_mki.Count()){
        pipe_t* pipe = (pipe_t*)m_pipe_tx_mki.GetReadPtr();
        if (m_dte_mki.SendData(pipe->Buffer(), pipe->size)){
            free(pipe->Buffer());
            m_pipe_tx_mki.ReadEmpty(sizeof(pipe_t));
        }
    }
}
void TMainForm::PutPipeToMKI(pipe_t* pipe)
{
    if (m_pipe_tx_mki.Free() >= sizeof(pipe_t)){
        m_pipe_tx_mki.Write((unsigned char*)pipe,
sizeof(pipe_t));
    } else {
        while (1);
    }
    PipeToMKI();
}
void TMainForm::CreatePipeMKI(unsigned char cmd, unsigned char* buf,
unsigned size)
{
    pipe_t pipe;
    pipe.size = sizeof(CDte::dte_header_t) + size;
    pipe.buff = (unsigned)malloc(pipe.size);
    if (pipe.buff == NULL){
        while (1);
    }
    CDte::dte_header_t* header =
(CDte::dte_header_t*)pipe.Buffer();
    CreateHeaderMKI(header, cmd);
    if (size){
        memcpy(header->Data(), buf, size);
    }
    if (LogCheckBox->Checked){
        TListItem* item = LogView->Items->Add();
        item->ImageIndex = 0;
        item->Caption = FormatDateTime("hh:nn:ss",Time());
        item->SubItems->Add(getMKIOutCmdLog(header->format));
        item->SubItems->Add("");
        item->SubItems->Add(getDataLog((unsigned
char*)pipe.Buffer(), pipe.size));
        LogView->Scroll(item->GetPosition().x, item-
>GetPosition().y);
    }
}

```

```

        PutPipeToMKI(&pipe);
    }
    //-----
void TMainForm::CreateHeaderMMAR(CDte::dte_header_t* header, unsigned
char format)
{
    header->sign = CDte::DTE_SIGN_MSH_MMAR;
    header->type = CDte::DTE_TYPE_CMD_OUT;
    header->format = format;
    header->result = 0;
}
void TMainForm::PipeToMMAR()
{
    if (m_pipe_tx_mmar.Count()){
        pipe_t* pipe = (pipe_t*)m_pipe_tx_mmar.GetReadPtr();
        if (m_dte_mmar.SendData(pipe->Buffer(), pipe->size)){
            free(pipe->Buffer());
            m_pipe_tx_mmar.ReadEmpty(sizeof(pipe_t));
        }
    }
}
void TMainForm::PutPipeToMMAR(pipe_t* pipe)
{
    if (m_pipe_tx_mmar.Free() >= sizeof(pipe_t)){
        m_pipe_tx_mmar.Write((unsigned char*)pipe,
sizeof(pipe_t));
    } else {
        while (1);
    }
    PipeToMMAR();
}
void TMainForm::CreatePipeMMAR(unsigned char cmd, unsigned char* buf,
unsigned size)
{
    pipe_t pipe;
    pipe.size = sizeof(CDte::dte_header_t) + size;
    pipe.buff = (unsigned)malloc(pipe.size);
    if (pipe.buff == NULL){
        while (1);
    }
    CDte::dte_header_t* header = (CDte::dte_header_t*)pipe.Buffer();
    CreateHeaderMMAR(header, cmd);
    if (size){
        memcpy(header->Data(), buf, size);
    }
    if (LogCheckBox->Checked){
        TListItem* item = LogView->Items->Add();
        item->ImageIndex = 0;
        item->Caption = FormatDateTime("hh:nn:ss",Time());
        item->SubItems->Add(getMMAROutCmdLog(header-
>format));
        item->SubItems->Add("");
        item->SubItems->Add(getDataLog((unsigned

```

```

char*)pipe.Buffer(), pipe.size));
        LogView->Scroll(item->GetPosition().x, item-
>GetPosition().y);
    }

    PutPipeToMMAR(&pipe);
}
//-----
AnsiString TMainForm::getDataLog(unsigned char* data, unsigned size)
{
    AnsiString text;
    for (unsigned i=0; i<size; i++){
        text += IntToHex(data[i], 2) + " ";
    }
    return text;
}
void TMainForm::PipeTxDataToMMAR()
{
    if (!is_mmar_busy_rx && m_pipe_tx_data.Count()){
        pipe_t* pipe = (pipe_t*)m_pipe_tx_data.GetReadPtr();
        if (m_dte_mmar.SendData(pipe->Buffer(), pipe->size)){
            free(pipe->Buffer());
            m_pipe_tx_data.ReadEmpty(sizeof(pipe_t));
            is_mmar_busy_rx = true;
        }
    }
}
bool TMainForm::CreatePipeTxData(unsigned char* buf, unsigned size)
{
    pipe_t pipe;
    pipe.size = sizeof(CDte::dte_header_t) + size;
    pipe.buff = (unsigned)malloc(pipe.size);
    if (pipe.buff == NULL){
        while (1);
    }

    CDte::dte_header_t* header =
(CDte::dte_header_t*)pipe.Buffer();
    CreateHeaderMMAR(header, MMAR_CMD_TXD_PKT);
    if (size){
        memcpy(header->Data(), buf, size);
    }
    if (LogCheckBox->Checked){
        TListItem* item = LogView->Items->Add();
        item->ImageIndex = 0;
        item->Caption = FormatDateTime("hh:nn:ss",Time());
        item->SubItems->Add(getMMAROutCmdLog(header-
>format));
        item->SubItems->Add("");
        item->SubItems->Add(getDataLog((unsigned
char*)pipe.Buffer(), pipe.size));
        LogView->Scroll(item->GetPosition().x, item-
>GetPosition().y);
    }
}

```

```

        MMARTXDLabel->Caption = IntToStr(StrToInt(MMARTXDLabel->Caption)+1);

        if (m_pipe_tx_data.Free() >= sizeof(pipe_t)){
            m_pipe_tx_data.Write((unsigned char*)&pipe,
sizeof(pipe_t));
        } else {
            while (1);
        }
        if (m_pipe_tx_data.Free() >= sizeof(pipe_t)){
            return true;
        }
        return false;
    }
void TMainForm::CmdAction(CDte::dte_header_t* header, unsigned size)
{
    msh_t msh_txd;
    msh_t* msh_rxd = (msh_t*)header->Data();
    unsigned s = size - sizeof(CDte::dte_header_t);
    bool ICVflag = false;
    // FILE* f;

    if (header->sign == CDte::DTE_SIGN_MKI_MSH){
        switch (header->format){
            case MKI_CMD_BEGIN:
                CreatePipeMKI(MKI_CMD_BEGIN_RES, NULL, 0);
                break;
            case MKI_CMD_SEND:
                // encode
                memcpy(msh_txd.data, header->Data(), s);
                encrypt_packet((unsigned*)&msh_txd.data,
&msh_txd.pream.pream, (unsigned*)ki, s/4, true);
                msh_txd.pream.SaveTime();
                msh_txd.pream.CalcCRC();
                if (CreatePipeTxData((unsigned char*)&msh_txd, s +
MSH_PACKET_PREAM_SIZE)){
                    PipeTxDataToMMAR();
                    CreatePipeMKI(MKI_CMD_SEND_RES, NULL, 0);
                } else {
                    from_mki_busy_rx = true;
                }
                MKIRXDLabel->Caption = IntToStr(StrToInt(MKIRXDLabel->Caption)+1);
                break;
            case MKI_CMD_GET_STATE:
                CreatePipeMKI(MKI_CMD_STATE, (unsigned
char*)&m_state, sizeof(m_state));
                break;
            default:
                break;
        };
    }
}

```

```

        if (header->sign == CDte::DTE_SIGN_MSH_MMAR){
            switch (header->format){
                case MMAR_CMD_BEGIN:
                    CreatePipeMMAR(MMAR_CMD_BEGIN_RES, NULL, 0);
                    break;
                case MMAR_CMD_RXD_PKT:
                    // decode
                    if (msh_rxd->pream.CheckCRC()){
                        if (msh_rxd->pream.isBorderMinutes(RXD_BORDER_MINUTES)){
                            ICVflag =
                                decrypt_packet((unsigned*)msh_rxd->data, &msh_rxd->pream.pream,
                                (unsigned*)ki, (s - MSH_PACKET_PREAM_SIZE)/4, true);
                            if (!ICVflag) {
                                ParamsForm->ErrImmLabel->Caption = IntToStr(ParamsForm->ErrImmLabel->Caption.ToInt()+1);
                                if (m_mode ==
                                MODE_TYPE_STREAM) {
                                    CreatePipeMKI(MKI_CMD
                                    _RXD, msh_rxd->data, s - MSH_PACKET_PREAM_SIZE);
                                }
                                } else {
                                    CreatePipeMKI(MKI_CMD_RXD,
                                    msh_rxd->data, s - MSH_PACKET_PREAM_SIZE);
                                }
                            }
                        } else {
                            ParamsForm->ErrCRCLabel->Caption =
                                IntToStr(ParamsForm->ErrCRCLabel->Caption.ToInt()+1);
                        }
                        CreatePipeMMAR(MMAR_CMD_RXD_PKT_RES, NULL, 0);
                        MMARRXDLabel->Caption =
                                IntToStr(StrToInt(MMARRXDLabel->Caption)+1);
                        MKITXDLabel->Caption = IntToStr(StrToInt(MKITXDLabel->Caption)+1);
                    }
                    break;
                case MMAR_CMD_TXD_PKT_RES:
                    is_mmar_busy_rx = false;
                    PipeTxDataToMMAR();
                    break;
                default:
                    break;
            };
        }
    }

void TMainForm::MKIIdle()
{
    unsigned char *buffer;
    if (from_mki_busy_rx && m_pipe_tx_data.Free() >=
    sizeof(pipe_t)){
        CreatePipeMKI(MKI_CMD_SEND_RES, NULL, 0);
        from_mki_busy_rx = false;
    }
}

```

```

    }
    unsigned size = m_fifo_tx_mki.Count();
    if (size){
        buffer = (unsigned char*)malloc(size);
        if (buffer){
            m_fifo_tx_mki.Read(buffer, size);
            SendToMKI(buffer, size);
            if (LogCheckBox->Checked){
                TListItem* item = LogView->Items-
>Add();
                item->ImageIndex = 0;
                item->Caption =
FormatDateTime("hh:nn:ss",Time());
                item->SubItems->Add("");
                item->SubItems-
>Add(getDataLog(buffer, size));
                LogView->Scroll(item-
>GetPosition().x, item->GetPosition().y);
            }
        }
    }
    size = m_dte_mki.Receive(tmp_buffer);
    if (size >= sizeof(tmp_buffer)){
        while (1);
    }
    if (size){
        CDte::dte_header_t* header =
(CDte::dte_header_t*)tmp_buffer;
        if (LogCheckBox->Checked){
            TListItem* item = LogView->Items->Add();
            item->ImageIndex = 1;
            item->Caption =
FormatDateTime("hh:nn:ss",Time());
            item->SubItems->Add(getMKIInCmdLog(header-
>format));
            item->SubItems->Add("");
            item->SubItems->Add(getDataLog((unsigned
char*)header, size));
            LogView->Scroll(item->GetPosition().x, item-
>GetPosition().y);
        }
        CmdAction(header, size);
    }
    PipeToMKI();
}

void TMainForm::MMARIdle()
{
    unsigned char *buffer;
    unsigned size = m_fifo_tx_mmar.Count();
    if (size){
        buffer = (unsigned char*)malloc(size);

```

