

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт
Факультет информационных
технологий и электроники

Кафедра
«Информационная безопасность
систем и технологий»

Специальность 10.05.03 –

Информационная безопасность
автоматизированных систем

Специализация -

Защищенные автоматизированные
системы управления

ДИПЛОМНЫЙ ПРОЕКТ

на тему

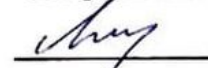
ПРОГРАММНЫЙ КОМПЛЕКС ДЛЯ ЗАЩИЩЕННОГО
ОБНОВЛЕНИЯ ВСТРОЕННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ
МИКРОКОНТРОЛЛЕРОВ STM32

Студент



Мирошников Антон Алексеевич

Руководитель



Липилин О.В.

Нормоконтролёр



Фатеев А. Г

Рецензент заместитель генерального
директора АО "ПНИЭИ"



Сомкин С. А.

Работа допущена к защите (протокол заседания кафедры от 25.06.21 № 16)

Заведующий кафедрой



С.Л. Зефилов

Работа защищена с отметкой отлично (протокол заседания ГЭК
от 28.06.21 № 6)



А.П. Иванов

Секретарь ГЭК

Пенза 2021

Оглавление

Введение.....	4
1 Анализ процесса загрузки программного обеспечения в микроконтроллерах STM32.....	5
2 Анализ угроз процесса загрузки.....	9
3 Модель нарушителя.....	12
4 Обзор защитных мер.....	17
5 Разработка алгоритмов программного комплекса.....	19
5.1 Обзор алгоритма шифрования ГОСТ Р 34.12—2015 «Магма».....	19
5.2 Обзор режима работы блочных шифров Multilinear Galois Mode.....	21
5.3 Разработка алгоритма генератора защищенного образа.....	25
5.4 Разработка алгоритма программатора.....	28
5.5 Разработка алгоритма загрузчика.....	31
6 Реализация программного комплекса	32
6.1 Реализация алгоритма шифрования ГОСТ Р 34.12—2015 «Магма»	32
6.2 Реализация режима работы блочных шифров Multilinear Galois Mode	36
6.3 Реализация генератора защищенного образа.....	44
6.4 Реализация программатора.....	47
6.5 Реализация загрузчика.....	50
7 Тестирование программного комплекса.....	54
7.1 Тестирование реализации алгоритма шифрования ГОСТ Р 34.12—2015 «Магма».....	54
7.2 Тестирование реализации режима работы блочных шифров Multilinear Galois Mode.....	57
7.3 Тестирование генератора защищенного образа.....	60
7.4 Тестирование программатора и загрузчика.....	63

8 Вредные психофизиологические факторы, влияющие на органы зрения специалиста по защите информации.....	69
8.1 Обзор вредных психофизиологических факторов.....	69
8.2 Методы и средства физической культуры, снижающие воздействие вредных психофизиологических факторов.....	71
Заключение.....	74
Список используемых источников.....	75
Приложение А Блок-схемы алгоритмов программного комплекса.....	77
Приложение Б Текст программы генератора защищенного образа.....	85
Приложение В Текст программы программатора.....	95
Приложение Г Текст программы загрузчика.....	106

Введение

В настоящее время широкое распространение получили системы, построенные на базе микроконтроллеров. В связи с этим возникает необходимость обеспечения безопасности таких систем. Ущерб, который может нанести несанкционированный доступ к паролям и персональным данным, достаточно очевиден, но в защите нуждается также и само встроенное программное обеспечение устройства (далее - ПО).

Встроенное ПО микроконтроллеров уязвимо к угрозам информационной безопасности в процессе его загрузки, так как при передаче по незащищенным каналам связи может быть нарушена его целостность, конфиденциальность и аутентичность. Нарушение конфиденциальности встроенного ПО может привести к обходу лицензионных и программных ограничений, копированию алгоритмов и даже к клонированию всего устройства и созданию реплики. Нарушитель так же может осуществить внедрение программных закладок с целью скрытого противодействия корректному функционированию устройства и систем, построенных на базе таких устройств.

Цель работы – разработка программного комплекса для защищенного обновления встроенного ПО предназначен для обеспечения безопасности процесса загрузки (далее - Комплекс).

Основными задачами работы являются:

- анализ угроз процесса загрузки встроенного ПО;
- определение защитных мер, противодействующих выявленным угрозам;
- разработка Комплекса, включающего определенные защитные меры;
- реализация и тестирование Комплекса.

1 Анализ процесса загрузки программного обеспечения в микроконтроллерах STM32

Микроконтроллеры предназначены для управления разными электронными приборами и устройствами. Область их применения включает, в частности, средства обеспечения физической и информационной безопасности (далее - ИБ). Часто возникает необходимость загрузки или обновления программного обеспечения таких устройств, когда они уже находятся в эксплуатации. Примером могут быть случаи, когда в процессе эксплуатации в программном обеспечении (далее - ПО) устройства были обнаружены ошибки, или возникла необходимость добавления новых функций. Такой процесс, передачи и записи встроенного ПО в энергонезависимую память микроконтроллера с целью его дальнейшего выполнения, называет процессом загрузки программного обеспечения.

Процесс загрузки подвержен угрозам информационной безопасности, следовательно, существует необходимость обеспечения защиты этого процесса. Защита подразумевает применение защитных мер. Выбор защитных мер осуществляется на основе модели угроз. Для составления модели угроз процесса надо описать сам процесс.

Было составлено описание типовой модели процесса загрузки программного обеспечения в микроконтроллерах STM32.

Источником встроенного ПО является организации-разработчик, которая размещает версии ПО для своих продуктов на сервере. С сервера по сети Интернет данное ПО может быть передано на персональный компьютер (ПК) лица, которое имеет физический доступ к устройству, ПО которого требуется загрузить или обновить. Данное лицо, подключив устройство к ПК, может осуществить передачу и запись встроенного ПО в энергонезависимую память микроконтроллера с помощью средств программирования.

Совокупность технических средств и каналов связи, используемых в процессе загрузки программного обеспечения, составляющих систему загрузки ПО, проиллюстрирована на рисунке 1.

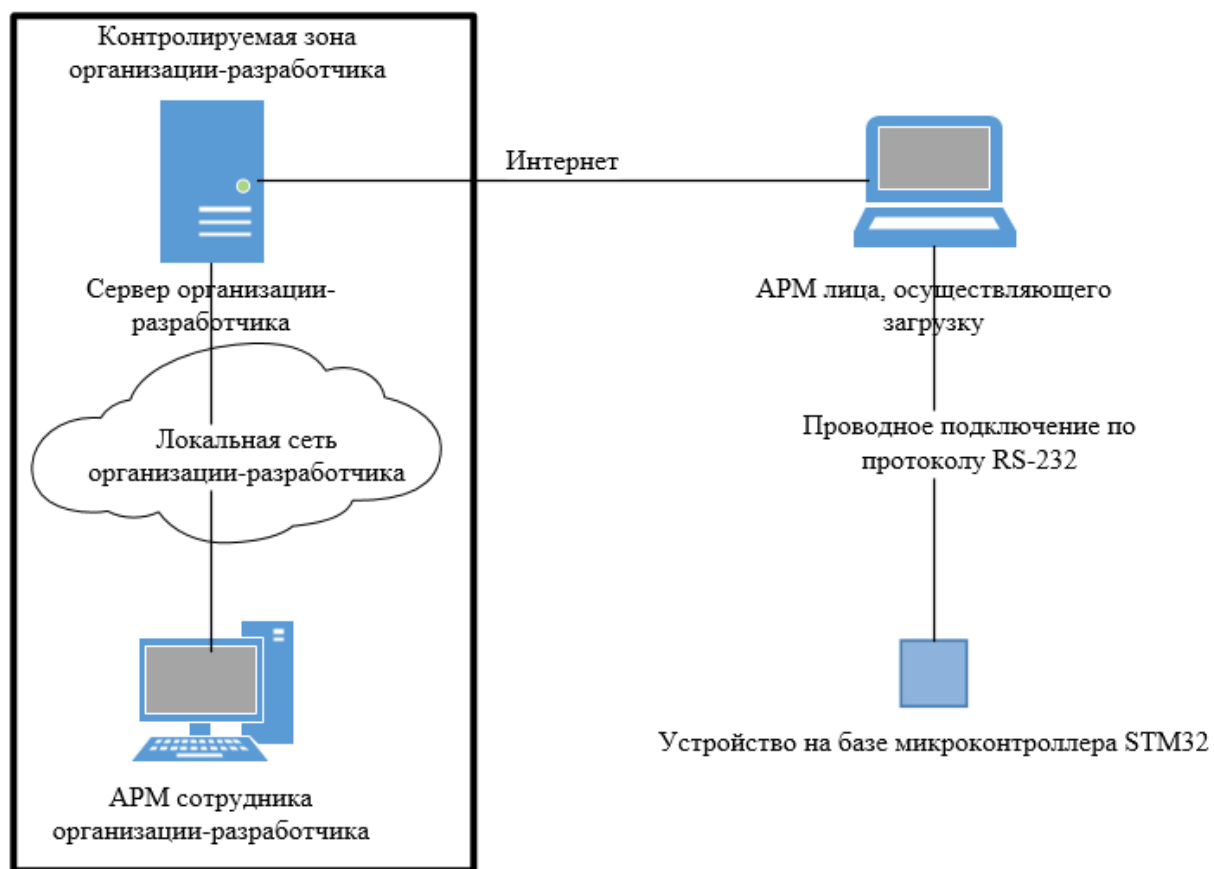


Рисунок 1 — Иллюстрация процесса загрузки программного обеспечения в микроконтроллерах STM32

Процесс загрузки программного обеспечения включает следующие процессы:

- размещение встроенного ПО на сервере организации-разработчика;
- передача встроенного ПО на ПК лица, осуществляющего загрузку;
- запись встроенного ПО в энергонезависимую память микроконтроллера с помощью загрузчика.

Описание процесса загрузки с использованием нотации IDEF0 приведено на рисунке 2.

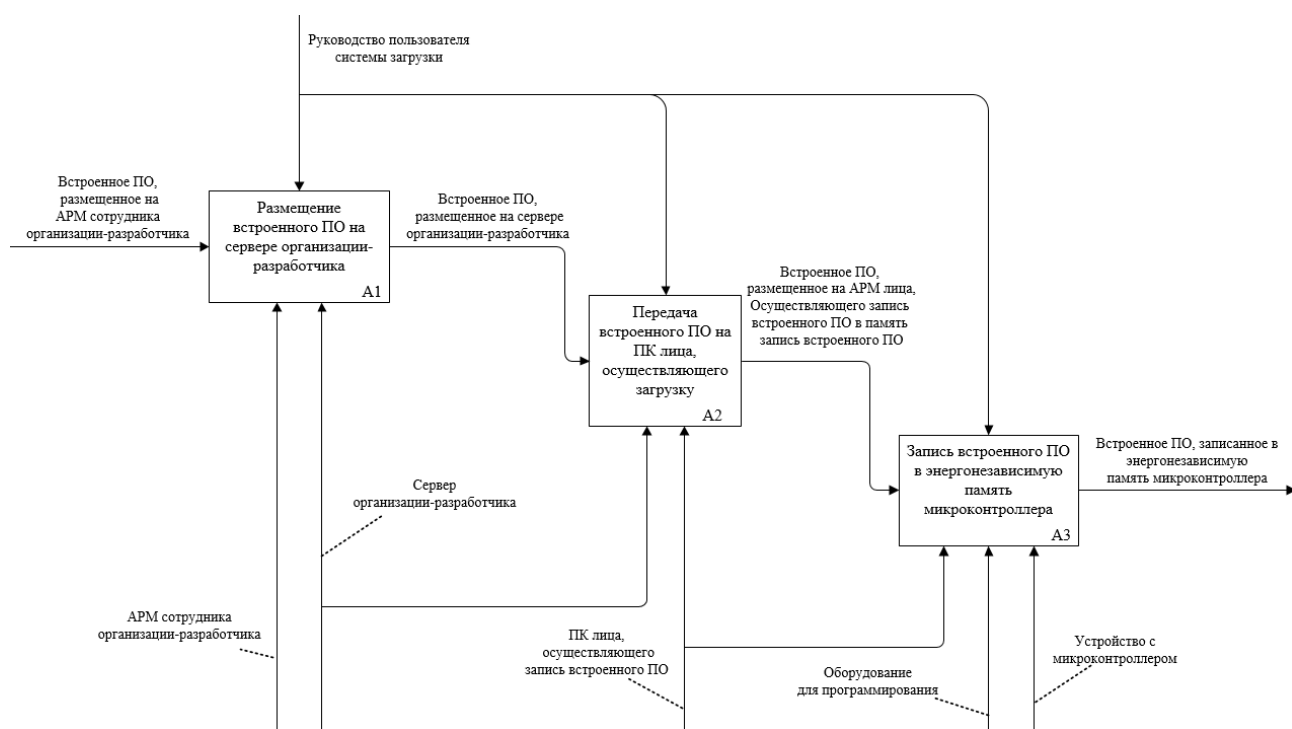


Рисунок 2 - Описание процесса загрузки с использованием нотации IDEF0

Входной информацией процесса «Размещение встроенного ПО на сервере организации-разработчика» является встроенное ПО, размещенное на АРМ сотрудника организации-разработчика. Размещение новых версий встроенного ПО на сервере организации-разработчика осуществляется сотрудником организации с его АРМ. Сервер и АРМ располагаются в пределах контролируемой зоны организации. Результатом процесса является «Встроенное ПО, размещенное на сервере организации-разработчика».

Для процесса «Передача встроенного ПО на ПК лица, осуществляющего загрузку» входной информацией является встроенное ПО, размещенное на сервере организации-разработчика. Передача встроенного ПО на ПК лица, осуществляющего загрузку осуществляет посредством подключения ПК к серверу организации-разработчика по сети Интернет. Результатом процесса является встроенное ПО, размещенное на АРМ лица, осуществляющего загрузку.

Входной информацией процесса «Запись встроенного ПО в энергонезависимую память микроконтроллера» является встроенное ПО, размещенное на АРМ лица, осуществляющего загрузку. Запись встроенного ПО в энергонезависимую память микроконтроллера с помощью средств загрузки.

Средства загрузки включают следующие компоненты:

- программатор, установленный на компьютере — это ПО, установленное на ПК, которое обеспечивает передачу ПО на микроконтроллер, индикацию и контроль процесса передачи;

- загрузчик — это ПО, функционирующее на микроконтроллере, которое обеспечивает прием встроенного ПО, его запись в энергонезависимую память микроконтроллера;

- коммутационное оборудование — преобразователь (USB - UART) и соединительные кабели, необходимые для подключения микроконтроллерного устройства к ПК.

Результатом процесса «Запись встроенного ПО в энергонезависимую память микроконтроллера» является встроенное ПО, записанное в энергонезависимую память микроконтроллера.

Таким образом, в результате анализа описанного процесса загрузки программного обеспечения в микроконтроллерах STM32 можно сделать вывод, что процессы А2 и А3 могут быть подвержены воздействию нарушителя ИБ, поскольку загружаемое встроенное ПО передается по незащищенным каналам связи и хранится на ПК лица, находящимся за пределами контролируемой зоны.

2 Анализ угроз процесса загрузки

На основе информации, полученной в результате анализа процесса загрузки программного обеспечения в микроконтроллерах STM32, были определены основные угрозы информационной безопасности данного процесса:

- угроза нарушения конфиденциальности встроенного ПО ;
- угроза подмены встроенного ПО;
- угроза нарушения целостности встроенного ПО.

Было составлено описание угроз. Результат приведен в таблице 1.

Таблица 1 — Описание угроз

№	Угроза	Описание угрозы	Возможные последствия реализации угрозы
1	Угроза нарушения конфиденциальности встроенного ПО	Угрозы заключается в копировании ПО во время процесса загрузки. Доступ к ПО может быть осуществлен при его передаче на ПК лица, осуществляющего загрузку и при записи встроенного ПО в энергонезависимую память микроконтроллера. При записи встроенного ПО в энергонезависимую память микроконтроллера нарушитель может осуществить копирование с ПК лица, осуществляющего записи посредством непосредственного физического доступа к ПК, либо эксплуатации уязвимостей системного и прикладного ПО ПК. Считывание встроенного ПО может быть произведено через незащищенные отладочные интерфейсы микроконтроллера.	Определение уязвимостей устройства в результате анализа нарушителем алгоритмов работы устройства. Обход лицензионных и программных ограничений. Копирование ключевых алгоритмов устройства с целью их продажи. Клонирование всего устройства и создание его реплики с целью продажи.

Продолжение таблицы 1

№	Угроза	Описание угрозы	Возможные последствия реализации угрозы
2	Угроза подмены встроенного ПО	<p>Угроза заключается в подмене оригинального встроенного ПО.</p> <p>Оригинальное встроенное ПО может быть подменено умышленно на ПО, созданное нарушителем, либо случайно в результате ошибки пользователя.</p> <p>Подмена встроенного ПО может быть проведена при передаче ПО на ПК, лица осуществляющего запись встроенного ПО и во время записи встроенного ПО в энергонезависимую память микроконтроллера.</p> <p>Во время процесса записи встроенного ПО энергонезависимую память микроконтроллера нарушитель может осуществить подмену встроенного ПО, хранящегося на ПК, лица осуществляющего запись посредством непосредственного физического доступа к ПК, либо эксплуатации уязвимостей системного и прикладного ПО ПК.</p>	<p>Нарушение работоспособности устройства.</p> <p>Внедрение программных закладок с целью скрытого противодействия корректному функционированию устройства и систем, построенных на базе таких устройств (саботаж)</p>

Продолжение таблицы 1

№	Угроза	Описание угрозы	Возможные последствия реализации угрозы
3	Угроза нарушения целостности ПО устройства	<p>Угроза заключается в возможности преднамеренного или непреднамеренного нарушения целостности встроенного ПО при его передаче или хранении.</p> <p>Преднамеренное нарушение целостности защищаемой информации может быть произведено путём осуществления нарушителем деструктивного физического воздействия на машинный носитель информации или деструктивного программного воздействия (в т.ч. изменение отдельных бит или полное затирание информации) на данные, хранящиеся на нём.</p>	Нарушение работоспособности устройства: полная неработоспособность или частичная потеря функциональных возможностей.

В результате анализа угроз было определено, что для процесса загрузки программного обеспечения в микроконтроллерах STM32 существуют угрозы, последствия реализации которых могут привести к финансовым и репутационным потерям для организации-разработчика.

3 Модель нарушителя

Анализ возможностей, которыми может обладать нарушитель, проводится в рамках модели нарушителя. Была описана модель нарушителя, включающая следующие компоненты:

- вид нарушителя;
- тип нарушителя;
- цели нарушителя;
- возможности по реализации угроз.

Были определены виды нарушителей безопасности информации в процессе загрузки программного обеспечения в микроконтроллерах STM32:

- конкурирующие организации;
- криминальные структуры;
- внешние субъекты (физические лица);
- лица, осуществляющие запись встроенного ПО энергонезависимую память микроконтроллера;
- сотрудники организации-разработчика, участвующие в процессе загрузки.

С учетом участия в процессе загрузки программного обеспечения нарушители подразделяются на два типа:

- внешние нарушители (тип I) – лица, не принимающие участия в процессе загрузки и реализующие угрозы безопасности информации из-за границ информационной системы;
- внутренние нарушители (тип II) – лица, принимающие участия в процессе загрузки и имеющие право доступа к информационной системе, ее отдельным компонентам.

Типы и виды нарушителя безопасности приведены в таблице 2.

Таблица 2 — Типы и виды нарушителей

№ п/п	Виды нарушителей	Типы нарушителей
1	Конкурирующие организации	Внешний
2	Криминальные структуры	Внешний
3	Внешние субъекты (физические лица)	Внешний
4	Лица, осуществляющие запись встроенного ПО энергонезависимую память микроконтроллера	Внутренний
5	Сотрудники организации-разработчика, участвующие в процессе загрузки	Внутренний

Были определены возможные цели (мотивация) реализации угроз безопасности информации нарушителями [1]. Результат приведен в таблице 3.

Таблица 3 — Цели нарушителей

№ п/п	Виды нарушителей	Возможные цели (мотивация) реализации угроз безопасности информации
1	Конкурирующие организации	Получение конкурентных преимуществ путем причинение имущественного ущерба, копирование устройств, на базе микроконтроллерах STM32, с целью их дальнейшей продажи, неправомерного использования встроенного ПО.
2	Криминальные структуры	Причинение имущественного ущерба путем мошенничества или иным преступным путем. Выявление уязвимостей и копирование ПО устройств, на базе микроконтроллерах STM32, с целью их дальнейшей продажи и получения финансовой выгоды.

Продолжение таблицы 3

№ п/п	Виды нарушителей	Возможные цели (мотивация) реализации угроз безопасности информации
3	Внешние субъекты (физические лица)	Причинение имущественного ущерба путем мошенничества или иным преступным путем. Любопытство или желание самореализации (подтверждение статуса). Выявление уязвимостей и копирование ПО устройств, на базе микроконтроллерах STM32, с целью их дальнейшей продажи и получения финансовой выгоды.
4	Лица, осуществляющие запись встроенного ПО энергонезависимую память микроконтроллера	Причинение имущественного ущерба путем мошенничества или иным преступным путем. Любопытство или желание самореализации (подтверждение статуса). Месть за ранее совершенные действия. Непреднамеренные, неосторожные или неквалифицированные действия.
5	Сотрудники организации-разработчика, участвующие в процессе загрузки	Причинение имущественного ущерба путем мошенничества или иным преступным путем. Любопытство или желание самореализации (подтверждение статуса). Месть за ранее совершенные действия. Непреднамеренные, неосторожные или неквалифицированные действия.