```

        if (buffer){
            m_fifo_tx_mmar.Read(buffer, size);
            SendToMMAR(buffer, size);
            if (LogCheckBox->Checked){
                TListItem* item = LogView->Items-
>Add();
                item->ImageIndex = 0;
                item->Caption =
FormatDateTime("hh:nn:ss",Time());
                item->SubItems->Add("");
                item->SubItems-
>Add(getDataLog(buffer, size));
                LogView->Scroll(item-
>GetPosition().x, item->GetPosition().y);
            }
        }
        size = m_dte_mmar.Receive(tmp_buffer);
        if (size >= sizeof(tmp_buffer)){
            while (1);
        }
        if (size){
            CDte::dte_header_t* header =
(CDte::dte_header_t*)tmp_buffer;
            if (LogCheckBox->Checked){
                TListItem* item = LogView->Items->Add();
                item->ImageIndex = 1;
                item->Caption =
FormatDateTime("hh:nn:ss",Time());
                item->SubItems->Add(getMMARInCmdLog(header-
>format));
                item->SubItems->Add("");
                item->SubItems->Add(getDataLog((unsigned
char*)header, size));
                LogView->Scroll(item->GetPosition().x, item-
>GetPosition().y);
            }
            CmdAction(header, size);
        }
        PipeToMMAR();
    }

void __fastcall TMainForm::WaitTimerTimer(TObject *Sender)
{
    if (!is_init){
        return;
    }
    MKIIdle();
    MMARIdle();
    hw_dte.Idle();
}
//-----
void __fastcall TMainForm::N3Click(TObject *Sender)

```

```

{
    LogView->Clear();
    MMARTXDLabel->Caption = "0";
    MMARRXDLabel->Caption = "0";
    MKITXDLabel->Caption = "0";
    MKIRXDLabel->Caption = "0";
}
//-----
void __fastcall TMainForm::UDPServerMMARUDPRead(TObject *Sender,
    TStream *AData, TIdSocketHandle *ABinding)
{
    unsigned s = AData->Size;
    unsigned char* buf = (unsigned char*)malloc(s);
    AData->Read(buf, s);
    AnsiString t;
    if (m_fifo_rx_mmar.Free() >= s) {
        m_fifo_rx_mmar.Write(buf, s);
    }
    if (LogCheckBox->Checked){
        for (unsigned i=0; i<s; i++){
            t += IntToHex(buf[i], 2) + " ";
        }

        TListItem* item = LogView->Items->Add();
        item->ImageIndex = 1;
        item->Caption = FormatDateTime("hh:nn:ss",Time());
        item->SubItems->Add("");
        item->SubItems->Add(t);
        LogView->Scroll(item->GetPosition().x, item-
>GetPosition().y);
    }
    free(buf);
    hw_dte.Idle();
}
//-----
void __fastcall TMainForm::Button1Click(TObject *Sender)
{
    InitSockets();
}
//-----
void __fastcall TMainForm::SendUDPClick(TObject *Sender)
{
    InitSockets();
}
//-----
void __fastcall TMainForm::ParamsButtonClick(TObject *Sender)
{
    ParamsForm->StreamRadioButton->Checked = m_mode;
    ParamsForm->Show();
}
//-----
void __fastcall TMainForm::ButtonControlClick(TObject *Sender)
{

```

```

        FormControl->Show();
    }
    //-----
void __fastcall TMainForm::ButtonNSAClick(TObject *Sender)
{
    NSAflag = true;
    UDPServerMKI->Active = false;
    UDPServerMMAR->Active = false;
    MessageDlg("Несанкционированный доступ! Дальнейшая работа
невозможна!",
               mtError, TMsgDlgButtons() << mbOK, 0);
}
    //-----
void __fastcall TMainForm::ButtonCompromiseClick(TObject *Sender)
{
    ButtonNSAClick(this);
}
    //-----

```

Файл «MSH.cpp»:

```

#include <vcl.h>
#pragma hdrstop
//-----
USEFORM("main.cpp", MainForm);
USEFORM("params.cpp", ParamsForm);
USEFORM("UnitControl.cpp", FormControl);
//-----
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
    try
    {
        Application->Initialize();
        Application->CreateForm(__classid(TMainForm),
&MainForm);
        Application->CreateForm(__classid(TParamsForm),
&ParamsForm);
        Application->CreateForm(__classid(TFormControl),
&FormControl);
        Application->Run();
    }
    catch (Exception &exception)
    {
        Application->ShowException(&exception);
    }
    catch (...)
    {
        try
        {
            throw Exception("");

```

```

    }
    catch (Exception &exception)
    {
        Application->ShowException(&exception);
    }
}
return 0;
}

```

Файл «gostgrasshopper.cpp»:

```

#include <memory.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <conio.h>
#define kImit14ContextLen 0x87 //размер контекста для режима формирования
имитовставки
//Нелинейное биективное преобразование множества двоичных векторов
static const unsigned char kPi[256] =
{
    252, 238, 221, 17, 207, 110, 49, 22, 251, 196, 250, 218, 35,
197, 4, 77,
    233, 119, 240, 219, 147, 46, 153, 186, 23, 54, 241, 187, 20,
205, 95, 193,
    249, 24, 101, 90, 226, 92, 239, 33, 129, 28, 60, 66, 139,
1, 142, 79,
    5, 132, 2, 174, 227, 106, 143, 160, 6, 11, 237, 152, 127,
212, 211, 31,
    235, 52, 44, 81, 234, 200, 72, 171, 242, 42, 104, 162, 253,
58, 206, 204,
    181, 112, 14, 86, 8, 12, 118, 18, 191, 114, 19, 71, 156,
183, 93, 135,
    21, 161, 150, 41, 16, 123, 154, 199, 243, 145, 120, 111, 157,
158, 178, 177,
    50, 117, 25, 61, 255, 53, 138, 126, 109, 84, 198, 128, 195,
189, 13, 87,
    223, 245, 36, 169, 62, 168, 67, 201, 215, 121, 214, 246, 124,
34, 185, 3,
    224, 15, 236, 222, 122, 148, 176, 188, 220, 232, 40, 80, 78,
51, 10, 74,
    167, 151, 96, 115, 30, 0, 98, 68, 26, 184, 56, 130, 100,
159, 38, 65,
    173, 69, 70, 146, 39, 94, 85, 47, 140, 163, 165, 125, 105,
213, 149, 59,
    7, 88, 179, 64, 134, 172, 29, 247, 48, 55, 107, 228, 136,
217, 231, 137,
    225, 27, 131, 73, 76, 63, 248, 254, 141, 83, 170, 144, 202,
216, 133, 97,
    32, 113, 103, 164, 45, 43, 9, 91, 203, 155, 37, 208, 190,
229, 108, 82,
    89, 166, 116, 210, 230, 244, 180, 192, 209, 102, 175, 194, 57,
75, 99, 182
};
//Обратное нелинейное биективное преобразование множества двоичных
векторов

```

```

static const unsigned char kReversePi[256] =
{
    0xa5,0x2d,0x32,0x8f,0x0e,0x30,0x38,0xc0,0x54,0xe6,0x9e,0x39,0x55,0x7
e,0x52,0x91,
    0x64,0x03,0x57,0x5a,0x1c,0x60,0x07,0x18,0x21,0x72,0xa8,0xd1,0x29,0xc
6,0xa4,0x3f,
    0xe0,0x27,0x8d,0x0c,0x82,0xea,0xae,0xb4,0x9a,0x63,0x49,0xe5,0x42,0xe
4,0x15,0xb7,
    0xc8,0x06,0x70,0x9d,0x41,0x75,0x19,0xc9,0xaa,0xfc,0x4d,0xbf,0x2a,0x7
3,0x84,0xd5,
    0xc3,0xaf,0x2b,0x86,0xa7,0xb1,0xb2,0x5b,0x46,0xd3,0x9f,0xfd,0xd4,0x0
f,0x9c,0x2f,
    0x9b,0x43,0xef,0xd9,0x79,0xb6,0x53,0x7f,0xc1,0xf0,0x23,0xe7,0x25,0x5
e,0xb5,0x1e,
    0xa2,0xdf,0xa6,0xfe,0xac,0x22,0xf9,0xe2,0x4a,0xbc,0x35,0xca,0xee,0x7
8,0x05,0x6b,
    0x51,0xe1,0x59,0xa3,0xf2,0x71,0x56,0x11,0x6a,0x89,0x94,0x65,0x8c,0xb
b,0x77,0x3c,
    0x7b,0x28,0xab,0xd2,0x31,0xde,0xc4,0x5f,0xcc,0xcf,0x76,0x2c,0xb8,0xd
8,0x2e,0x36,
    0xdb,0x69,0xb3,0x14,0x95,0xbe,0x62,0xa1,0x3b,0x16,0x66,0xe9,0x5c,0x6
c,0x6d,0xad,
    0x37,0x61,0x4b,0xb9,0xe3,0xba,0xf1,0xa0,0x85,0x83,0xda,0x47,0xc5,0xb
0,0x33,0xfa,
    0x96,0x6f,0x6e,0xc2,0xf6,0x50,0xff,0x5d,0xa9,0x8e,0x17,0x1b,0x97,0x7
d,0xec,0x58,
    0xf7,0x1f,0xfb,0x7c,0x09,0x0d,0x7a,0x67,0x45,0x87,0xdc,0xe8,0x4f,0x1
d,0x4e,0x04,
    0xeb,0xf8,0xf3,0x3e,0x3d,0xbd,0x8a,0x88,0xdd,0xcd,0x0b,0x13,0x98,0x0
2,0x93,0x80,
    0x90,0xd0,0x24,0x34,0xcb,0xed,0xf4,0xce,0x99,0x10,0x44,0x40,0x92,0x3
a,0x01,0x26,
    0x12,0x1a,0x48,0x68,0xf5,0x81,0x8b,0xc7,0xd6,0x20,0x0a,0x08,0x00,0x4
c,0xd7,0x74
};
//Коэффициенты умножения в преобразовании 1
static const unsigned char kB[16] = {148, 32, 133, 16, 194, 192, 1, 251,
1, 192, 194, 16, 133, 32, 148, 1};

int funcX(unsigned char* a, unsigned char* b, unsigned char* outdata)
{
    unsigned int i;

    if(!a || !b || !outdata)
    {
        printf("funcX: internal error! \n");
        return -1;
    }

    for(i = 0; i < 16; ++i)
    {
        outdata[i] = a[i] ^ b[i];
    }

    return 0;
}

```

```

int funcS(unsigned char* indata, unsigned char* outdata)
{
    size_t i = 0;

    if(!indata || !outdata)
    {
        printf("funcS: internal error! \n");
        return -1;
    }

    for(i = 0; i < 16; ++i)
    {
        outdata[i] = kPi[indata[i]];
    }

    return 0;
}

//Коэффициенты умножения в преобразовании l hex
static const uint8_t kuz_lvec[16] = { 0x94, 0x20, 0x85, 0x10, 0xc2, 0xc0,
0x01, 0xfb,
                                0x01, 0xc0, 0xc2, 0x10, 0x85, 0x20,
0x94, 0x01};

static uint8_t kuz_mul_gf256(uint8_t x, uint8_t y)
{
    uint8_t z;
    z = 0;

    while(y)
    {
        if (y & 1) z^=x;
        x = (x << 1) ^ (x & 0x80 ? 0xC3 : 0x00);
        y >>= 1;
    }
    return z;
}

int funcR(unsigned char* indata, unsigned char* outdata)
{
    size_t i;
    unsigned char sum = 0;

    if(!indata || !outdata)
    {
        printf("funcR: internal error! \n");
        return -1;
    }

    for(i = 0; i < 16; ++i)
    {

        sum ^= kuz_mul_gf256(indata[i], kuz_lvec[i]);
    }
}

```

```

        outdata[0] = sum;
        memcpy(outdata+1, indata, 15);

        return 0;
}

int funcL(unsigned char* indata, unsigned char* outdata)
{
    unsigned char tmp[16];
    int i = 0;

    if(!indata || !outdata)
    {
        printf("funcL: internal error! \n");
        return -1;
    }
    memcpy(tmp, indata, 16);
    for(i = 0; i < 16; ++i)
    {
        funcR(tmp, outdata);
        memcpy(tmp, outdata, 16);
    }
    return 0;
}

int funcLSX(unsigned char* a, unsigned char* b, unsigned char* outdata)
{
    unsigned char temp1[16];
    unsigned char temp2[16];
    if(!a || !b || !outdata)
    {
        printf("funcLSX: internal error! \n");
        return -1;
    }

    funcX(a, b, temp1);
    funcS(temp1, temp2);
    funcL(temp2, outdata);
    return 0;
}

int funcF(unsigned char* inputKey, unsigned char* inputKeySecond,
unsigned char* iterationConst, unsigned char* outputKey, unsigned char*
outputKeySecond)
{
    unsigned char temp1[16];
    unsigned char temp2[16];

    if(!inputKey || !inputKeySecond || !iterationConst || !outputKey
|| !outputKeySecond)
    {
        printf("funcF: internal error!", 0, 0);
        return -1;
    }
}

```