Были определены возможности по реализации угроз безопасности информации. Результат приведен в таблице 4.

Таблица 4 — Возможности по реализации угроз безопасности информации

№ п/п	Виды нарушителей	Возможности по реализации угроз безопасности информации
1	Конкурирующие организации	<p>Имеют возможность получить информацию об уязвимостях процесса загрузки, опубликованную в общедоступных источниках.</p> <p>Имеют возможность получить информацию о методах и средствах реализации угроз безопасности информации (компьютерных атак), опубликованных в общедоступных источниках, и (или) самостоятельно осуществляет создание методов и средств реализации атак и реализацию атак на систему загрузки ПО.</p> <p>Имеют осведомленность о мерах защиты информации, применяемых в процессе загрузки.</p>
2	Криминальные структуры	
3	Внешние субъекты (физические лица)	
4	Лица, осуществляющие запись встроенного ПО энергонезависимую память микроконтроллера	<p>Имеют возможность получить информацию об уязвимостях процесса загрузки, опубликованную в общедоступных источниках.</p> <p>Имеют возможность получить информацию о методах и средствах реализации угроз безопасности информации (компьютерных атак), опубликованных в общедоступных источниках, и (или) самостоятельно осуществляет создание методов и средств реализации атак и реализацию атак на систему загрузки ПО.</p> <p>Имеют осведомленность о мерах защиты информации, применяемых в процессе загрузки.</p> <p>Имеют физический доступ к компонентам системы загрузки в процессе записи встроенного ПО энергонезависимую память микроконтроллера.</p>

Продолжение таблицы 4

№ п/п	Виды нарушителей	Возможности по реализации угроз безопасности информации
5	Сотрудники организации-разработчика, участвующие в процессе загрузки	<p>Имеют возможность получить информацию об уязвимостях процесса загрузки, опубликованную в общедоступных источниках.</p> <p>Имеют возможность получить информацию о методах и средствах реализации угроз безопасности информации (компьютерных атак), опубликованных в общедоступных источниках, и (или) самостоятельно осуществляет создание методов и средств реализации атак и реализацию атак на систему загрузки ПО.</p> <p>Имеют осведомленность о мерах защиты информации, применяемых в процессе загрузки</p> <p>Имеют доступ к сведениям о структурно-функциональных характеристиках и особенностях функционирования системы загрузки ПО.</p> <p>Имеют физический доступ к компонентам системы загрузки в процессе размещения встроенного ПО на сервере организации-разработчика .</p>

Таким образом, была составлена модель нарушителя для процесса загрузки программного обеспечения в микроконтроллерах STM32.

4 Обзор защитных мер

Были выбраны защитные меры, направленные на противодействие ранее выявленным угрозам процесса загрузки программного обеспечения в микроконтроллерах STM32.

В результате анализа угроз было определено, что нарушения конфиденциальности встроенного ПО может произойти при его передаче и хранении на ПК лица, осуществляющего загрузку. Для противодействия данной угрозы необходимо применять шифрование, так как данное криптографическое преобразование позволяет обеспечивать конфиденциальность информации [2]. ПО должно зашифровываться перед его размещением на сервере организации-разработчика и расшифровываться на микроконтроллере во время процесса записи встроенного ПО в энергонезависимую память микроконтроллера. Ключ шифрования должен быть заранее сгенерирован и записан в энергонезависимую память микроконтроллера при производстве вместе с программой загрузчика. Используемый загрузчик должен обеспечивать функцию расшифрования с использованием данного ключа. Должна обеспечиваться конфиденциальность ключа, хранимого в памяти микроконтроллера.

Для обеспечения конфиденциальности ключей шифрования и встроенного ПО после его записи в энергонезависимую память микроконтроллера необходимо использовать механизм RDP (Code Read Out Protection), либо PCROP (Proprietary code readout protection).

RDP – защитный механизм, отключающий возможность считывать флэш-память через отладочный интерфейс, а также позволяющий запретить загрузку из внешних носителей [3].

PCROP (Proprietary code readout protection) – механизм, обеспечивающий возможность запрета на осуществление операций чтения, записи и стирания с Flash-памятью [4].

Для противодействия угрозе подмены встроенного ПО и угрозе нарушения целостности могут использоваться коды аутентификации. Коды аутентификации сообщений – это разновидность криптографической контрольной суммы, предназначенной для проверки аутентичности защищаемой информации [2]. Они могут быть построены на основе хэш-функций с ключом, алгоритмах вычисления имитовставки, а также на основе некоторых режимов использования шифров. Для обеспечения конфиденциальности ключа необходимо применять ранее описанный защитный механизм – RDP.

Соответствие выбранных защитных меры и угроз, которым они противодействуют, показано в таблице 5.

Таблица 5 — Защитных меры и угрозы, которым они противодействуют

№	Угроза	Защитные меры
1	Угроза нарушения конфиденциальности встроенного ПО	Шифрование, RDP, PCROP
2	Угроза подмены встроенного ПО	Коды аутентификации сообщений, RDP, PCROP
3	Угроза нарушения целостности ПО устройства	Коды аутентификации сообщений, RDP, PCROP

Таким образом, были выбранные защитные меры, которые позволяют противодействовать всем выявленным угрозам.

5 Разработка алгоритмов программного комплекса

5.1 Обзор алгоритма шифрования ГОСТ Р 34.12—2015 «Магма»

Так как техническое задание содержит требование использования в программном комплексе алгоритма шифрования ГОСТ Р 34.12—2015 «Магма», перед разработкой алгоритмов модулей программного комплекса необходимо провести обзор данного алгоритма шифрования.

Алгоритм блочного шифрования "Магма", включенный в ГОСТ Р 34.12-2015 основан на схеме Фейстеля.

Его основные параметры: [5]

- длина блока данных – 64 бита;
- длина ключа – 256 бит;
- длина итерационного ключа – 32 бита;
- количество итераций – 32.

Алгоритм зашифрования E_k реализуется 32 итерациями сети Фейстеля в соответствии с формулой (1).

$$\begin{aligned} E_k(a) &= G^*[k_{32}]G[k_{31}]..G[k_2]G[k_1](a_1, a_0), \\ a &= a_1 \parallel a_0 \in V_{64}, \quad a_0, a_1 \in V_{32}, \\ k_i &\in V_{32}, \quad i = 1, 2, \dots, 32. \end{aligned} \tag{1}$$

где E_k – преобразование зашифрования, a – открытый текст, $G[k]$ – итерационное преобразование сети Фейстеля, V_s – множество всех двоичных строк длины s , k_i – итерационный ключ, i – номер итерации.

Алгоритм расшифрования D_k реализуется той же схемой с обратным порядком следования итерационных ключей соответствии с формулой (2).

$$\begin{aligned}
D_k(a) &= G^*[k_1]G[k_1]..G[k_{31}]G[k_{32}](a_1, a_0), \\
a &= a_1 \parallel a_0 \in V_{64}, \quad a_0, a_1 \in V_{32}, \\
k_i &\in V_{32}, \quad i = 1, 2, \dots, 32.
\end{aligned} \tag{2}$$

где D_k – преобразование расшифрования, a – шифртекст, $G[k]$ – итерационное преобразование сети Фейстеля, V_s – множество всех двоичных строк длины s , k_i – итерационный ключ, i – номер итерации.

Итерация сети Фейстеля описывается в виде преобразования $G[k]$, заданного формулой (3).

$$\begin{aligned}
G[k]: V_{32} \times V_{32} &\rightarrow V_{32} \times V_{32}, \\
G[k](a_1, a_0) &= (a_0, g[k](a_0) \oplus a_1), \\
k, a_0, a_1 &\in V_{32}.
\end{aligned} \tag{3}$$

где $G[k]$ – итерационное преобразование сети Фейстеля, V_s – множество всех двоичных строк длины s , a_1, a_0 – преобразуемые данные, k – итерационный ключ.

Последняя итерация описывается формулой (4).

$$\begin{aligned}
G^*[k]: V_{32} \times V_{32} &\rightarrow V_{64}, \\
G^*[k](a_1, a_0) &= (g[k](a_0) \oplus a_1) \parallel a_0, \\
k, a_0, a_1 &\in V_{32}.
\end{aligned} \tag{4}$$

где $G^*[k]$ – преобразование последней итерации сети Фейстеля, V_s – множество всех двоичных строк длины s , a_1, a_0 – преобразуемые данные, k – итерационный ключ.

Преобразование $g[k]$ заданно формулой (5).

$$\begin{aligned}
g[k]: V_{32} &\rightarrow V_{32}, \\
g[k](a) &= (t(Vec_{32}(Int_{32}(a) \oplus_{32} Int_{32}(k)))) \ll_{11}, \\
a, k &\in V_{32}.
\end{aligned} \tag{5}$$

где V_s – множество всех двоичных строк длины s , k – итерационный ключ, a – преобразуемые данные, t – преобразование подстановки, Vec_{32} – биективное отображение, сопоставляющее элементу кольца Z_2^{32} его двоичное представление, Int_{32} – отображение, обратное к отображению Vec_{32} .

Функция шифрования g параметризована итерационным ключом k и ставит в соответствие 32х битному входному вектору a 32х битный выходной вектор (преобразование правого подблока в сети Фейстеля). Преобразование включает следующие шаги: сложение значения обрабатываемого подблока с итерационным ключом в кольце Z_2^{32} . Входной подблок и ключ представляются числами в этом кольце, затем выполняется сложение по модулю 2^{32} . Результат опять представляется 32х разрядным вектором, над которым выполняется преобразование t . Результатом подстановки также является 32х битный вектор, над которым выполняется операция циклического сдвига на 11 разрядов влево.

Преобразование подстановки t выполняется следующим образом:

$$\begin{aligned}
t: V_{32} &\rightarrow V_{32}, \\
t(a) &= t(a_7 \parallel \dots \parallel a_0) = \pi_7(a_7) \parallel \dots \parallel \pi_0(a_0),
\end{aligned} \tag{6}$$

где t – преобразование подстановки, V_s – множество всех двоичных строк длины s , a_i – преобразуемые данные, π – блок подстановок.

Преобразование t ставит в соответствие входному 32х битному вектору выходной вектор такого же размера. Преобразование заключается в разбиении блока на восемь (от нулевого до седьмого) четырехбитных векторов. Входной четырехбитный вектор преобразуется в элемент кольца Z_2^4 , над элементом

кольца выполняется табличная подстановка, результат которой преобразуется в четырехбитный вектор (выполняется операция $\text{Vec } n$, ставящая в соответствие числу вектор).