```

    funcLSX(inputKey, iterationConst, temp1);
    funcX(temp1, inputKeySecond, temp2);

    memcpy(outputKeySecond, inputKey, 16);
    memcpy(outputKey, temp2, 16);

    return 0;
}

int funcC(unsigned char number, unsigned char* output)
{
    unsigned char tempI[16];

    if(!output)
    {
        printf("funcC: internal error! \n");
        return -1;
    }

    memset( tempI, 0, 15 );
    tempI[15] = number;
    funcL(tempI, output);

    return 0;
}

int ExpandKey(unsigned char* masterKey, unsigned char* keys)
{
    unsigned char C[16];
    unsigned char temp1[16];
    unsigned char temp2[16];
    unsigned char j, i;

    if(!masterKey || !keys)
    {
        printf("ExpandKey: internal error! \n");
        return -1;
    }

    memcpy(keys, masterKey, 16);
    memcpy(keys + 16, masterKey + 16, 16);

    printf("ExpandKey: master key: \n",masterKey);
    printf("ExpandKey: output key: \n",keys);
    printf("ExpandKey: output key +16: \n", keys + 16);

    for(j = 0; j < 4; ++j)
    {
        memcpy(temp1, keys + j * 2 * 16, 16);
        memcpy(temp2, keys + (j * 2 + 1) * 16, 16);

        for( i = 1; i < 8; ++i )
        {

```

```

        funcC(j*8+i, C);
        funcF(temp1, temp2, C, temp1, temp2);
    }

    funcC(j*8+8, C);
    funcF(temp1, temp2, C, temp1, temp2);

    memcpy(keys + (j * 2 + 2) * 16, temp1, 16);
    memcpy(keys + (j * 2 + 3) * 16, temp2, 16);

    //print_array("ExpandKey: output key: ", keys + (j * 2 + 2)
* 16, 16);
    //print_array("ExpandKey: output key: ", keys + (j * 2 + 3)
* 16, 16);
    }
    return 0;
}

int Encrypt_14(unsigned char* plainText, unsigned char* chipherText,
unsigned char* keys)
{
    unsigned char xTemp[16];
    unsigned char yTemp[16];
    int i;

    if(!plainText || !chipherText || !keys)
    {
        printf("Encrypt_14: internal error! \n");
        return -1;
    }
    memcpy(xTemp, plainText, 16);

    for(i = 0; i < 9; ++i)
    {
        funcLSX(xTemp, keys + 16*i, yTemp);
        memcpy(xTemp, yTemp, 16);
    }
    funcX(yTemp, keys+9*16, chipherText);

    {

        printf("Encrypt_14: plain text: ", plainText);
        printf("Encrypt_14: chipher text: ", chipherText);

    }

    return 0;
}

unsigned long long convert32(unsigned long long l)
{
    unsigned long long res = (l << 56);
    res |= ((l<<40) & 0x00ff000000000000);
    res |= ((l<<24) & 0x0000ff0000000000);
    res |= ((l<<8) & 0x000000ff00000000);
    res |= (l>>56);
}

```

```

    res |= ((l>>40) & 0x0000000000000ff00);
    res |= ((l>>24) & 0x000000000000ff0000);
    res |= ((l>>8) & 0x00000000ff000000);
    return res;
}
int encrypt_packet(unsigned char* in, unsigned char* out, /*unsigned
char**/unsigned long long* iv, int len, unsigned char* K)
{
    int i;
    unsigned char temp[16];
    while(len--)
    {
        Encrypt_14((unsigned char*)iv, temp, K);

        for(i=0; i<16; i++)
        {
            *out++ = *in++ ^ temp[i];
        }

        iv[0] = convert32(iv[0]);
        iv[1] = convert32(iv[1]);

        iv[1]++;
        if (iv[1] == 0){
            iv[0]++;
        }

        iv[0] = convert32(iv[0]);
        iv[1] = convert32(iv[1]);

    }

    return 0;
}
static void ShifttLeftOne(unsigned char *r, size_t length) //для
выработки вспом. ключей имитовставки
{
    size_t i;

    for(i =0; i < length -1 ; ++i)
    {
        r[i] <= 1;
        r[i] &= 0xfe;
        r[i] |= ((r[i+1]>>7)&0x1);
    }

    r[length -1] <= 1;
    r[length -1] &= 0xfe;
}

static int ExpandImitKey(unsigned char *K, size_t BlockLen, unsigned
char* key_imit1, unsigned char* key_imit2) //BlockLen=16
{
    unsigned char tmpblock[16]; // временный блок
    unsigned char R[16];
    unsigned char B[16]; //вспомогательный параметр B

```

```

int r;
size_t i;

if(!K || !BlockLen)
    return -1;

memset(tmpblock, 0, BlockLen);

Encrypt_14(tmpblock, R, K); //1 (ГОСТ)

r = ((R[0] & 0x80) == 0x80); //2

ShifttLeftOne(R, BlockLen); //3

    memset(B, 0, 16);
    B[15] = 0x87; //16-1

if(r == 1)
{
    for(i = 0; i < BlockLen; ++i)
    {
        key_imit1[i] = R[i] ^ B[i];
    }
}
else
{
    memcpy(key_imit1, R, BlockLen);
}

memcpy(tmpblock, key_imit1, BlockLen);

ShifttLeftOne(tmpblock, BlockLen);

if((key_imit1[0] & 0x80) == 0x80)
{
    for(i = 0; i < BlockLen; ++i)
    {
        key_imit2[i] = tmpblock[i] ^ B[i];
    }
}
else
{
    memcpy(key_imit2, tmpblock, BlockLen);
}

return 0;
}

int mac(unsigned char* in, unsigned char* out, int len, unsigned char* K)
//K-раундовые ключи базового преобразования, K1 K2 - вспомогательные
ключи имитовставки
{
    int i;
    unsigned long long Co = 0;
    unsigned char temp_x[16];

```

```

unsigned char Ci[16];
unsigned char resimit[16];

unsigned char K_imit[16];
unsigned char K1[16];
unsigned char K2[16]; //значение не берем
ExpandImitKey(K, 16, K1, K2);

for(i=0; i<16; i++)
{
    temp_x[i] = *in++ ^ Co;
}
Encrypt_14(temp_x, Ci, K);

while(len--)
{
    if(len!=1)
    {
        for(i=0; i<16; i++)
        {
            temp_x[i] = *in++ ^ Ci[i];
        }

        Encrypt_14(temp_x, Ci, K);
    }
    else
    {
        for(i = 0; i < 16; ++i)
        {
            Ci[i] ^= K1[i];
            Ci[i] ^=*in++;
        }

        Encrypt_14(Ci, resimit, K);
        memcpy(out, resimit, 8);
    }
}
return 0;
}

int main()
{
    #define BUFF1 64
    #define BUFF2 64
    #define BUFF3 8
    #define FILE_PATH 32
    char file_path[FILE_PATH];
    unsigned char buf1[BUFF1]; //opentext
    unsigned char C1 [BUFF2]; //ciphertext
    unsigned char imitovstavka [BUFF3]; //imit

    unsigned char synhr1[]={0x12, 0x34, 0x56, 0x78, 0x90, 0xab, 0xce,
0xf0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}; //IV

    unsigned char Mkey[]={0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee,
0xff, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0xfe, 0xdc, 0xba,
0x98, 0x76, 0x54, 0x32, 0x10, 0x01, 0x23, 0x45, 0x67, 0x89, 0xab, 0xcd,

```

```

0xef}); //masterkey

    unsigned char roundkeys[160]; //keys
    ExpandKey(Mkey, roundkeys);
    int k;
    printf("GOST 34.13-2015.CTR,n-64\n");
    FILE *f = fopen("opentext.bin", "rb");
    FILE *res = fopen("ciphertext.bin", "wb+");

    int size;
    fseek(f, 0L, SEEK_END);
    size = ftell(f);
    fseek(f, 0L, SEEK_SET);
    if ((f!=NULL)&&(res!=NULL))
    {
        while (1)
        {
            k = fread (buf1, 1, sizeof(buf1), f);
            if (k == 0)
            {
                break;
            }
            CounterGost(buf1, C1, (unsigned long long*)synhr1,
sizeof((unsigned long long*)buf1), roundkeys); //CTR gost
            fwrite(C1, 1, sizeof(C1), res);
            mac(buf1, imitovstavka, sizeof((unsigned long
long*)buf1), roundkeys);
        }

        fwrite(imitovstavka, 1, sizeof(imitovstavka), res);
        printf("Encryption complete\n");
        //printf("mac = %x%x\n", mac[0],mac[1]);
        fclose(f);
        fclose(res);
    }
    else
    {
        if (f == NULL)
            printf("Failed to opentext file");
        if (res == NULL)
            printf("Failed to write ciphertext");
    }

    return 0;
}

```

Приложение Д

(обязательное)

Исходный текст программного обеспечения модуля шифрования «Магма»

Файл «main.cpp»

```
#include <vcl.h>
#pragma hdrstop
#include "main.h"
#include "crc32.h"
//-----
#pragma package(smart_init)
#pragma link "VaClasses"
#pragma link "VaComm"
#pragma link "CGAUGES"
#pragma resource "*.dfm"
#define PORT_MKI_MSH_RX          4011
#define PORT_MSH_MKI_RX          4111
#define PORT_MSH_MMAR_RX         4122
#define PORT_MMAR_MSH_RX         4022
#define PORT_MMAR_MSKZI_RX       4033
#define PORT_MSKZI_MMAR_RX       4133
#define PORT_MSKZI_PP_RX         4144
#define PORT_PP_MSKZI_RX         4044
#define RXD_BORDER_MINUTES      5

TMainForm *MainForm;
TIdSocketHandle* MMAR_MZI_SOCKET;
TIdSocketHandle* MZI_PP_SOCKET;

#define MKU_FIFO_TX_SIZE        2048
#define MKU_FIFO_RX_SIZE        2048
#define MKU_PIPE_TX_SIZE        30
#define MKU_PIPE_RX_SIZE        30
#define MKU_TIME_WAIT_ACK       2
#define MKU_TXD_MAX_REPEAT     2
#define MKU_LOG_SIZE             (100*sizeof(CDte::dte_header_t))

unsigned char m_fifo_buffer_tx_pp[MKU_FIFO_TX_SIZE];
CFifo m_fifo_tx_pp;
unsigned char m_fifo_buffer_rx_pp[MKU_FIFO_RX_SIZE];
CFifo m_fifo_rx_pp;
unsigned char m_pipe_buffer_tx_pp[MKU_PIPE_TX_SIZE*sizeof(pipe_t)];
CFifo m_pipe_tx_pp;
unsigned char m_log_buffer_pp[MKU_LOG_SIZE];
unsigned char m_fifo_buffer_tx_mmar[MKU_FIFO_TX_SIZE];
CFifo m_fifo_tx_mmar;
unsigned char m_fifo_buffer_rx_mmar[MKU_FIFO_RX_SIZE];
CFifo m_fifo_rx_mmar;
unsigned char m_pipe_buffer_tx_mmar[MKU_PIPE_TX_SIZE*sizeof(pipe_t)];
CFifo m_pipe_tx_mmar;
```

```

unsigned char m_log_buffer_mmar[MKU_LOG_SIZE];