Таблицы подстановок (S-блоки) описаны формулой (7).

$$\begin{aligned}
 \pi_0' &= (12, 4, 6, 2, 10, 5, 11, 9, 14, 8, 13, 7, 0, 3, 15, 1); \\
 \pi_1' &= (6, 8, 2, 3, 9, 10, 5, 12, 1, 14, 4, 7, 11, 13, 0, 15); \\
 \pi_2' &= (11, 3, 5, 8, 2, 15, 10, 13, 14, 1, 7, 4, 12, 9, 6, 0); \\
 \pi_3' &= (12, 8, 2, 1, 13, 4, 15, 6, 7, 0, 10, 5, 3, 14, 9, 11); \\
 \pi_4' &= (7, 15, 5, 10, 8, 1, 6, 13, 0, 9, 3, 14, 11, 4, 2, 12); \\
 \pi_5' &= (5, 13, 15, 6, 9, 2, 12, 10, 11, 7, 8, 1, 4, 3, 14, 0); \\
 \pi_6' &= (8, 14, 2, 5, 6, 9, 1, 12, 15, 4, 11, 0, 13, 10, 3, 7); \\
 \pi_7' &= (1, 7, 14, 13, 0, 5, 8, 3, 4, 15, 10, 6, 9, 12, 11, 2).
 \end{aligned} \tag{7}$$

где π – блок подстановок.

Алгоритм развертывания ключа в шифре «Магма» заключается в непосредственном использовании частей секретного ключа k . При шифровании используется 32 итерационных ключа k_i по 32 бита каждый. Итерационные ключи вычисляются в соответствии с формулой (8).

$$\begin{aligned}
 k &= \kappa_{255} \parallel \dots \parallel \kappa_0, \kappa_j \in V_1, j = 0, 1, \dots, 255, \\
 k_1 &= \kappa_{255} \parallel \dots \parallel \kappa_{224}, \quad k_5 = \kappa_{127} \parallel \dots \parallel \kappa_{96}, \\
 k_2 &= \kappa_{223} \parallel \dots \parallel \kappa_{192}, \quad k_6 = \kappa_{95} \parallel \dots \parallel \kappa_{64}, \\
 k_3 &= \kappa_{191} \parallel \dots \parallel \kappa_{160}, \quad k_7 = \kappa_{63} \parallel \dots \parallel \kappa_{32}, \\
 k_4 &= \kappa_{159} \parallel \dots \parallel \kappa_{128}, \quad k_8 = \kappa_{31} \parallel \dots \parallel \kappa_0. \\
 k_{i+8} &= k_{i+16} = k_i, \quad i = 1, 2, \dots, 8, \\
 k_{i+24} &= k_{9-i}, \quad i = 1, 2, \dots, 8.
 \end{aligned} \tag{8}$$

где k_i – итерационный ключ, i – номер итерации, κ_j – бит j ключа шифрования.

5.2 Обзор режима работы блочных шифров Multilinear Galois Mode

Режим MGM (Multilinear Galois Mode) — режим работы блочного шифра, который можно использовать в качестве режима работы блочного шифра с длиной блока n равно 64 или n равно 128 [6]. Данный режим является AEAD-режимом и обеспечивает конфиденциальность и имитозащиту сообщений. При этом сообщения могут состоять из одной или двух частей. Если сообщение состоит из двух частей, то первая часть содержит дополнительные имитозащищаемые данные AAD и обозначается через A , а вторая часть называется открытым текстом и обозначается через P . Если сообщение состоит из одной части, то пустой строке должны быть равны либо дополнительные имитозащищаемые данные, либо открытый текст. Дополнительные имитозащищаемые данные не подлежат шифрованию, но требуют имитозащиты. Для открытого текста обеспечены конфиденциальность и имитозащита. Параметром режима MGM является длина имитовставки S , выраженная в битах, S принимает значение от 32 до n . Значение S должно быть зафиксировано в рамках каждого конкретного протокола исходя из требований к производительности системы и требований к стойкости режима относительно угрозы нарушения имитозащиты.

Процесс зашифрования сообщения, изображенный на рисунке 3, осуществляется при помощи функции MGM-Encrypt.

Функция MGM-Encrypt принимает на вход следующие значения:

- ключ шифрования K ;
- уникальный вектор поинса V_{n-1} ;
- дополнительные имитозащищаемые данные A , длина $|A|$ больше 0, но меньше $2^{n/2}$;
- открытый текст P , длина $|P|$ больше 0, но меньше $2^{n/2}$.

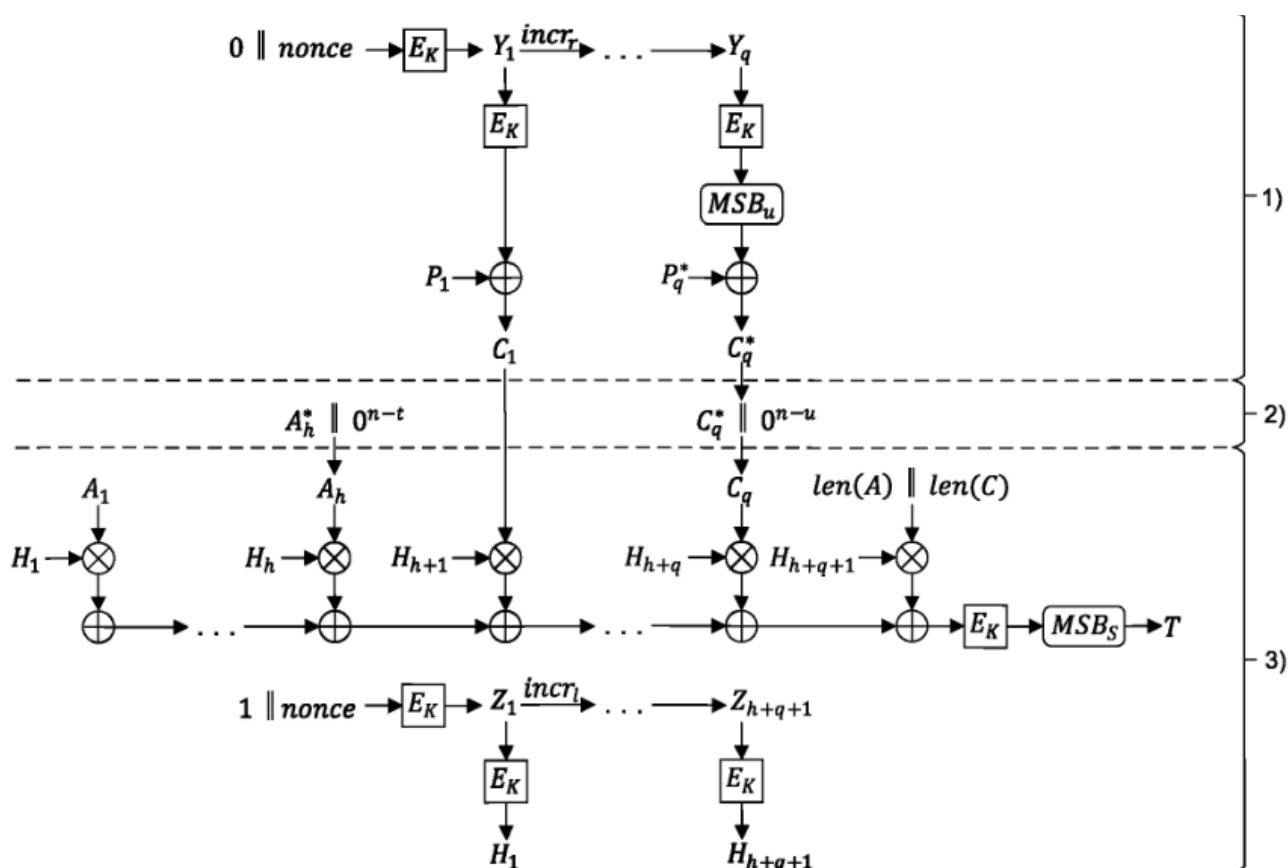


Рисунок 3 — Зашифрование сообщения в режиме MGM

Функция MGM-Encrypt принимает на вход следующие значения:

- ключ шифрования K ;
- уникальный вектор nonce V_{n-1} ;
- дополнительные имитозащищаемые данные A , длина $|A|$ больше 0, но меньше $2^{n/2}$;
- открытый текст P , длина $|P|$ больше 0, но меньше $2^{n/2}$.

Процесс расшифрования сообщения, изображенный на рисунке 4, осуществляется при помощи функции MGM-Decrypt .

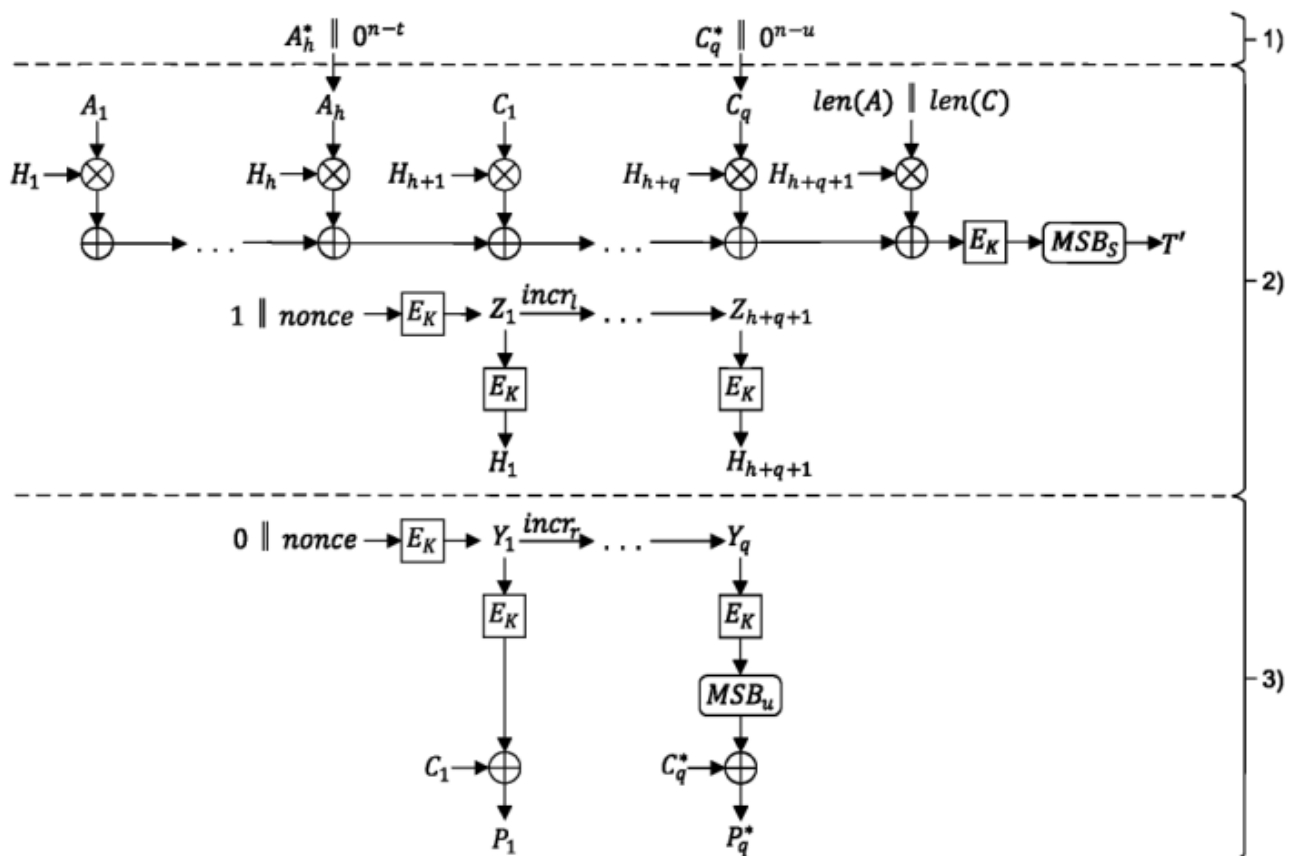


Рисунок 4 — Расшифрование сообщения в режиме MGM

Функция `MGM-Decrypt` принимает на вход следующие значения:

- ключ шифрования K ;
- уникальный вектор `nonce` V_{n-1} ;
- дополнительные имитозащищаемые данные A ;
- шифртекст C , длина $|C|$ больше 0, но меньше $2^{n/2}$;
- имитовставка T .