CDte m_dte_pp;
CDte m_dte_mmar;
CFifoMenager hw_fifo;
CDteMenager hw_dte;

unsigned char m_pipe_buffer_tx_data[MKU_PIPE_TX_SIZE*sizeof(pipe_t)];
CFifo m_pipe_tx_data;
unsigned char m_pipe_buffer_rx_data[MKU_PIPE_TX_SIZE*sizeof(pipe_t)];
CFifo m_pipe_rx_data;

typedef struct preambula_t {
    unsigned iv[4];
    unsigned short crypto_num;
    unsigned short key_num;
    unsigned short time;
    unsigned char type;
    unsigned icv;
    unsigned crc;
    void CalcCRC(){
        crc = calc_csum((unsigned char*)iv, sizeof(preambula_t)-sizeof(crc));
    }
    bool CheckCRC(){
        unsigned crc32 = calc_csum((unsigned char*)iv,
        sizeof(preambula_t)-sizeof(crc));
        if (crc32 == crc){
            return true;
        }
        return false;
    }
    void SaveTime(){
        TDateTime DateTime = Time().CurrentDateTime();
        time =
        DateTime.FormatString("nn").ToInt(); // minutes
        time |= (DateTime.FormatString("hh").ToInt() <<
6); // hour
        time |= (DateTime.FormatString("dd").ToInt() <<
11); // day
    }
    unsigned GetTimeMinutes(){
        unsigned char nn, hh, dd;
        nn = time & 0x3f; // minutes
        hh = (time >> 6) & 0x1f; // hour
        dd = (time >> 11) & 0x1f; // day
        return ((dd*24)+(hh*60)+nn);
    }
    bool isBorderMinutes(unsigned minutes){
        TDateTime DateTime = Time().CurrentDateTime();
        unsigned nn =
        DateTime.FormatString("nn").ToInt(); // minutes
        unsigned hh =
        DateTime.FormatString("hh").ToInt(); // hour
        unsigned dd =
        DateTime.FormatString("dd").ToInt(); // day
        unsigned curr = ((dd*24)+(hh*60)+nn);

```

```

        unsigned inpkt = GetTimeMinutes();
        unsigned inpkt_min = inpkt > minutes ? inpkt-minutes : 0;
        unsigned inpkt_max = inpkt + minutes;
        if (curr >= inpkt_min && curr <= inpkt_max) {
            return true;
        }
        return false;
    }
    unsigned char* Data(){
        return ((unsigned char*)&crc)+sizeof(crc);
    }
}preambula_t;

#define MSKZI_PACKET_PREAM_SIZE    sizeof(preambula_t)
#define MSKZI_PACKET_SIZE          96

typedef struct skzi_t {
    preambula_t pream;
    unsigned char data[MSKZI_PACKET_SIZE];
}skzi_t;

unsigned char tmp_buffer[2048];

bool is_pp_busy_rx;
bool is_mmar_busy_rx;
bool from_pp_busy_rx;
bool from_mmar_busy_rx;

bool is_init = false;

// OT MMAP
enum {
    SKZI_CMD_BEGIN          = 0x01,
    SKZI_CMD_TXD            = 0x03,
    SKZI_CMD_RXD_RES        = 0x08
};
AnsiString mmar_mc_cmd[] = {
    "Undefine"              ,
    "MSKZI_CMD_BEGIN"       ,
    "Undefine"              ,
    "MSKZI_CMD_TXD"         ,
    "Undefine"              ,
    "Undefine"              ,
    "Undefine"              ,
    "Undefine"              ,
    "MSKZI_CMD_RXD_RES"
};
// B MMAP
enum {
    SKZI_CMD_BEGIN_RES      = 0x01,
    SKZI_CMD_SEND_RES       = 0x03,
    SKZI_CMD_RXD            = 0x08
};
AnsiString mmar_cm_cmd[] = {
    "Undefine"              ,
    "MSKZI_CMD_BEGIN_RES"   ,

```

```

        "Undefine"
        "MSKZI_CMD_TXD_RES"
        "Undefine"
        "Undefine"
        "Undefine"
        "Undefine"
        "MSKZI_CMD_RXD"
};

unsigned char skzi_format[] = {
    SKZI_CMD_BEGIN
    SKZI_CMD_TXD
    SKZI_CMD_RXD_RES
};

// от III
enum {
    PP_CMD_BEGIN = 0x01,
    PP_CMD_RXD_PKT = 0x03,
    PP_CMD_TXD_PKT_RES = 0x04
};

AnsiString pp_cm_cmd[] = {
    "Undefine"
    "PP_CMD_BEGIN"
    "Undefine"
    "PP_CMD_RXD_PKT"
    "PP_CMD_TXD_PKT_RES"
    "Undefine"
    "Undefine"
    "Undefine"
    "Undefine"
};

// в III
enum {
    PP_CMD_BEGIN_RES = 0x01,
    PP_CMD_RXD_PKT_RES = 0x03,
    PP_CMD_TXD_PKT = 0x04
};

AnsiString pp_mc_cmd[] = {
    "Undefine"
    "PP_CMD_BEGIN_RES"
    "Undefine"
    "PP_CMD_RXD_PKT_RES"
    "PP_CMD_TXD_PKT"
    "Undefine"
    "Undefine"
    "Undefine"
    "Undefine"
};

unsigned char pp_format[] = {
    PP_CMD_BEGIN
    PP_CMD_RXD_PKT

```

```

        PP_CMD_TXD_PKT_RES
};

enum {
    IMAGE_TXD          = 0x00,
    IMAGE_RXD          = 0x01,
    IMAGE_RXD_FAIL     = 0x02,
    IMAGE_RXD_FAIL2    = 0x03
};
CDte::dte_sign_format_t  pp_sign_format          = {CDte::DTE_SIGN_MZI_PP,
pp_format, sizeof(pp_format)};
CDte::dte_sign_format_t  mmар_sign_format        = {CDte::DTE_SIGN_MMAR_MZI,
skzi_format, sizeof(skzi_format)};

CDte::dte_sign_format_t* zi_mmар_sign_format[] = {
    &mmар_sign_format
};
CDte::dte_sign_format_t* mmар_pp_sign_format[] = {
    &pp_sign_format
};

__fastcall TMainForm::TMainForm(TComponent* Owner)
    : TForm(Owner)
{
}

bool WeAreAlone(LPSTR szName)
{
    HANDLE hMutex=CreateMutex(NULL, TRUE, szName);
    if (GetLastError()==ERROR_ALREADY_EXISTS){
        CloseHandle(hMutex);
        return false;
    }
    return true;
}

AnsiString TMainForm::getMMARInCmdLog(unsigned char format)
{
    return mmар_mc_cmd[format];
}

AnsiString TMainForm::getMMAROutCmdLog(unsigned char format)
{
    return mmар_cm_cmd[format];
}

AnsiString TMainForm::getPPInCmdLog(unsigned char format)
{
    return pp_cm_cmd[format];
}

AnsiString TMainForm::getPPOutCmdLog(unsigned char format)
{
    return pp_mc_cmd[format];
}

void __fastcall TMainForm::FormCreate(TObject *Sender)
{
    if (!WeAreAlone("MyMZI")){
        Application->MessageBox("Программа уже запущена",
"Предупреждение", MB_OK);
    }
}

```

```

        exit(0);
    }
    MMAR_MZI_SOCKET = UDPServerMMAR->Bindings->Add();
    MZI_PP_SOCKET = UDPServerPP->Bindings->Add();

    UDPClientPP->Port = PORT_MSKZI_PP_RX;
    UDPClientMMAR->Port = PORT_MSKZI_MMAR_RX;

    InitSockets();

    m_fifo_tx_pp.Init("MZI_FIFO_TX", m_fifo_buffer_tx_pp,
MKU_FIFO_TX_SIZE);
    m_fifo_rx_pp.Init("MZI_FIFO_RX", m_fifo_buffer_rx_pp,
MKU_FIFO_RX_SIZE);
    m_pipe_tx_pp.Init("MZI_PIPE_TX", m_pipe_buffer_tx_pp,
MKU_PIPE_TX_SIZE*sizeof(pipe_t));
    hw_fifo.Add(&m_fifo_tx_pp);
    hw_fifo.Add(&m_fifo_rx_pp);
    hw_fifo.Add(&m_pipe_tx_pp);
    m_dte_pp.Init(&m_fifo_tx_pp, &m_fifo_rx_pp, MKU_TIME_WAIT_ACK,
MKU_TXD_MAX_REPEAT, &mmar_pp_sign_format[0], 1, m_log_buffer_pp,
MKU_LOG_SIZE);
    hw_dte.Add(&m_dte_pp);

    m_fifo_tx_mmar.Init("MMAR_FIFO_TX", m_fifo_buffer_tx_mmar,
MKU_FIFO_TX_SIZE);
    m_fifo_rx_mmar.Init("MMAR_FIFO_RX", m_fifo_buffer_rx_mmar,
MKU_FIFO_RX_SIZE);
    m_pipe_tx_mmar.Init("MMAR_PIPE_TX", m_pipe_buffer_tx_mmar,
MKU_PIPE_TX_SIZE*sizeof(pipe_t));
    hw_fifo.Add(&m_fifo_tx_mmar);
    hw_fifo.Add(&m_fifo_rx_mmar);
    hw_fifo.Add(&m_pipe_tx_mmar);
    m_dte_mmar.Init(&m_fifo_tx_mmar, &m_fifo_rx_mmar,
MKU_TIME_WAIT_ACK, MKU_TXD_MAX_REPEAT, &zi_mmar_sign_format[0], 1,
m_log_buffer_mmar, MKU_LOG_SIZE);
    hw_dte.Add(&m_dte_mmar);

    m_pipe_tx_data.Init("MKI_PIPE_TX_DATA", m_pipe_buffer_tx_data,
MKU_PIPE_TX_SIZE*sizeof(pipe_t));
    hw_fifo.Add(&m_pipe_tx_data);
    m_pipe_rx_data.Init("MKI_PIPE_RX_DATA", m_pipe_buffer_rx_data,
MKU_PIPE_RX_SIZE*sizeof(pipe_t));
    hw_fifo.Add(&m_pipe_rx_data);

    is_pp_busy_rx = false;
    is_mmar_busy_rx = false;
    from_pp_busy_rx = false;
    from_mmar_busy_rx = false;

    is_init = true;
    CryptoInit();
}
//-----
void TMainForm::SendToMMAR(unsigned char* data, unsigned size)
{

```

```

        UDPClientMMAR->SendBuffer(data, size);
    }
    void TMainForm::SendToPP(unsigned char* data, unsigned size)
    {
        UDPClientPP->SendBuffer(data, size);
    }
    void TMainForm::InitSockets()
    {
        UDPServerPP->Active = false;
        UDPServerMMAR->Active = false;
        UDPClientPP->Host = IP_PP->Text;
        UDPClientMMAR->Host = IP_MMAR->Text;
        MMAR_MZI_SOCKET->Port = PORT_MMAR_MSKZI_RX;
        MZI_PP_SOCKET->Port = PORT_PP_MSKZI_RX;
        try {
            UDPServerPP->Active = true;
            UDPServerMMAR->Active = true;
        }
        catch (...) {
            Application->MessageBox("Проверьте правильность
порта", "Ошибка сокета", MB_OK);
            return;
        }
    }
    void __fastcall TMainForm::TimerTimer(TObject *Sender)
    {
        hw_dte.TimerAction();
    }
    //-----
    void __fastcall TMainForm::UDPServerPPUDPRead(TObject *Sender,
        TStream *AData, TIdSocketHandle *ABinding)
    {
        unsigned s = AData->Size;
        unsigned char* buf = (unsigned char*)malloc(s);
        AData->Read(buf, s);
        AnsiString t;
        if (m_fifo_rx_pp.Free() >= s) {
            m_fifo_rx_pp.Write(buf, s);
        }
        if (LogCheckBox->Checked){
            for (unsigned i=0; i<s; i++){
                t += IntToHex(buf[i], 2) + " ";
            }
            TListItem* item = LogView->Items->Add();
            item->ImageIndex = 1;
            item->Caption = FormatDateTime("hh:nn:ss", Time());
            item->SubItems->Add("");
            item->SubItems->Add(t);
            LogView->Scroll(item->GetPosition().x, item-
>GetPosition().y);
        }
        free(buf);
        hw_dte.Idle();
    }

    void TMainForm::CreateHeaderPP(CDte::dte_header_t* header, unsigned char

```

```

format)
{
    header->sign = CDte::DTE_SIGN_MZI_PP;
    header->type = CDte::DTE_TYPE_CMD_OUT;
    header->format = format;
    header->result = 0;
}
void TMainForm::PipeToPP()
{
    if (m_pipe_tx_pp.Count()){
        pipe_t* pipe = (pipe_t*)m_pipe_tx_pp.GetReadPtr();
        if (m_dte_pp.SendData(pipe->Buffer(), pipe->size)){
            free(pipe->Buffer());
            m_pipe_tx_pp.ReadEmpty(sizeof(pipe_t));
        }
    }
}
void TMainForm::PutPipeToPP(pipe_t* pipe)
{
    if (m_pipe_tx_pp.Free() >= sizeof(pipe_t)){
        m_pipe_tx_pp.Write((unsigned char*)pipe, sizeof(pipe_t));
    } else {
        while (1);
    }
    PipeToPP();
}
void TMainForm::CreatePipePP(unsigned char cmd, unsigned char* buf,
unsigned size)
{
    pipe_t pipe;
    pipe.size = sizeof(CDte::dte_header_t) + size;
    pipe.buff = (unsigned)malloc(pipe.size);
    if (pipe.buff == NULL){
        while (1);
    }
    CDte::dte_header_t* header = (CDte::dte_header_t*)pipe.Buffer();
    CreateHeaderPP(header, cmd);
    if (size){
        memcpy(header->Data(), buf, size);
    }
    if (LogCheckBox->Checked){
        TListItem* item = LogView->Items->Add();
        item->ImageIndex = 0;
        item->Caption = FormatDateTime("hh:nn:ss",Time());
        item->SubItems->Add(getPPOutCmdLog(header->format));
        item->SubItems->Add("");
        item->SubItems->Add(getDataLog((unsigned
char*)pipe.Buffer(), pipe.size));
        LogView->Scroll(item->GetPosition().x, item-
>GetPosition().y);
    }
    PutPipeToPP(&pipe);
}
//-----
void TMainForm::CreateHeaderMMAR(CDte::dte_header_t* header, unsigned
char format)

```

```

{
    header->sign = CDte::DTE_SIGN_MMAR_MZI;
    header->type = CDte::DTE_TYPE_CMD_OUT;
    header->format = format;
    header->result = 0;
}
void TMainForm::PipeToMMAR()
{
    if (m_pipe_tx_mmar.Count()){
        pipe_t* pipe = (pipe_t*)m_pipe_tx_mmar.GetReadPtr();
        if (m_dte_mmar.SendData(pipe->Buffer(), pipe->size)){
            free(pipe->Buffer());
            m_pipe_tx_mmar.ReadEmpty(sizeof(pipe_t));
        }
    }
}
void TMainForm::PutPipeToMMAR(pipe_t* pipe)
{
    if (m_pipe_tx_mmar.Free() >= sizeof(pipe_t)){
        m_pipe_tx_mmar.Write((unsigned char*)pipe,
sizeof(pipe_t));
    } else {
        while (1);
    }
    PipeToMMAR();
}
void TMainForm::CreatePipeMMAR(unsigned char cmd, unsigned char* buf,
unsigned size)
{
    pipe_t pipe;
    pipe.size = sizeof(CDte::dte_header_t) + size;
    pipe.buff = (unsigned)malloc(pipe.size);
    if (pipe.buff == NULL){
        while (1);
    }
    CDte::dte_header_t* header = (CDte::dte_header_t*)pipe.Buffer();
    CreateHeaderMMAR(header, cmd);
    if (size){
        memcpy(header->Data(), buf, size);
    }
    if (LogCheckBox->Checked){
        TListItem* item = LogView->Items->Add();
        item->ImageIndex = 0;
        item->Caption = FormatDateTime("hh:nn:ss",Time());
        item->SubItems->Add(getMMAROutCmdLog(header->format));
        item->SubItems->Add("");
        item->SubItems->Add(getDataLog((unsigned
char*)pipe.Buffer(), pipe.size));
        LogView->Scroll(item->GetPosition().x, item-
>GetPosition().y);
    }
    PutPipeToMMAR(&pipe);
}
//-----
AnsiString TMainForm::getDataLog(unsigned char* data, unsigned size)
{

```

```

        AnsiString text;
        for (unsigned i=0; i<size; i++){
            text += IntToHex(data[i], 2) + " ";
        }
        return text;
    }
}
void TMainForm::PipeTxDataToPP()
{
    if (!is_pp_busy_rx && m_pipe_tx_data.Count()){
        pipe_t* pipe = (pipe_t*)m_pipe_tx_data.GetReadPtr();
        if (m_dte_pp.SendData(pipe->Buffer(), pipe->size)){
            free(pipe->Buffer());
            m_pipe_tx_data.ReadEmpty(sizeof(pipe_t));
            is_pp_busy_rx = true;
            PPTXDLabel->Caption =
IntToStr(StrToInt(PPTXDLabel->Caption)+1);
        }
    }
}
bool TMainForm::CreatePipeTxData(unsigned char* buf, unsigned size)
{
    pipe_t pipe;
    pipe.size = sizeof(CDte::dte_header_t) + size;
    pipe.buff = (unsigned)malloc(pipe.size);
    if (pipe.buff == NULL){
        while (1);
    }
    CDte::dte_header_t* header = (CDte::dte_header_t*)pipe.Buffer();
    CreateHeaderPP(header, PP_CMD_TXD_PKT);
    if (size){
        memcpy(header->Data(), buf, size);
    }
    if (m_pipe_tx_data.Free() >= sizeof(pipe_t)){
        m_pipe_tx_data.Write((unsigned char*)&pipe,
sizeof(pipe_t));
    } else {
        while (1);
    }
    if (LogCheckBox->Checked){
        TListItem* item = LogView->Items->Add();
        item->ImageIndex = 0;
        item->Caption = FormatDateTime("hh:nn:ss",Time());
        item->SubItems->Add(getPPOutCmdLog(header->format));
        item->SubItems->Add("");
        item->SubItems->Add(getDataLog((unsigned
char*)pipe.Buffer(), pipe.size));
        LogView->Scroll(item->GetPosition().x, item-
>GetPosition().y);
    }
    if (m_pipe_tx_data.Free() >= sizeof(pipe_t)){
        return true;
    }
    return false;
}

void TMainForm::PipeRxDataToMMAR()

```

```

{
    if (!is_mmar_busy_rx && m_pipe_rx_data.Count()){
        pipe_t* pipe = (pipe_t*)m_pipe_rx_data.GetReadPtr();
        if (m_dte_mmar.SendData(pipe->Buffer(), pipe->size)){
            free(pipe->Buffer());
            m_pipe_rx_data.ReadEmpty(sizeof(pipe_t));
            is_mmar_busy_rx = true;
            MMARTXDLabel->Caption =
IntToStr(StrToInt(MMARTXDLabel->Caption)+1);
        }
    }
}

bool TMainForm::CreatePipeRxData(unsigned char* buf, unsigned size)
{
    pipe_t pipe;
    pipe.size = sizeof(CDte::dte_header_t) + size;
    pipe.buff = (unsigned)malloc(pipe.size);
    if (pipe.buff == NULL){
        while (1);
    }
    CDte::dte_header_t* header = (CDte::dte_header_t*)pipe.Buffer();
    CreateHeaderMMAR(header, SKZI_CMD_RXD);
    if (size){
        memcpy(header->Data(), buf, size);
    }
    if (m_pipe_rx_data.Free() >= sizeof(pipe_t)){
        m_pipe_rx_data.Write((unsigned char*)&pipe,
sizeof(pipe_t));
    } else {
        while (1);
    }
    if (LogCheckBox->Checked){
        TListItem* item = LogView->Items->Add();
        item->ImageIndex = 0;
        item->Caption = FormatDateTime("hh:nn:ss",Time());
        item->SubItems->Add(getMMAROutCmdLog(header->format));
        item->SubItems->Add("");
        item->SubItems->Add(getDataLog((unsigned
char*)pipe.Buffer(), pipe.size));
        LogView->Scroll(item->GetPosition().x, item-
>GetPosition().y);
    }
    if (m_pipe_rx_data.Free() >= sizeof(pipe_t)){
        return true;
    }
    return false;
}

void TMainForm::CmdAction(CDte::dte_header_t* header, unsigned size)
{
    unsigned s = size - sizeof(CDte::dte_header_t);
    skzi_t* skzi_pkt_rx = (skzi_t*)header->Data();
    skzi_t skzi_pkt_tx;
    if (header->sign == CDte::DTE_SIGN_MZI_PP){
        switch (header->format){
            case PP_CMD_BEGIN:
                CreatePipePP(PP_CMD_BEGIN_RES, NULL, 0);

```

```

        break;
        case PP_CMD_RXD_PKT:
            // decode
            // прием из ПП, расшифрование
            if (skzi_pkt_rx->pream.CheckCRC()){
if (skzi_pkt_rx->pream.isBorderMinutes(RXD_BORDER_MINUTES)){
decrypt_packet((unsigned*)skzi_pkt_rx->data, (unsigned*)&skzi_pkt_rx-
>pream, (unsigned*)ki, MSKZI_PACKET_SIZE/4, false);
if (CreatePipeRxData(skzi_pkt_rx->data, MSKZI_PACKET_SIZE)){
PipeRxDataToMMAR();
CreatePipePP(PP_CMD_RXD_PKT_RES, NULL, 0);
} else {
from_pp_busy_rx = true;
}
} else {
CreatePipePP(PP_CMD_RXD_PKT_RES, NULL, 0);
}
} else {
CreatePipePP(PP_CMD_RXD_PKT_RES, NULL, 0);
}
PPRXDLabel->Caption = IntToStr(StrToInt(PPRXDLabel->Caption)+1);
        break;
        case PP_CMD_TXD_PKT_RES:
            is_pp_busy_rx = false;
            PipeTxDataToPP();
            break;
        default:
            break;
    };
}
if (header->sign == CDte::DTE_SIGN_MMAR_MZI){
    switch (header->format){
        case SKZI_CMD_BEGIN:
            CreatePipeMMAR(SKZI_CMD_BEGIN_RES, NULL, 0);
            break;
        case SKZI_CMD_TXD:
            MMARIdle();
            MMARRXDLabel->Caption = IntToStr(StrToInt(MMARRXDLabel-
>Caption)+1);
            memcpy(skzi_pkt_tx.data, header->Data(), s);
            // encode
            // шифрование, передача в ПП
            encrypt_packet((unsigned*)&skzi_pkt_tx.data,
(unsigned*)&skzi_pkt_tx.pream, (unsigned*)ki, MSKZI_PACKET_SIZE/4,
false);
            skzi_pkt_tx.pream.SaveTime();
            skzi_pkt_tx.pream.CalcCRC();
            if (CreatePipeTxData((unsigned char*)&skzi_pkt_tx,
sizeof(skzi_pkt_tx))){
                PipeTxDataToPP();
                CreatePipeMMAR(SKZI_CMD_SEND_RES, NULL, 0);
            } else {
                from_mmar_busy_rx = true;
            }
            break;
        case SKZI_CMD_RXD_RES:

```

```

        is_mmar_busy_rx = false;
        PipeRxDataToMMAR();
        break;
    default:
        break;
    };
}

}

void TMainForm::PPIdle()
{
    unsigned char *buffer;
    if (from_pp_busy_rx && m_pipe_rx_data.Free() >= sizeof(pipe_t)){
        CreatePipePP(PP_CMD_RXD_PKT_RES, NULL, 0);
        from_pp_busy_rx = false;
    }
    PipeTxDataToPP();
    unsigned size = m_fifo_tx_pp.Count();
    if (size){
        buffer = (unsigned char*)malloc(size);
        if (buffer){
            m_fifo_tx_pp.Read(buffer, size);
            SendToPP(buffer, size);
            if (LogCheckBox->Checked){
                TListItem* item = LogView->Items->Add();
                item->ImageIndex = 0;
                item->Caption = FormatDateTime("hh:nn:ss",Time());
                item->SubItems->Add("");
                item->SubItems->Add(getDataLog(buffer, size));
                LogView->Scroll(item->GetPosition().x, item->GetPosition().y);
            }
        }
    }

    size = m_dte_pp.Receive(tmp_buffer);
    if (size >= sizeof(tmp_buffer)){
        while (1);
    }

    if (size){
        CDte::dte_header_t* header =
(CDte::dte_header_t*)tmp_buffer;
        if (LogCheckBox->Checked){
            TListItem* item = LogView->Items->Add();
            item->ImageIndex = 1;
            item->Caption =
FormatDateTime("hh:nn:ss",Time());
            item->SubItems->Add(getPPInCmdLog(header-
>format));
            item->SubItems->Add("");
            item->SubItems->Add(getDataLog((unsigned
char*)header, size));
            LogView->Scroll(item->GetPosition().x, item-
>GetPosition().y);
        }
        CmdAction(header, size);
    }
    PipeToPP();
}

```