5.3 Разработка алгоритма генератора защищенного образа

Был разработан следующий алгоритм генератора защищенного образа.

После запуска программа ожидает ввода пользователем входных данных. После ввода осуществляется проверка входных данных. Если проверка была

пройдена успешно, происходит генерация защищенного образа, иначе программа осуществляет вывод сообщения о возникших ошибках.

К входным данным относятся:

- файл ключа шифрования;
- файл встроенного ПО в формате Intel HEX;
- версия встроенного ПО;
- идентификатор устройства, для которого предназначено встроенное ПО;
- каталог, в котором будет располагаться создаваемый защищенный образ.

Перед определением требований, предъявляемых к входным данным, был описан формат Intel HEX. [7]

Файл формата Intel HEX состоит из текстовых ASCII строк. Каждая строка представляет собой одну запись. Каждая запись начинается с двоеточия (символ «:»), после которого идёт набор шестнадцатеричных цифр кратных байту. Каждая запись состоит из следующих фрагментов:

- начало записи (символ «:») – 1 байт;
- количество байт данных, содержащихся в этой записи (значение в диапазоне от 0 до 255) – 1 байт;
- начальный адрес блока записываемых данных – 2 байта (этот адрес определяет абсолютное местоположение данных этой записи в двоичном файле);
- тип записи – 2 байта.
- данные, которые требуется сохранить в энергонезависимую память;
- контрольная сумма;
- конец записи (управляющие символы 0Dh 0Ah).

Существуют следующие типы записей:

- 00 – запись содержит данные двоичного файла;

- 01 – запись является концом файла;
- 02 – запись адреса сегмента;
- 03 – Start Segment Address Record;
- 04 – запись расширенного адреса;
- 05 – Start Linear Address Record.

Были определены требования, предъявляемые к входным данным. Дынные требования приведены в таблице 7.

Таблица 7 – Требования, предъявляемые к входным данным генератора защищенного образа

Входные данные	Требования, предъявляемые к входным данным
Файл ключа шифрования	Длина ключа должна быть 32 байта
Файл встроенного ПО	Файл должен содержать записи в описанном выше формате Intel HEX
Версия встроенного ПО	Версия ПО должна принимать значения в диапазоне от 0 до 4294967295 (4 байта)
Идентификатор устройства, для которого предназначено встроенное ПО	Длина идентификатора устройства должна быть не более 16 байт, идентификатор должен содержать только символы из следующего набора: прописные и строчные буквы латинского алфавита, символ «-», символ «_»
Каталог, в котором будет располагаться создаваемый защищенный образ	Указанный каталог должен существовать

Алгоритм проверки входных данных приведен на рисунках А.1 и А.2 в приложении А.

Перед разработкой алгоритма генерации защищенного образа был определен формат файла защищенного образа встроенного ПО, который показан на рисунке 5.

Поле идентификатора устройства необходимо для обеспечения возможности проверки соответствия защищенного образа встроенного ПО и устройства, в которое будет осуществляться загрузка.

Заголовок (24 байта)		
Идентификатор устройства (16 байт)	Версия ПО (4 байта)	Длина ПО (в блоках) (4 байта)
Блок 1 (8-256 байт)		
Длина блока (4 байта)	Данные – зашифрованная строка ПО (4 – 256 байт)	
Блок 2 (8-256 байт)		
...		
Блок N (8-256 байт)		
Имитовставка (8 байт)		

Рисунок 5 – Формат файла защищенного образа встроенного ПО

Поле версии ПО необходимо для обеспечения возможности сравнения версии загружаемого ПО и ПО, записанного в памяти микроконтроллера до начала загрузки, что позволяет обеспечить защиту от подмены защищённого образа встроенного ПО на образ, содержащий более раннюю версию ПО.

Поля длина ПО и длины блока необходимы для обеспечения возможности чтения защищённого образа встроенного ПО программатором.

Алгоритм генерации защищенного образа встроенного ПО показан приведен на рисунке А.3 в приложении А.

Таким образом, был разработан алгоритм работы генератора защищенного образа.

5.4 Разработка алгоритма программатора

Был разработан следующий алгоритм программатора.

После запуска программа ожидает выбора порта интерфейса стандарта RS-232, используемого для загрузки, и указания пользователем файла защищенного образа встроенного ПО. После выбора файла осуществляется его проверка. Если проверка была пройдена успешно, начинается процесс загрузки, иначе программа осуществляет вывод сообщений о возникших ошибках.

Алгоритм проверки входных данных приведен на рисунке А.4 в приложении А.

Процесс загрузки представляет собой обмен данными между программатором и загрузчиком, в результате которого происходит передача защищенного образа встроенного ПО. Описание команд программатора, используемых в процессе загрузки приведен в таблице 8.

Таблица 8 – Описание команд программатора

Команда	Описание команды
«Boot»	<p>Команда «Boot» отправляется программатором в начале процесса загрузки. Вместе с командой программатором отправляется идентификатор устройства и версия ПО. При получении данной команды загрузчик проверяет полученные данные и отправляет один из следующих ответов: «Boot_OK», «OldVer», «DiffID».</p> <p>Ответ «Boot_OK» означает, что идентификатор устройства и версия ПО корректны и загрузка может быть продолжена.</p> <p>Ответ «OldVer» означает, что устройство использует более новую версию ПО и загрузка не может быть продолжена. Вместе со строкой «OldVer» загрузчик также отправляет используемую версию для информирования пользователя.</p> <p>Ответ «DiffID» означает, что идентификаторы устройства различны, то есть защищенный образ встроенного ПО не предназначен для данного устройства и загрузка не может быть продолжена.</p>
«Clear»	<p>Команда «Clear» отправляется программатором после получения ответа «Boot_OK». При получении данной команды загрузчик осуществляет стирание области памяти, предназначенной для основной программы. В результате загрузчик отправляет один из следующих ответов: «Clear_OK», «Clear_ER».</p> <p>Ответ «Clear_OK» означает, что стирание выполнено успешно и загрузка может быть продолжена.</p> <p>Ответ «Clear_ER» означает, что в процессе выполнения стирания памяти возникли ошибки и загрузка не может быть продолжена.</p>

Продолжение таблицы 8

Команда	Описание команды
«Clear»	<p>Команда «Clear» отправляется программатором после получения ответа «Boot_OK». При получении данной команды загрузчик осуществляет стирание области памяти, предназначенной для основной программы. В результате загрузчик отправляет один из следующих ответов: «Clear_OK», «Clear_ER».</p> <p>Ответ «Clear_OK» означает, что стирание выполнено успешно и загрузка может быть продолжена.</p> <p>Ответ «Clear_ER» означает, что в процессе выполнения стирания памяти возникли ошибки и загрузка не может быть продолжена.</p>
Блок защищенного образа	<p>Блоки защищенного образа отправляется программатором после получения ответа «Clear_OK». При получении блока защищенного образа загрузчик осуществляет расшифрование, проверку и запись блока в память. В результате загрузчик отправляет один из следующих ответов: «Line_OK», «Line_ER».</p> <p>Ответ «Line_OK» означает, что обработка блока выполнена успешно и загрузка может быть продолжена.</p> <p>Ответ «Line_ER» означает, что в процессе обработки блока возникли ошибки и загрузка не может быть продолжена.</p>
«TCheck»	<p>Команда «TCheck» отправляется программатором после успешной передачи всех блоков защищенного образа. Вместе с командой программатором отправляется имитовставка. При получении данной команды загрузчик осуществляет проверку имитовставки и отправляет один из следующих ответов: «TCheck_OK», «TCheck_ER».</p> <p>Ответ «TCheck_OK» означает, что отправленная имитовставка совпала с имитовставкой, выработанной от полученных данных.</p> <p>Ответ «TCheck_ER» означает, что отправленная имитовставка не совпала с имитовставкой, выработанной от полученных данных.</p>

Была составлена блок-схема алгоритма реализации процесса загрузки со стороны программатора, показанная на рисунках А.5, А.6 в приложении А.

5.5 Разработка алгоритма загрузчика

Был разработан следующий алгоритм загрузчика.

После запуска программа в течении 3 секунд ожидает приема команды начала загрузки. Если данная команда не была принята, то осуществляется проверка наличия основной программы в памяти микроконтроллера. В случае обнаружения основной программы происходит переход к её выполнению, иначе продолжается ожидание приема команды начала загрузки «Boot». Если загрузчик принимает команду начала загрузки, то начинается процесс загрузки. При его удачном завершении происходит переход к выполнению основной программы, иначе происходит возврат к ожиданию приема команды начала загрузки.

Была составлена блок-схема алгоритма реализации процесса загрузки со стороны загрузчика, показанная на рисунке А.7 в приложении А.

6 Реализация программного комплекса

6.1 Реализация алгоритма шифрования ГОСТ Р 34.12—2015 «Магма»

Алгоритм был реализован на языке программирования «Си». В реализации использовались 32-битные переменные, так как в соответствии с требованиями реализация должна быть рассчитана на использование в системах с разрядностью процессора 32 или более.