```

void TMainForm::MMARIdle()
{
    unsigned char *buffer;
    if (from_mmar_busy_rx && m_pipe_tx_data.Free() >= sizeof(pipe_t))
    {
        CreatePipeMMAR(SKZI_CMD_SEND_RES, NULL, 0);
        from_mmar_busy_rx = false;
    }
    PipeRxDataToMMAR();
    unsigned size = m_fifo_tx_mmar.Count();
    if (size){
        buffer = (unsigned char*)malloc(size);
        if (buffer){
            m_fifo_tx_mmar.Read(buffer, size);
            SendToMMAR(buffer, size);
            if (LogCheckBox->Checked){
                TListItem* item = LogView->Items->Add();
                item->ImageIndex = 0;
                item->Caption =
FormatDateTime("hh:nn:ss",Time());
                item->SubItems->Add("");
            }
            item->SubItems->Add(getDataLog(buffer, size));
            LogView->Scroll(item->GetPosition().x, item->GetPosition().y);
        }
    }
    size = m_dte_mmar.Receive(tmp_buffer);
    if (size >= sizeof(tmp_buffer)){
        while (1);
    }
    if (size){
        CDte::dte_header_t* header = (CDte::dte_header_t*)tmp_buffer;
        if (LogCheckBox->Checked){
            TListItem* item = LogView->Items->Add();
            item->ImageIndex = 1;
            item->Caption = FormatDateTime("hh:nn:ss",Time());
            item->SubItems->Add(getMMARInCmdLog(header->format));
            item->SubItems->Add("");
            item->SubItems->Add(getDataLog((unsigned
char*)header, size));
            LogView->Scroll(item->GetPosition().x, item-
>GetPosition().y);
        }
        CmdAction(header, size);
    }
    PipeToMMAR();
}
void __fastcall TMainForm::WaitTimerTimer(TObject *Sender)
{
    if (!is_init){
        return;
    }
    PPIIdle();
    MMARIdle();
    hw_dte.Idle();
}

```

```

//-----
void __fastcall TMainForm::N3Click(TObject *Sender)
{
    LogView->Clear();
    MMARRXDLabel->Caption = "0";
    PPTXDLabel->Caption = "0";
    PPRXDLabel->Caption = "0";
    MMARTXDLabel->Caption = "0";
}
//-----
void __fastcall TMainForm::UDPServerMMARUDPRead(TObject *Sender,
    TStream *AData, TIdSocketHandle *ABinding)
{
    unsigned s = AData->Size;
    unsigned char* buf = (unsigned char*)malloc(s);
    AData->Read(buf, s);
    AnsiString t;
    if (m_fifo_rx_mmar.Free() >= s) {
        m_fifo_rx_mmar.Write(buf, s);
    }
    if (LogCheckBox->Checked){
        for (unsigned i=0; i<s; i++){
            t += IntToHex(buf[i], 2) + " ";
        }
        TListItem* item = LogView->Items->Add();
        item->ImageIndex = 1;
        item->Caption = FormatDateTime("hh:nn:ss", Time());
        item->SubItems->Add("");
        item->SubItems->Add(t);
        LogView->Scroll(item->GetPosition().x, item-
>GetPosition().y);
    }
    free(buf);
    hw_dte.Idle();
}
//-----
void __fastcall TMainForm::Button1Click(TObject *Sender)
{
    InitSockets();
}

```

Файл «MSKZI.cpp»

```

#include <vcl.h>
#pragma hdrstop
//-----
USEFORM("main.cpp", MainForm);
//-----
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
    try
    {
        Application->Initialize();
        Application->CreateForm(__classid(TMainForm),
&MainForm);
        Application->Run();
    }
}

```

```

    }
    catch (Exception &exception)
    {
        Application->ShowException(&exception);
    }
    catch (...)
    {
        try
        {
            throw Exception("");
        }
        catch (Exception &exception)
        {
            Application->ShowException(&exception);
        }
    }
    return 0;
}

```

Файл «magma.cpp»

```

#include <memory.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <conio.h>
typedef uint8_t  u8;
typedef uint16_t u16;
typedef uint32_t u32;
typedef uint64_t u64;
static unsigned char const k1[16] = {12, 4, 6, 2, 10, 5, 11, 9,
14, 8, 13, 7, 0, 3, 15, 1};
static unsigned char const k2[16] = {6, 8, 2, 3, 9, 10, 5, 12, 1,
14, 4, 7, 11, 13, 0, 15};
static unsigned char const k3[16] = {11, 3, 5, 8, 2, 15, 10, 13,
14, 1, 7, 4, 12, 9, 6, 0};
static unsigned char const k4[16] = {12, 8, 2, 1, 13, 4, 15, 6, 7,
0, 10, 5, 3, 14, 9, 11};
static unsigned char const k5[16] = {7, 15, 5, 10, 8, 1, 6, 13, 0,
9, 3, 14, 11, 4, 2, 12};
static unsigned char const k6[16] = {5, 13, 15, 6, 9, 2, 12, 10,
11, 7, 8, 1, 4, 3, 14, 0};
static unsigned char const k7[16] = {8, 14, 2, 5, 6, 9, 1, 12, 15,
4, 11, 0, 13, 10, 3, 7};
static unsigned char const k8[16] = {1, 7, 14, 13, 0, 5, 8, 3, 4,
15, 10, 6, 9, 12, 11, 2};

static unsigned char k87[256];
static unsigned char k65[256];
static unsigned char k43[256];
static unsigned char k21[256];

void kboxinit(void)

```

```

{
    int i;
    for (i = 0; i < 256; i++) {
        k87[i] = k8[i >> 4] << 4 | k7[i & 15];
        k65[i] = k6[i >> 4] << 4 | k5[i & 15];
        k43[i] = k4[i >> 4] << 4 | k3[i & 15];
        k21[i] = k2[i >> 4] << 4 | k1[i & 15];
    }
}
static u32 f(u32 x)
{
    x = k87[x>>24 & 255] << 24 | k65[x>>16 & 255] << 16 |
        k43[x>> 8 & 255] << 8 | k21[x & 255];

    return x<<11 | x>>(32-11);
}
void gostcrypt(u32 const in[2], u32 out[2], u32 const key[8])
{
    uint8_t i;
    register u32 n1, n2;
    n1 = in[0];
    n2 = in[1];
    for (i=0; i<23; i+=2)
    {
        n2 ^= f(n1+key[i % 8]);
        n1 ^= f(n2+key[(i+1)%8]);
    }
    for (i=24; i<31; i+=2)
    {
        n2 ^= f(n1+key[31-i]);
        n1 ^= f(n2+key[31-(i+1)]);
    }
    out[0] = n2;
    out[1] = n1;
}
void gostdecrypt(u32 const in[2], u32 out[2], u32 const key[8])
{
    uint8_t i;
    register u32 n1, n2;
    n1 = in[0];
    n2 = in[1];
    for (i=0; i<7; i+=2)
    {
        n2 ^= f(n1+key[i]);
        n1 ^= f(n2+key[i+1]);
    }
    for (i=8; i<31; i+=2)
    {
        n2 ^= f(n1+key[(31-i)%8]);
        n1 ^= f(n2+key[(31-(i+1))%8]);
    }
}

```

```

    }
    out[0] = n2;
    out[1] = n1;
}
void gostcnt(u32 const *in, u32 *out, int len, u32 iv[2], u32
const key[8])
{
    u32 temp[2];
    u32 gamma[2];
    uint64_t value;
    gostcrypt(iv, temp, key);
    len /= 2;
    while (len--)
    {
        *out++ = *in++ ^ temp[0];
        *out++ = *in++ ^ temp[1];
        iv[0]++;
        if (iv[0] == 0)
        {
            iv[1]++;
        }
    }
}

}
void gostmac(u32 const *in, int len, u32 out[2], u32 const key[8])
{
    register u32 n1, n2;
    n1 = 0;
    n2 = 0;

    while (len--) {
        n1 ^= *in++;
        n2 ^= *in++;

        n2 ^= f(n1+key[0]);
        n1 ^= f(n2+key[1]);
        n2 ^= f(n1+key[2]);
        n1 ^= f(n2+key[3]);
        n2 ^= f(n1+key[4]);
        n1 ^= f(n2+key[5]);
        n2 ^= f(n1+key[6]);
        n1 ^= f(n2+key[7]);

        n2 ^= f(n1+key[0]);
        n1 ^= f(n2+key[1]);
        n2 ^= f(n1+key[2]);
        n1 ^= f(n2+key[3]);
        n2 ^= f(n1+key[4]);
        n1 ^= f(n2+key[5]);
        n2 ^= f(n1+key[6]);
    }
}

```

```

        n1 ^= f(n2+key[7]);
    }
    out[0] = n1;
    out[1] = n2;
}

int main()
{
    #define BUFF1 2
    #define BUFF2 2
    #define BUFF3 2
    #define FILE_PATH 32
    char file_path[FILE_PATH];
    u32 buf1[BUFF1];
    u32 buf2[BUFF2];
    u32 mac[BUFF3];
    u32 key[8] =
    {
        {0xffeeddcc},
        {0xbbaa9988},
        {0x77665544},
        {0x33221100},
        {0xf0f1f2f3},
        {0xf4f5f6f7},
        {0xf8f9fafb},
        {0xfcfdfeff}
    };

    kbboxinit();
    u32 synh[2]={0x00000000, 0x12345678};//синхропосылка
    int k;
    printf("GOST 34.13-2015.CTR, n-64\n");
    FILE *f = fopen("opentext.bin", "rb");
    FILE *res = fopen("ciphertext.bin", "wb+");
    int size;
    fseek(f, 0L, SEEK_END);
    size = ftell(f)/4;
    fseek(f, 0L, SEEK_SET);

    if ((f!=NULL)&&(res!=NULL))
    {
        while (1)
        {
            k = fread (buf1, 1, sizeof(buf1), f);
            if (k == 0)
            {
                break;
            }
            gostcnt(buf1, buf2, sizeof(buf1)/4, synh, key);
            fwrite(buf2, 1, sizeof(buf2), res);
            gostmac(buf1, sizeof(mac)/4, mac, key);
        }
    }
}

```

```

    }

    fwrite(&mac[0], 1, sizeof(u32), res);

    printf("Encryption complete\n");
    //printf("mac = %x%x\n", mac[0], mac[1]);

    fclose(f);
    fclose(res);

}
else
{
    if (f == NULL)
        printf("Failed to opentext file");
    if (res == NULL)
        printf("Failed to write ciphertext");
}

/* TEST
//проверка t преобразования
    u32 t = {0x2a196f34}; //0xfdb97531
    f(t);
//проверка циклов зашифрования, расшифрования
    u32 plain[2]={0x76543210, 0xfedcba98};
    u32 cipher[2];
    gostcrypt(plain, cipher, key);

    u32 b[2]={0xc2d8ca3d, 0x4ee901e5};
    u32 opent[2];
    gostdecrypt(b, opent, key);
    //u32 opentext[2]={0x3c130a59, 0x92def06b};
    //u32 ciphertext[2];
    //u32 mac[2];
    //gostcnt(opentext, ciphertext, 2, synh, key);*/
    while(!kbhit());
    return 0;
}

```

Приложение Е

(обязательное)

Исходный текст программного обеспечения интерфейсного модуля

Файл «main.cpp»

```
#include <vcl.h>
#pragma hdrstop
#include "main.h"
#include "progress.h"
#include "rpu.h"
//-----
#pragma package(smart_init)
#pragma link "VaClasses"
#pragma link "VaComm"
#pragma link "CGAUGES"
#pragma resource "*.dfm"
#define PORT_MKI_MSH_RX          4011
#define PORT_MSH_MKI_RX          4111
#define PORT_MSH_MMAR_RX         4122
#define PORT_MMAR_MSH_RX         4022
#define PORT_MMAR_MSKZI_RX       4033
#define PORT_MSKZI_MMAR_RX       4133
#define PORT_MSKZI_PP_RX         4144
#define PORT_PP_MSKZI_RX         4044

TMainForm *MainForm;
TIdSocketHandle* MSH_MKI_SOCKET;

TProgressForm* ProgressFormTx = NULL;
TProgressForm* ProgressFormRx = NULL;

#define MKU_FIFO_TX_SIZE          2048
#define MKU_FIFO_RX_SIZE          2048
#define MKU_PIPE_TX_SIZE          30
#define MKU_TIME_WAIT_ACK         2
#define MKU_TXD_MAX_REPEATS       2
#define MKU_LOG_SIZE              (100*sizeof(CDte::dte_header_t))

unsigned char m_fifo_buffer_tx[MKU_FIFO_TX_SIZE];
CFifo m_fifo_tx;
unsigned char m_fifo_buffer_rx[MKU_FIFO_RX_SIZE];
CFifo m_fifo_rx;
unsigned char m_pipe_buffer_tx[MKU_PIPE_TX_SIZE*sizeof(pipe_t)];
CFifo m_pipe_tx;
unsigned char m_fifo_buffer_file_read_tx[RPU_SAMPLES_2400*10];
CFifo m_fifo_file_read_tx;
unsigned char m_fifo_buffer_data_tx[DATA_BYTES*1000];
CFifo m_fifo_data_tx;

unsigned char m_fifo_buffer_data_rx[DATA_BYTES*1000];
CFifo m_fifo_data_rx;
```

```

unsigned char m_log_buffer[MKU_LOG_SIZE];
CDte m_dte;
CFifoMenager hw_fifo;
CDteMenager hw_dte;
unsigned char tmp_buffer[2048];

FILE* m_txd_file;
unsigned m_txd_file_size;
unsigned m_txd_file_ptr;

unsigned m_txd_data_size;

FILE* m_rxd_file;
bool is_init = false;
bool is_msh_busy_rx;
bool is_rpu;
bool is_wait_rx_rpu;
#define PKT_HDR_SIZE 8
#define MAX_PKT_DATA_SIZE (SendSizeEdit->Text.ToInt()-PKT_HDR_SIZE)

enum {
    PACKET_TYPE_DATA = 0,
    PACKET_TYPE_RPU = 1
};

typedef struct packet_t {
    unsigned short pkt_num;
    unsigned short pkt_size;
    unsigned short counts;
    unsigned short type;
    unsigned char data[1536-PKT_HDR_SIZE];
}packet_t;

packet_t m_txd_pkt;

enum {
    MKI_CMD_BEGIN           = 0x01,
    MKI_CMD_SEND_PTP        = 0x02,
    MKI_CMD_SEND             = 0x03,
    MKI_CMD_GET_STATE       = 0x05,
    MKI_CMD_END              = 0x09
};

AnsiString cm_cmd[] = {
    "Undefine"              ,
    "MKI_CMD_BEGIN"         ,
    "MKI_CMD_SEND_PTP"      ,
    "MKI_CMD_SEND"          ,
    "Undefine"              ,
    "MKI_CMD_GET_STATE"     ,
    "Undefine"              ,
    "Undefine"              ,
    "Undefine"              ,
    "MKI_CMD_END"           ,
};

```

```

enum {
    MKI_CMD_BEGIN_RES      = 0x01,
    MKI_CMD_SEND_PTP_RES   = 0x02,
    MKI_CMD_SEND_RES       = 0x03,
    MKI_CMD_STATE          = 0x05,
    MKI_CMD_RXD            = 0x06,
    MKI_CMD_RXD_READY      = 0x07,
    MKI_CMD_RXD_BUSY       = 0x08,
    MKI_CMD_END_RES        = 0x09
};

AnsiString mc_cmd[] = {
    "Undefine"              ,
    "MKI_CMD_BEGIN_RES"     ,
    "MKI_CMD_SEND_PTP_RES"  ,
    "MKI_CMD_SEND_RES"      ,
    "Undefine"              ,
    "MKI_CMD_STATE"         ,
    "MKI_CMD_RXD"           ,
    "MKI_CMD_RXD_READY"     ,
    "MKI_CMD_RXD_BUSY"      ,
    "MKI_CMD_END"           ,
};

unsigned char marskzi_format[] = {
    MKI_CMD_BEGIN_RES      ,
    MKI_CMD_SEND_PTP_RES   ,
    MKI_CMD_SEND_RES       ,
    MKI_CMD_STATE          ,
    MKI_CMD_RXD            ,
    MKI_CMD_RXD_READY      ,
    MKI_CMD_RXD_BUSY       ,
    MKI_CMD_END_RES        ,
};

enum {
    IMAGE_TXD              = 0x00,
    IMAGE_RXD              = 0x01,
    IMAGE_RXD_FAIL         = 0x02,
    IMAGE_RXD_FAIL2        = 0x03
};

CDte::dte_sign_format_t marskzi_sign_format = {CDte::DTE_SIGN_MKI_MSH,
marskzi_format, sizeof(marskzi_format)};
CDte::dte_sign_format_t* mku_sign_format[] = {
    &marskzi_sign_format
};

__fastcall TMainForm::TMainForm(TComponent* Owner)
    : TForm(Owner)
{
}

bool WeAreAlone(LPSTR szName)
{
    HANDLE hMutex=CreateMutex(NULL,TRUE,szName);
    if (GetLastError()==ERROR_ALREADY_EXISTS){
        CloseHandle(hMutex);
        return false;
    }
}

```

```

        return true;
    }
    AnsiString TMainForm::getInCmdLog(unsigned char format)
    {
        return mc_cmd[format];
    }
    AnsiString TMainForm::getOutCmdLog(unsigned char format)
    {
        return cm_cmd[format];
    }
    void __fastcall TMainForm::FormCreate(TObject *Sender)
    {
        if (!WeAreAlone("MyMKI")){
            Application->MessageBox("Программа уже запущена",
            "Предупреждение", MB_OK);
            exit(0);
        }
        Left = 0;
        Top = 0;
        MSH_MKI_SOCKET = UDPServer->Bindings->Add();
        InitSockets();
        m_fifo_tx.Init("MKU_FIFO_TX", m_fifo_buffer_tx,
MKU_FIFO_TX_SIZE);
        m_fifo_rx.Init("MKU_FIFO_RX", m_fifo_buffer_rx,
MKU_FIFO_RX_SIZE);
        m_pipe_tx.Init("MKU_PIPE_TX", m_pipe_buffer_tx,
MKU_PIPE_TX_SIZE*sizeof(pipe_t));
        hw_fifo.Add(&m_fifo_tx);
        hw_fifo.Add(&m_fifo_rx);
        hw_fifo.Add(&m_pipe_tx);
        m_dte.Init(&m_fifo_tx, &m_fifo_rx, MKU_TIME_WAIT_ACK,
MKU_TXD_MAX_REPEAT, &mku_sign_format[0], 1, m_log_buffer, MKU_LOG_SIZE);
        hw_dte.Add(&m_dte);
        memset(&m_txd_pkt, 0, sizeof(m_txd_pkt));
        m_fifo_file_read_tx.Init("FIFO_FILE_TX",
m_fifo_buffer_file_read_tx, sizeof(m_fifo_buffer_file_read_tx));
        m_fifo_data_tx.Init("FIFO_DATA_TX", m_fifo_buffer_data_tx,
sizeof(m_fifo_buffer_data_tx));
        m_fifo_data_rx.Init("FIFO_DATA_RX", m_fifo_buffer_data_rx,
sizeof(m_fifo_buffer_data_rx));
        hw_fifo.Add(&m_fifo_file_read_tx);
        hw_fifo.Add(&m_fifo_data_tx);
        hw_fifo.Add(&m_fifo_data_rx);
        RPUInit(RPU_MODE_2400);
        is_init = true;
        is_msh_busy_rx = false;
        is_rpu = false;
        is_wait_rx_rpu = false;
    }
    //-----
    void TMainForm::SendToMSH(unsigned char* data, unsigned size)
    {
        UDPCClient->SendBuffer(data, size);
    }
    void TMainForm::InitSockets()
    {

```

```

        UDPServer->Active = false;
        MSH_MKI_SOCKET->Port = PORT_MSH_MKI_RX;
        try {
            UDPServer->Active = true;
        }
        catch (...) {
            Application->MessageBox("Проверьте правильность
порта", "Ошибка сокета", MB_OK);
            return;
        }
    }
    void __fastcall TMainForm::TimerTimer(TObject *Sender)
    {
        hw_dte.TimerAction();
    }
    //-----
    void __fastcall TMainForm::UDPServerUDPRead(TObject *Sender,
        TStream *AData, TIdSocketHandle *ABinding)
    {
        unsigned s = AData->Size;
        unsigned char* buf = (unsigned char*)malloc(s);
        AData->Read(buf, s);
        AnsiString t;
        if (m_fifo_rx.Free() >= s) {
            m_fifo_rx.Write(buf, s);
        }
        for (unsigned i=0; i<s; i++){
            t += IntToHex(buf[i], 2) + " ";
        }
        TListItem* item = LogView->Items->Add();
        item->ImageIndex = 1;
        item->Caption = FormatDateTime("hh:nn:ss", Time());
        item->SubItems->Add("");
        item->SubItems->Add(t);
        LogView->Scroll(item->GetPosition().x, item->GetPosition().y);
        free(buf);
        hw_dte.Idle();
    }
    //-----
    void TMainForm::ReadFile()
    {
        unsigned cnt = m_fifo_file_read_tx.Free();
        if (m_txd_file_ptr < m_txd_file_size){
            if (m_txd_file && cnt){
                if (m_txd_file_ptr+cnt > m_txd_file_size){
                    cnt = m_txd_file_size-m_txd_file_ptr;
                }
                unsigned char* buf = (unsigned char*)malloc(cnt);
                fread(buf, cnt, 1, m_txd_file);
                m_fifo_file_read_tx.Write(buf, cnt);
                free(buf);
                m_txd_file_ptr += cnt;
            }
        } else {
            if (m_txd_file){
                fclose(m_txd_file);
            }
        }
    }

```

```

        m_txd_file = NULL;
    }
}
short tmp_buffer1[DATA_BYTES];
short rpu_out_buffer[320];
unsigned char tmp_buffer2[sizeof(tmp_buffer)];
void TMainForm::ProcessRxFile()
{
    unsigned s = 0;
    if (is_rpu){
        if (m_fifo_data_rx.Count() >= DATA_BYTES) {
            m_fifo_data_rx.Read((unsigned char*)tmp_buffer1, DATA_BYTES);
            s = RPURx(tmp_buffer1, DATA_BYTES/2, rpu_out_buffer)*2;
            memcpy(tmp_buffer2, rpu_out_buffer, s) ;
        }
    } else {
        s = m_fifo_data_rx.Count();
        if (s > sizeof(tmp_buffer2)){
            s = sizeof(tmp_buffer2);
        }
        m_fifo_data_rx.Read(tmp_buffer2, s);
    }
    if (m_rxd_file && s){
        fwrite(tmp_buffer2, s, 1, m_rxd_file);
    }
    if (is_wait_rx_rpu){
        if (m_rxd_file){
            if (m_fifo_data_rx.Count() < DATA_BYTES){
                AnsiString t = FormatDateTime("hh:nn:ss",Time()) + " Файл принят";
                LogMemo->Lines->Add(t);
                fclose(m_rxd_file);
                m_rxd_file = NULL;
                if (ProgressFormRx){
                    ProgressFormRx->Free();
                    ProgressFormRx = NULL;
                }
                m_fifo_data_rx.Clear();
                is_wait_rx_rpu = false;
            }
        }
    }
}

short rpu_in_buffer[320];
void TMainForm::ProcessTxFile()
{
    unsigned s = 0;
    if (RPUCheckBox->Checked){
        if (m_fifo_file_read_tx.Count() >= RPU_SAMPLES_2400 &&
m_fifo_data_tx.Free() >= DATA_BYTES) {
            m_fifo_file_read_tx.Read((unsigned
char*)rpu_in_buffer, RPU_SAMPLES_2400*2);
            for (unsigned i=0; i<RPU_SAMPLES_2400; i++){
                rpu_in_buffer[i] = ((rpu_in_buffer[i] <<
8) & 0xff00) | ((rpu_in_buffer[i] >> 8) & 0x00ff);
            }
        }
    }
}

```