Для удобства в программе были использованы следующие переопределения типов и макроподстановки:

```
#define MAGMA_BLOCK_SIZE_INT32 2
typedef uint32_t magma_block_t[MAGMA_BLOCK_SIZE_INT32]; // Блок шифра
#define MAGMA_KEY_SIZE_INT32 8
typedef uint32_t magma_key_t[MAGMA_KEY_SIZE_INT32]; // Ключ шифрования
```

Были созданы переменные для хранения итерационных ключей и таблицы подстановок:

```
uint32_t RoundKey[32]; // Итерационные ключи шифрования
static unsigned char Pi[8][16] =
{
    { 1, 7, 14, 13, 0, 5, 8, 3, 4, 15, 10, 6, 9, 12, 11, 2 },
    { 8, 14, 2, 5, 6, 9, 1, 12, 15, 4, 11, 0, 13, 10, 3, 7 },
    { 5, 13, 15, 6, 9, 2, 12, 10, 11, 7, 8, 1, 4, 3, 14, 0 },
    { 7, 15, 5, 10, 8, 1, 6, 13, 0, 9, 3, 14, 11, 4, 2, 12 },
    { 12, 8, 2, 1, 13, 4, 15, 6, 7, 0, 10, 5, 3, 14, 9, 11 },
    { 11, 3, 5, 8, 2, 15, 10, 13, 14, 1, 7, 4, 12, 9, 6, 0 },
    { 6, 8, 2, 3, 9, 10, 5, 12, 1, 14, 4, 7, 11, 13, 0, 15 },
    { 12, 4, 6, 2, 10, 5, 11, 9, 14, 8, 13, 7, 0, 3, 15, 1 }
};
```

Алгоритм развертывания ключа был реализован в виде следующей функции:

```

inline void Magma_CalcRoundKey(const uint32_t *key)
{
    for (int j = 0; j < 3; j++)
        for (int i = 0; i < 8; i++)
            RoundKey[j * 8 + i] = key[i]; // Формируется 24
итерационных ключа с использованием части ключа в прямом порядке
    for (int i = 0; i < 8; i++)
        RoundKey[24 + i] = key[7 - i]; // В обратном порядке
}

```

Данная функция в качестве аргумента принимает 256-битного ключ `key`. На основе данного ключа 32 итерационных ключа `RoundKey`. Для этого исходный ключ делится на восемь 32-битных подключей, и далее они используются в следующем порядке: три раза с первого по восьмой и один раз с восьмого по первый.

Была реализована функция инициализации алгоритма шифрования:

```

void Magma_Init(const magma_key_t *key)
{
    if (key == NULL) return;
    Magma_CalcRoundKey(key);
}

```

Данная функция включает проверку полученного ключа и формирование итерационных ключей.

Было реализовано преобразование T в виде следующей функции:

```

inline uint32_t Magma_T(const uint32_t input)
{
    return ((uint32_t)(Pi[0][(input & 0xf0000000) >> 28]) << 28 |
            (uint32_t)(Pi[1][(input & 0x0f000000) >> 24]) << 24 |
            (uint32_t)(Pi[2][(input & 0x00f00000) >> 20]) << 20 |
            (uint32_t)(Pi[3][(input & 0x000f0000) >> 16]) << 16 |
            (uint32_t)(Pi[4][(input & 0x0000f000) >> 12]) << 12 |
            (uint32_t)(Pi[5][(input & 0x00000f00) >> 8]) << 8 |
            (uint32_t)(Pi[6][(input & 0x000000f0) >> 4]) << 4 |

```

```

        (uint32_t)(Pi[7][input & 0x0000000f]));
    }

```

В данной функции осуществляется разбиение входных данных на 4-битные фрагменты, их замена в соответствие с таблицей подстановок и их объединение в 32-битный фрагмент.

Преобразование g было реализовано в виде следующей функции:

```

inline uint32_t Magma_g(const uint32_t RoundKey, const uint32_t
input)
{
    uint32_t temp32 = Magma_T(input + RoundKey); // Сложение по
модулю 32 правой части блока с итерационным ключом
    return (temp32 << 11) | (temp32 >> 21); // Циклически сдвиг
на 11 разрядов влево
}

```

В данной функции выполняется сложение блока входных данных по модулю 32 правой с итерационным ключом. Выполняется преобразование T при помощи ранее реализованной функции $Magma_T$ и циклический сдвиг результата на 11 разрядов влево.

Преобразование G было реализовано в виде следующей функции:

```

inline void Magma_G(const uint32_t RoundKey, uint32_t *InputBlock,
uint32_t *output)
{
    uint32_t temp = InputBlock[0]; //для сохранения InputBlock[0]
    output[0] = InputBlock[1];
    output[1] = temp ^ Magma_g(RoundKey, InputBlock[1]);
}

```

В данной функции в левую часть обрабатываемого блока записывается значение правого, а в правый - результат покомпонентного сложения по модулю 2 левой части и результата работы функции $Magma_g$. Правая часть блока предварительно сохраняется в переменную $temp$.

Последнее преобразование G было реализовано в виде следующей функции:

```
inline void Magma_G_Fin(const uint32_t RoundKey, uint32_t
*InputBlock, uint32_t *output)
{
    output[0] = InputBlock[0] ^ Magma_g(RoundKey, InputBlock[1]);
    output[1] = InputBlock[1];
}
```

Зашифрование блока было реализовано в виде следующей функции:

```
void MagmaEncryptBlock(const uint32_t *InputBlock, uint32_t
*OutputBlock)
{
    Magma_G(RoundKey[0], InputBlock, OutputBlock); // Первое
преобразование G
    for (int i = 1; i < 31; i++)
        Magma_G(RoundKey[i], OutputBlock, OutputBlock); //
Последующие (со второго по тридцать первое) преобразования G
    Magma_G_Fin(RoundKey[31], OutputBlock, OutputBlock); //
Последнее (тридцать второе) преобразование G
}
```

Расшифрование блока было реализовано в виде следующей функции:

```
void MagmaDecryptBlock(const uint32_t *InputBlock, uint32_t
*OutputBlock)
{
    Magma_G(RoundKey[31], InputBlock, OutputBlock);
    for (int i = 30; i > 0; i--)
        Magma_G(RoundKey[i], OutputBlock, OutputBlock);
    Magma_G_Fin(RoundKey[0], OutputBlock, OutputBlock);
}
```

Таким образом, весь алгоритм был реализован в виде двух файлов cipher_magma.c и cipher_magma.h. В файл cipher_magma.h были вынесены объявления функций программного интерфейса, необходимые для

работы с алгоритмом. Полный текст программы приведен в составе текста программы загрузчика в приложении Г.

Алгоритм был реализован на языке программирования «С#» в виде класса *Magma*. Функции, реализованные ранее на языке программирования «Си» были преобразованы в методы данного класса. Полный текст на языке программирования «С#» программы приведен в составе текста программы генератора защищенного образа в приложении Б.

6.2 Реализация режима работы блочных шифров Multilinear Galois Mode

Алгоритм был реализован на языке программирования «Си». При реализации учитывалось, что считывание и запись файла встроенного ПО должна производиться построчно и, из-за ограничения памяти, шифруемое сообщение не должно храниться в оперативной памяти целиком. Для реализации этого требования вычисление сумм, необходимых для выработки имитовставки, производилось поэтапно вместе с шифрованием открытого текста *P* и обработкой дополнительных имитозащищаемых данных *A*.

Для удобства применения алгоритм был разделен на пять основных функции:

- функция инициализации;
- функция шифрования;
- функция вычисления суммы для имитовставки;
- функция выработки имитовставки;
- функция проверки имитовставки.

Функция инициализации предназначена для вызова перед началом обработки сообщения (дополнительных имитозащищаемых данных и открытого текста). Она включает инициализацию используемых переменных и алгоритма шифрования «Магма».

Функция шифрования предназначена для зашифрования или расшифрования блока открытого текста. В режиме зашифрования данная функция должна быть вызвана перед функцией вычисления суммы для имитовставки, так как для работы последней требуется блок шифротекста.

Функция вычисления суммы для имитовставки предназначена для подсчета промежуточного значения суммы, используемой для выработки имитовставки в результате обработки блока дополнительных имитозащищаемых данных или открытого текста.

Функция выработки имитовставки предназначена для получения значения имитовставки. В режиме зашифрования она должна вызываться после обработки всего сообщения функцией вычисления суммы для имитовставки.

Функция проверки имитовставки предназначена для получения значения имитовставки в режиме расшифрования и сравнения полученной имитовставки с проверяемой имитовставкой.

Для хранения данных, используемых в процессе работы алгоритма была создана структура, приведенная ниже.

```
typedef struct {  
    magma_key_t k;  
    magma_block_t nonce;  
    magma_block_t sum;  
    magma_block_t Y_i;  
    magma_block_t Z_i;  
    uint32_t Zcount;  
    uint32_t Pcount;  
} T_MGM;  
T_MGM MGM_context;
```

Переменная `k` используется для хранения ключа шифрования. Переменная `nonce` применяется для хранения значения уникального вектора. Переменная `sum` используется для хранения значения суммы, используемого для выработки имитовставки. Переменные `Y_i` и `Z_i` предназначены для хранения

значений Y и Z , выработанных на текущей итерации алгоритма. Переменная `Zcount` используется для хранения номера текущей итерации выполнения алгоритма подсчета суммы, необходимой для выработки имитовставки. Переменная `Pcount` используется для хранения номера текущей итерации выполнения алгоритма шифрования.

Были реализованы вспомогательные функции, необходимые для работы основных функций.

Функция, выполняющая покомпонентное сложение по модулю 2 двух 64-битных блоков, приведена ниже.

```
inline uint32_t *MGM_XOR64(const uint32_t *inp1, const uint32_t
*inp2, uint32_t *output)
{
    output[0] = inp1[0] ^ inp2[0];
    output[1] = inp1[1] ^ inp2[1];
    return output;
}
```

Функция, выполняющая умножение в поле $GF(2^{64})$ приведена ниже.

```
inline uint32_t *MGM_gf64_mul(uint32_t *inp1, uint32_t *inp2,
uint32_t *output)
{
    uint32_t inp1_l = inp1[0];
    uint32_t inp1_r = inp1[1];
    uint32_t inp2_l = inp2[0];
    uint32_t inp2_r = inp2[1];
    uint32_t x_MSB;
    output[0] = 0;
    output[1] = 0;
    for (int i = 0; i < 64; i++) {
        if (inp2_r & 0x1)
        {
            output[0] ^= inp1_l;
            output[1] ^= inp1_r;

```

```

    }
    //Сдвиг inp2 на 1 бит вправо
    inp2_r = ((inp2_r >> 1) | ((inp2_l & 0x1) << 31));
    inp2_l = inp2_l >> 1;
    x_MSB = inp1_l & 0x80000000;
    //Сдвиг inp1 на 1 бит влево
    inp1_l = ((inp1_l << 1) | ((inp1_r & 0x80000000) >>
31));

    inp1_r = inp1_r << 1;
    if (x_MSB) inp1_r ^= 0x1B;
}
return output;
}

```

В данной функции был применен алгоритм, основанный на приведении по модулю после каждого шага алгоритма. Алгоритм выполняется за 64 итерации. На каждом шаге алгоритма осуществляется умножение аргумента `inp1` на младший член аргумента `inp2 - inp2_r & 0x1`, после чего производится битовый сдвиг `inp2` на 1 бит вправо. Таким образом осуществляется пошаговое перемножение многочленов. После умножения выполняется битовый сдвиг `inp1` на 1 бит влево, при необходимости, выполняется приведение `inp1` по модулю путем его покомпонентного сложения по модулю 2 с битовой строкой, соответствующей порождающему полиному в данном поле. Для предотвращения потери старшего бита `inp1` после применения к нему операции битового сдвига данный бит предварительно сохраняется в переменную `x_MSB`.

Реализация функций `incrr` и `incrl` приведены ниже.

```

inline uint32_t *MGM_incr_l(uint32_t *input)
{
    input[0]++;
    return input;
}

```

```

inline uint32_t *MGM_incr_r(uint32_t *input)
{
    input[1]++;
    return input;
}

```

Так как в реализации используются 32-битные переменные, суммирование переменных равносильно сложению по модулю 2^{32} , поэтому в данных функциях осуществляется простое инкриминирование соответствующих частей входных блоков.

Реализация функции MSB приведена ниже.

```

inline uint32_t *MGM_MSB(uint32_t *InpBlock, uint32_t
InpBlockLengh)
{
    if (InpBlockLengh < 8)
    {
        if (InpBlockLengh <= 4)
        {
            uint32_t tmp = (4 - InpBlockLengh) * 8;
            InpBlock[1] = 0;
            InpBlock[0] = (InpBlock[0] >> tmp) << tmp;
        }
        else
        {
            uint32_t tmp = (8 - InpBlockLengh) * 8;
            InpBlock[1] = (InpBlock[1] >> tmp) << tmp;
        }
    }
    return InpBlock;
}

```

Была реализована функция инициализации MGM_Init. Данная функция принимает на вход ключ шифрования K, уникальный вектор попсе, сохраняет эти данные для дальнейшего использования. В функции происходит обнуление

переменных, используемых в процессе работы алгоритма и инициализирует алгоритма шифрования «Магма» с помощью вызова функции Magma_Init.

Реализация данной функции приведена ниже.

```
void MGM_Init(const uint32_t *K, const uint32_t *nonce)
{
    memcpy(MGM_context.k, K, 32);
    memcpy(MGM_context.nonce, nonce, 8);
    memset(MGM_context.sum, 0, 8);
    MGM_context.Zcount = 0;
    MGM_context.Pcount = 0;
    Magma_Init(MGM_context.k);
}
```

Была реализована функция шифрования блока данных. Реализация данной функции приведена ниже.

```
void MGM_CipherBlock(const uint32_t *InpBlock, uint32_t
*OutpBlock, uint32_t Block_lengh)
{
    uint32_t temp[2], H_i[2];
    if (MGM_context.Pcount >= MGM_MAX_LENGTH) return; //ошибка
    if (MGM_context.Pcount == 0)
    {
        MagmaEncryptBlock(MGM_context.nonce,
MGM_context.Y_i); //y1 = Ek(0 || nonce)
        MagmaEncryptBlock(MGM_context.Y_i, temp); // E_K(Y_1)
    }
    else
    {
        MGM_incr_r(MGM_context.Y_i); // Y_i = incr_r(Y_{i-1})
        MagmaEncryptBlock(MGM_context.Y_i, temp); //E_K(Y_i)
    }
    MGM_context.Pcount++;
    MGM_MSB(MGM_XOR64(temp, InpBlock, OutpBlock), Block_lengh);
}
```

В данной функции осуществляются проверка длины обработанной части сообщения. Если длина не превышает максимальное значение, то выполняется шифрование блока данных в соответствии с алгоритмом режима Multilinear Galois Mode. В функции используются реализованные ранее вспомогательные функции, а также функция шифрования блока из реализации алгоритма ГОСТ Р 34.12—2015 «Магма».

Была реализована функция вычисления суммы для имитовставки. Реализация данной функции приведена ниже.

```
void MGM_ProcBlock(const uint32_t *InpBlock)
{
    uint32_t temp[2], H_i[2];
    if (MGM_context.Zcount >= MGM_MAX_LENGTH) return; //ошибка
    if (MGM_context.Zcount++ == 0)
    {
        memcpy(temp, MGM_context.nonce, 8); // temp = 0 || поспе
        temp[0] |= 0x80000000; // 1 || nonce
        MagmaEncryptBlock(temp, MGM_context.Z_i); //Z_1 = E_K(1
        || ICN)
        MagmaEncryptBlock(MGM_context.Z_i, H_i); //H_i =
        E_K(Z_i)
    }
    else
    {
        MGM_incr_1(MGM_context.Z_i); // Z_{i+1} = incr_1(Z_i)
        MagmaEncryptBlock(MGM_context.Z_i, H_i); //H_i =
        E_K(Z_i)
    }
    MGM_gf64_mul(H_i, InpBlock, temp);
    MGM_XOR64(MGM_context.sum, temp, MGM_context.sum);
    MGM_context.Zcount++;
}
```

В данной функции осуществляются проверка длины обработанной части сообщения. Если длина не превышает максимальное значение, то выполняется обработка блока данных в соответствии с алгоритмом режима Multilinear Galois Mode: вычисляются значения Z_i и H_i , подсчитывается промежуточное значение суммы, используемой для выработки имитовставки. В функции используются реализованные ранее вспомогательные функции, а также функция шифрования блока из реализации алгоритма ГОСТ Р 34.12—2015 «Магма».

Была реализована функция выработки имитовставки. Реализация данной функции приведена ниже.

```
void MGM_GenerateT(uint32_t A_length, uint32_t P_length, uint32_t
*T)
{
    uint32_t temp[2], temp2[2], H_i[2];
    MGM_incr_l(MGM_context.Z_i); // Z_{i+1} = incr_l(Z_i)
    MagmaEncryptBlock(MGM_context.Z_i, H_i); //H_{h+q+1}
    temp[0] = A_length * 8;
    temp[1] = P_length * 8;
    MGM_gf64_mul(H_i, temp, temp2); // H_{h+q+1} (x) || (len(A)
|| len(C))
    MagmaEncryptBlock(MGM_XOR64(MGM_context.sum, temp2,
MGM_context.sum), T);
}
```

Была реализована функция проверки имитовставки. Реализация данной функции приведена ниже.

```
uint8_t MGM_CheckT(uint32_t A_length, uint32_t P_length, uint32_t
*T)
{
    uint32_t NewT[2];
    MGM_GenerateT(A_length, P_length, NewT);
    if ((NewT[0] == T[0]) && (NewT[1] == T[1])) return TRUE;
    else return FALSE;
}
```

В данной функции осуществляются выработка имитовставки с помощью функции `MGM_GenerateT` и сравнения полученной имитовставки с имитовставкой, поданной на вход функции.

Таким образом, весь алгоритм был реализован в виде двух файлов `mgm.c` и `mgm.h`. В файл `mgm.h` были вынесены объявления функций программного интерфейса, необходимые для работы с алгоритмом. Полный текст программы приведен в составе текста программы загрузчика в приложении Г.

Алгоритм был реализован на языке программирования «С#» в виде класса `MGM`. Функции, реализованные ранее на языке программирования «Си» были преобразованы в методы данного класса. Полный текст программы на языке программирования «С#» приведен в составе текста программы генератора защищенного образа в приложении Б.

6.3 Реализация генератора защищенного образа

Разработанный алгоритм генератора защищенного образа был реализован на языке программирования «С#» в виде приложения `Windows Forms`. В программе были использованы реализованные ранее классы `Magma` и `MGM`. Остальная часть алгоритма была реализована в виде двух классов: `SFI_Generator` и `Form1`. Класс `SFI_Generator` предназначен для проверки входных данных, их обработки и записи в файл защищенного образа встроенного ПО.

В классе `SFI_Generator` были созданы следующие поля:

```
public string FirmwareFileName, ImageDirName;
public bool KeyFileSelected, FirmwareFileSelected,
ImageDirSelected;
private uint CounterLine, Cblock_lengh;
private uint[] Key = new uint[8];
```

Поля `FirmwareFileName`, `ImageDirName` предназначены для хранения пути расположения файлов встроенного ПО и создаваемого защищенного образа.

Поля `KeyFileSelected`, `FirmwareFileSelected`, `ImageDirSelected` являются флагами выбора соответствующих путей расположения.

В классе `SFI_Generator` были реализованы следующие вспомогательные методы:

- `StrHexToByteArray` – осуществляет преобразование шестнадцатеричной строки (тип `String`) в массив значений типа `Byte`;
- `ByteToUint32` – осуществляет преобразование массива значений типа `Byte` в тип `Uint32`;
- `ByteToUint32Array` – осуществляет преобразование массива значений типа `Byte` в массив значений типа `Uint32`;
- `StringToUint32Array` – осуществляет преобразование строки (тип `String`) в массив значений типа `Uint32`;
- `Uint32ToByte` – осуществляет преобразование значения типа `Uint32` в массив значений типа `Byte`.

Чтение и проверка ключа были реализованы в виде метода `ReadKey`. В данном методе осуществляется чтение ключа из указанного файла, проверка его длины и его сохранение в поле `Key`. В случае несоответствия длины ключа вызывается исключение. При успешной записи ключа выставляется флаг `KeyFileSelected`.

Чтение и проверка файла встроенного ПО были реализованы в виде метода `ReadFirmware`. В данном методе осуществляется чтение встроенного ПО и его проверка в соответствие с алгоритмом.

Обработка идентификатора устройства, версии ПО и формирование заголовка защищенного образа было реализовано в виде метода ProcSupportInfo.

Обработка строки входных данных была реализована в виде метода ProcLine. В данном методе с помощью вспомогательных методов осуществляется преобразование шестнадцатеричной строки в массив значений типа UInt32, который обрабатывается методами класса MGM. После шифрования данные записываются в файл защищенного образа.

Создание защищенного образа было реализовано в виде метода EncryptFirmware. В данном методе с помощью ранее реализованных методов осуществляется преобразование входных данных в файл защищенного образа в соответствии с алгоритмом.

С помощью конструктора Windows Forms был создан пользовательский интерфейс генератора защищенного образа, который показан на рисунке 6.