```

        s = RPUTx(rpu_in_buffer, RPU_SAMPLES_2400,
(short*)tmp_buffer);
        m_fifo_data_tx.Write(tmp_buffer, s*2);
    }
    } else {
        s = m_fifo_data_tx.Free();
        if (m_fifo_file_read_tx.Count() && s){
            if (m_fifo_file_read_tx.Count() < s){
                s = m_fifo_file_read_tx.Count();
            }
            m_fifo_file_read_tx.Read(tmp_buffer, s);
            m_fifo_data_tx.Write(tmp_buffer, s);
        }
    }
}

void TMainForm::SendFile()
{
    if (is_msh_busy_rx == true){
        return;
    }
    if (m_txd_pkt.pkt_num*MAX_PKT_DATA_SIZE > m_txd_data_size){
        m_txd_pkt.pkt_size = m_txd_data_size -
((m_txd_pkt.pkt_num-1)*MAX_PKT_DATA_SIZE);
    } else {
        m_txd_pkt.pkt_size = MAX_PKT_DATA_SIZE;
    }
    if (m_fifo_data_tx.Count() >= m_txd_pkt.pkt_size){
        m_fifo_data_tx.Read(m_txd_pkt.data, m_txd_pkt.pkt_size);
        CreatePipe(MKI_CMD_SEND, (unsigned char*)&m_txd_pkt,
PKT_HDR_SIZE+m_txd_pkt.pkt_size);
        is_msh_busy_rx = true;
    }
}

void __fastcall TMainForm::SendButtonClick(TObject *Sender)
{
    UDPCClient->Host = IP_MSH->Text;
    UDPCClient->Port = PORT_MKI_MSH_RX;
    if (RadioButton1->Checked){
        CreatePipe(MKI_CMD_BEGIN, NULL, 0);
    }
    if (RadioButton2->Checked){
        OpenFileDialog->Filter = "Any files (*.*)|*.*";
        if (OpenFileDialog->Execute()){
            if (m_txd_file){
                fclose(m_txd_file);
                m_txd_file = NULL;
            }
            if (m_rxd_file){
                fclose(m_rxd_file);
                m_rxd_file = NULL;
            }
        }
        if (m_txd_file = fopen(OpenFileDialog->FileName.c_str(), "rb")){
            m_txd_file_ptr = 0;
            m_txd_file_size = FileSizeByName(OpenFileDialog->FileName);
            m_txd_data_size = m_txd_file_size;
            memset(&m_txd_pkt, 0, sizeof(m_txd_pkt));

```

```

        m_txd_pkt.type = PACKET_TYPE_DATA;
        if (RPUCheckBox->Checked){
            m_txd_data_size /= RPU_SAMPLES_2400*2;
            m_txd_data_size *= DATA_BYTES;
            m_txd_pkt.type = PACKET_TYPE_RPU;
        }
        m_txd_pkt.pkt_num = 1;
        m_txd_pkt.counts = m_txd_data_size / MAX_PKT_DATA_SIZE;
        if (m_txd_pkt.counts *
MAX_PKT_DATA_SIZE != m_txd_data_size){
            m_txd_pkt.counts++;
        }
        if (ProgressFormTx){
            ProgressFormTx->Free();
            ProgressFormTx = NULL;
        }
        ProgressFormTx = new TProgressForm(this);
        ProgressFormTx->Caption = "Процесс передачи";
        ProgressFormTx->CGauge->MaxValue = m_txd_pkt.counts;
        ProgressFormTx->CGauge->ForeColor = clBlue;
        ProgressFormTx->CGauge->Progress = 0;
        ProgressFormTx->Show();
        ReadFile();
        ProcessTxFile();
        SendFile();
    }
}

if (RadioButton3->Checked){
m_txd_data_size = SendSizeEdit->Text.ToInt()-PKT_HDR_SIZE;
    m_txd_pkt.pkt_num = 1;
    m_txd_pkt.counts = 1;
    m_txd_pkt.type = PACKET_TYPE_DATA;
    for (unsigned i=0; i<m_txd_pkt.pkt_size; i++){
        m_txd_pkt.data[i] = i;
    }
    CreatePipe(MKI_CMD_SEND, (unsigned char*)&m_txd_pkt,
PKT_HDR_SIZE+m_txd_pkt.pkt_size);
    AnsiString t = FormatDateTime("hh:nn:ss",Time()) + "
Передача пакета " + IntToStr(PKT_HDR_SIZE+m_txd_pkt.pkt_size) + " байт";
    LogMemo->Lines->Add(t);
}
if (RadioButton5->Checked){
    CreatePipe(MKI_CMD_END, NULL, 0);
}
}
void TMainForm::CreateHeader(CDte::dte_header_t* header, unsigned char
format)
{
    header->sign = CDte::DTE_SIGN_MKI_MSH;
    header->type = CDte::DTE_TYPE_CMD_OUT;
    header->format = format;
    header->result = 0;
}
void TMainForm::PipeTo()
{

```

```

        if (m_pipe_tx.Count()){
            pipe_t* pipe = (pipe_t*)m_pipe_tx.GetReadPtr();
            if (m_dte.SendData(pipe->Buffer(), pipe->size)){
                free(pipe->Buffer());
                m_pipe_tx.ReadEmpty(sizeof(pipe_t));
            }
        }
    }
}

void TMainForm::PutPipeTo(pipe_t* pipe)
{
    if (m_pipe_tx.Free() >= sizeof(pipe_t)){
        m_pipe_tx.Write((unsigned char*)pipe, sizeof(pipe_t));
    } else {
        while (1);
    }
    PipeTo();
}

void TMainForm::CreatePipe(unsigned char cmd, unsigned char* buf,
unsigned size)
{
    pipe_t pipe;
    pipe.size = sizeof(CDte::dte_header_t) + size;
    pipe.buff = (unsigned)malloc(pipe.size);
    if (pipe.buff == NULL){
        while (1);
    }
    CDte::dte_header_t* header = (CDte::dte_header_t*)pipe.Buffer();
    CreateHeader(header, cmd);
    if (size){
        memcpy(header->Data(), buf, size);
    }
    TListItem* item = LogView->Items->Add();
    item->ImageIndex = 0;
    item->Caption = FormatDateTime("hh:nn:ss",Time());
    item->SubItems->Add(getOutCmdLog(header->format));
    item->SubItems->Add("");
    item->SubItems->Add(getDataLog((unsigned char*)pipe.Buffer(),
pipe.size));
    LogView->Scroll(item->GetPosition().x, item->GetPosition().y);
    PutPipeTo(&pipe);
}

AnsiString TMainForm::getDataLog(unsigned char* data, unsigned size)
{
    AnsiString text;
    for (unsigned i=0; i<size; i++){
        text += IntToHex(data[i], 2) + " ";
    }
    return text;
}

void TMainForm::CmdAction(CDte::dte_header_t* header, unsigned size)
{
    AnsiString t;
    AnsiString file_name;
    packet_t* rxd_pkt = (packet_t*)header->Data();
    unsigned s = size - sizeof(CDte::dte_header_t);
    TListItem* item;

```

```

        if (header->sign == CDte::DTE_SIGN_MKI_MSH){
            switch (header->format){
                case MKI_CMD_BEGIN_RES:
                    break;
                case MKI_CMD_RXD:
                    //item = LogView->Items->Add();
                    //item->Caption = "rx pkt_num " + IntToStr(rxd_pkt-
>pkt_num) + ", pkt_size " + IntToStr(rxd_pkt->pkt_size) + ", counts " +
IntToStr(rxd_pkt->counts);
                    if (rxd_pkt->pkt_num == 1){
                        file_name = FormatDateTime("yymmdd_hhnnss",
Now());

                        if (rxd_pkt->type == PACKET_TYPE_DATA){
                            file_name += ".bin";
                            is_rpu = false;
                        } else {
                            file_name += ".pcm";
                            is_rpu = true;
                        }

                        t = FormatDateTime("hh:nn:ss",Time()) + "
Создание файла для приёма " + file_name;
                        LogMemo->Lines->Add(t);
                        m_rxd_file = fopen(file_name.c_str(), "wb");
                        if (ProgressFormRx){
                            ProgressFormRx->Free();
                            ProgressFormRx = NULL;
                        }
                        ProgressFormRx = new TProgressForm(this);
                        ProgressFormRx->Caption = "Процесс приёма";
                        ProgressFormRx->CGauge->MaxValue = rxd_pkt->counts;
                        ProgressFormRx->CGauge->ForeColor = clGreen;
                        ProgressFormRx->CGauge->Progress = 0;
                        ProgressFormRx->Show();
                        ProgressFormRx->Top = ProgressFormRx->Top + 80;
                    }
                    if (m_rxd_file){
                        if (m_fifo_data_rx.Free() >= rxd_pkt->pkt_size){
                            m_fifo_data_rx.Write(rxd_pkt->data, rxd_pkt->pkt_size);
                        }
                        ProcessRxFile();

                        if (ProgressFormRx){
                            ProgressFormRx->CGauge->Progress++;
                        }
                    }
                    t = FormatDateTime("hh:nn:ss",Time()) + " Приём пакета " + IntToStr(s) +
" байт";

                    if (rxd_pkt->pkt_num == rxd_pkt->counts){
                        is_wait_rx_rpu = true;
                    }
                    break;
                case MKI_CMD_SEND_RES:
                    is_msh_busy_rx = false;
                    if (m_txd_pkt.pkt_num != m_txd_pkt.counts){
                        m_txd_pkt.pkt_num++;
                        if (ProgressFormTx){

```

```

        ProgressFormTx->CGauge->Progress++;
    }
    SendFile();
} else {
    if (ProgressFormTx){
        ProgressFormTx->Free();
        ProgressFormTx = NULL;
    }
    memset(&m_txd_pkt, 0, sizeof(m_txd_pkt));
}
break;
default:
break;
};
}

}
void __fastcall TMainForm::WaitTimerTimer(TObject *Sender)
{
    unsigned char *buffer;
    if (!is_init){
        return;
    }
    ReadFile();
    ProcessTxFile();
    SendFile();
    ProcessRxFile();
    unsigned size = m_fifo_tx.Count();
    if (size){
        buffer = (unsigned char*)malloc(size);
        if (buffer){
            m_fifo_tx.Read(buffer, size);
            SendToMSH(buffer, size);
            TListItem* item = LogView->Items->Add();
            item->ImageIndex = 0;
            item->Caption =
FormatDateTime("hh:nn:ss",Time());
            item->SubItems->Add("");
            item->SubItems->Add(getDataLog(buffer, size));
            LogView->Scroll(item->GetPosition().x, item-
>GetPosition().y);
        }
    }
    size = m_dte.Receive(tmp_buffer);
    if (size >= sizeof(tmp_buffer)){
        while (1);
    }
    if (size){
        CDte::dte_header_t* header =
(CDte::dte_header_t*)tmp_buffer;
        TListItem* item = LogView->Items->Add();
        item->ImageIndex = 1;
        item->Caption = FormatDateTime("hh:nn:ss",Time());
        item->SubItems->Add(getInCmdLog(header->format));
        item->SubItems->Add("");
        item->SubItems->Add(getDataLog((unsigned char*)header,
size));
    }
}

```

```

                                LogView->Scroll(item->GetPosition().x, item-
>GetPosition().y);
                                CmdAction(header, size);
                                }
                                PipeTo();
                                hw_dte.Idle();
                                }
//-----
void __fastcall TMainForm::N3Click(TObject *Sender)
{
    LogView->Clear();
}
//-----
void __fastcall TMainForm::N1Click(TObject *Sender)
{
    LogMemo->Clear();
}

```

Файл «MKI.cpp»

```

#include <vcl.h>
#pragma hdrstop
//-----
USEFORM("main.cpp", MainForm);
USEFORM("progress.cpp", ProgressForm);
//-----
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
    try
    {
        Application->Initialize();
        Application->CreateForm(__classid(TMainForm),
&MainForm);
        Application->CreateForm(__classid(TProgressForm),
&ProgressForm);
        Application->Run();
    }
    catch (Exception &exception)
    {
        Application->ShowException(&exception);
    }
    catch (...)
    {
        try
        {
            throw Exception("");
        }
        catch (Exception &exception)
        {
            Application->ShowException(&exception);
        }
    }
    return 0;
}

```