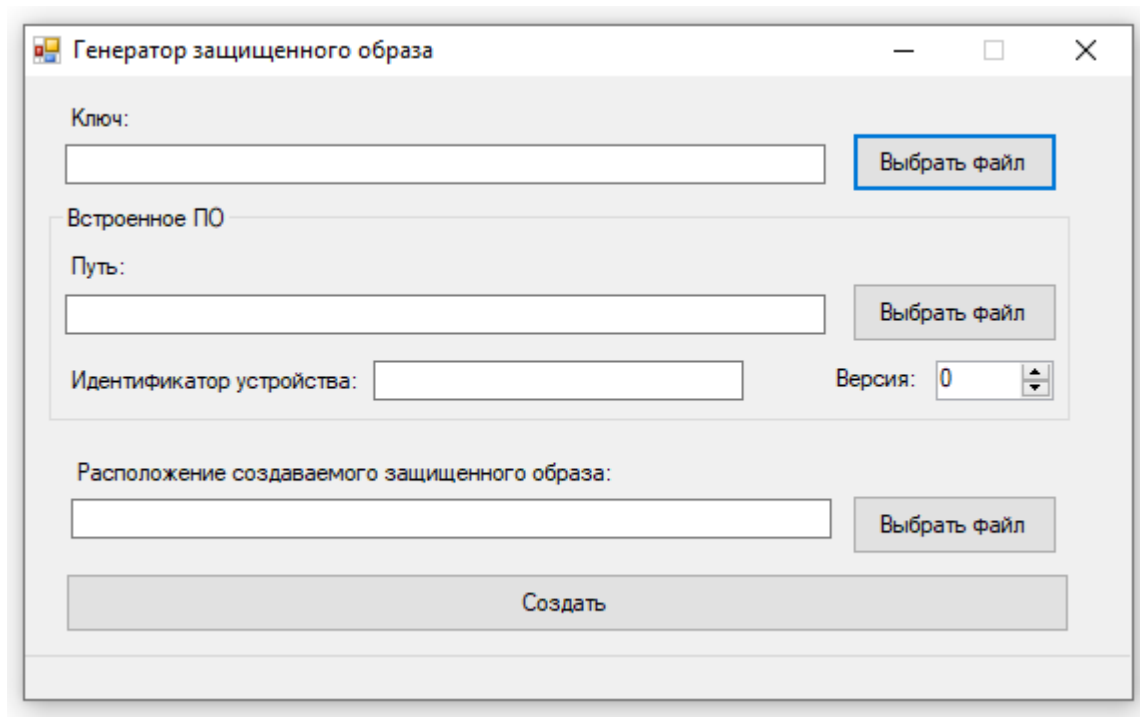


Рисунок 6 – Пользовательский интерфейс генератора защищенного образа

Работа элементов пользовательского интерфейса и их взаимодействие с функционалом класса `SFI_Generator` было реализовано в виде класса `Form1`.

В классе `Form1` были реализованы методы для обработки нажатия кнопок выбора файлов: `button1_Click`, `button2_Click`, `button3_Click`.

Для ввода файлов в программе использовались компоненты `TextBox` и `OpenFileDialog`. Класс `OpenFileDialog` использовался для фильтрации файлов и проверки наличия указанного файла. Была реализована возможность выбора файла, как посредством ввода пути файла в `TextBox` и его последующей проверки методами `OpenFileDialog`, так и выбор полностью с помощью `OpenFileDialog`.

Для указания каталога использовались компоненты `TextBox` и `folderBrowserDialog1`. Каталог может быть выбран посредством ввода пути файла в `TextBox`, так и с помощью иерархического графического представления.

Обработка исключений, вызываемых методами класса `SFI_Generator`, была реализована в методах класса `Form1`. Данная обработка включает вывод сообщений об ошибках с помощью `MessageBox`. Вывод статуса создания защищенного образа был реализован с помощью компонента `StatusStrip`.

Генератор защищенного образа был реализован в виде файлов `SFI_Generator.cs`, `Form1.cs`, `Program.cs`. Полный текст программы приведен в приложении Б.

6.4 Реализация программатора

Разработанный алгоритм программатора был реализован на языке программирования «C#» в виде приложения `Windows Forms`. Алгоритм был

реализован в виде следующих классов: `SFI_Reader`, `Receiver`, `SFI_Loader`, `COMPortInfo`, `ProcessConnection` и `Form1`.

Класс `SFI_Reader` реализует функции чтения и проверки файла защищенного образа встроенного ПО. В классе `SFI_Reader` были реализованы следующие методы:

- `UInt32ToByte` – осуществляет преобразование значения типа `UInt32` в массив значений типа `Byte`;
- `UInt32ToByteArray` – осуществляет преобразование массива значений типа `UInt32` в массив значений типа `Byte`;
- `CheckFile` – осуществляет проверка файла защищенного образа на соответствие формата;
- `GetSFI_Line` – осуществляет чтение, проверку блока защищенного образа;
- `GetSupportInfo` – осуществляет чтение, проверку заголовка защищенного образа;
- `GetSFI_T` – осуществляет чтение имитовставки из защищенного образа.

Класс `Receiver` реализует функцию приема данных от загрузчика. В методе `Receive` данного класса производится чтения буфера данных, полученных по интерфейсу стандарта RS-232. На основе маркерных символов («:» - начало блока, «\n – конец блока») производится сборка блока данных из полученных фрагментов.

Класс `SFI_Loader` предназначен для реализации процесса загрузки защищенного образа встроенного ПО. Класс содержит два основных метода `StartLoad` и `LoadProc`.

Метод `StartLoad` инициирует начало загрузки. Он содержит процедуры инициализации объектов классов, используемых в процессе загрузки (для

чтения файла защищенного образа, работы с последовательным портом и т. д.). В данном методе также осуществляется обработка возникших в процессе инициализации исключений.

Метод `LoadProc` реализует процедуры обмена данными в процессе загрузки. Данный метод рассчитан на использование двух таймеров: таймера ожидания ответа и таймера обработки данных. Таймер ожидания ответа используется для отсчета временного интервала, по истечению которого при отсутствии ответа от загрузчика определяется потеря связи. Запуск данного таймера осуществляется каждый раз после отправки команды. Таймер обработки данных используется для вызова метода `LoadProc`.

Класс содержит вспомогательный метод `LoadException`, предназначенный для вызова исключений, возникших в процессе загрузки.

Для преобразования шестнадцатеричной строки (тип `String`) в массив значений типа `Byte` был использован метод `StrHexToByteArray`.

Классы `COMPortInfo` и `ProcessConnection` предназначены для получения и вывода списка портов интерфейса стандарта RS-232. Метод `GetCOMPortsInfo` класса `COMPortInfo` формирует список доступных портов и информации о них.

С помощью конструктора `Windows Forms` был создан пользовательский интерфейс программатора, который показан на рисунке 7.

Работа элементов пользовательского интерфейса и их взаимодействие с функционалом описанных ранее классов было реализовано в виде класса `Form1`.

Возможность выбора порта интерфейса стандарта RS-232 была реализована с использованием компонента `ComboBox`.

Работа элементов пользовательского интерфейса и их взаимодействие с функционалом описанных ранее классов было реализовано в виде класса `Form1`.

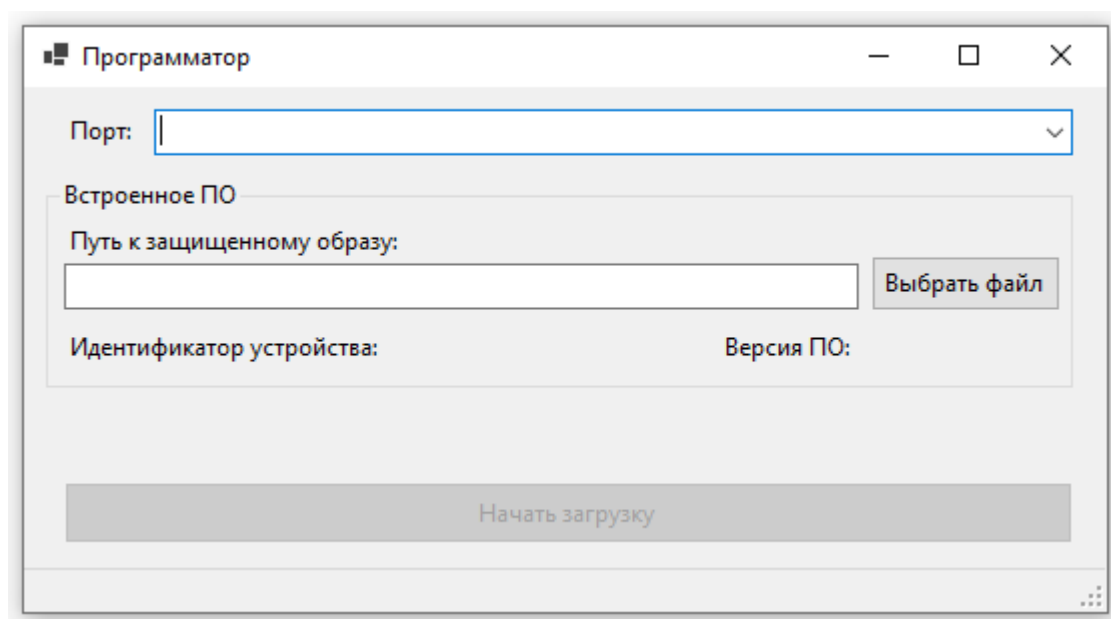


Рисунок 7 – Пользовательский интерфейс программатора

Возможность выбора порта интерфейса стандарта RS-232 была реализована с использованием компонента `ComboBox`.

Выбор файла защищенного образа был реализован с помощью компонентов `TextBox` и `OpenFileDialog`. Кроме вызова `OpenFileDialog` в обработчика начатая кнопки «Выбор файла» вызывается метод `CheckFile` класса `SFI_Reader`, который осуществляет проверку файла.

Кнопка «Начать загрузку» становится доступной только после выбора порта, файла и проверки файла.

Весь алгоритм был реализован в виде файлов `ComPortName.cs`, `BootLoader.cs` и `Form1`. Полный текст программы приведен в приложении В.

6.5 Реализация загрузчика

Разработанный алгоритм загрузчика был реализован на языке программирования «С». При реализации использовалась библиотека HAL (Hardware Abstraction Layer).

Для настройки тактирования и периферийных модулей микроконтроллера была использована программа. Настройка портов ввода-вывода показана на рисунке 8.

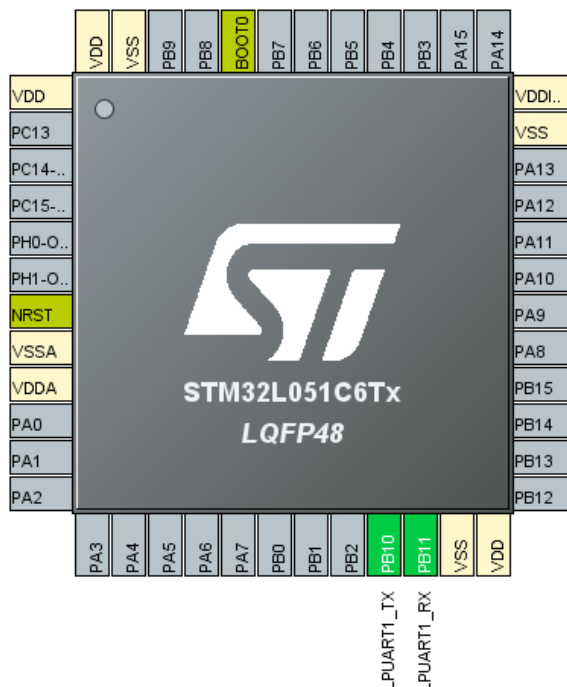


Рисунок 8 – Настройка портов ввода-вывода

Настройка тактирования показана на рисунке 9.

Настройка тактирования была реализована в виде функции `SystemClock_Config`. Тактирование производилось от встроенного генератора HSI на частоте 16 МГц.

Для обмена данными с программатором использовался интерфейс LPUART1, подключенный к портам PB10, PB11. Инициализация интерфейса была реализована в виде функции `InitLPUART`.

Для хранения данных, используемых в процессе загрузки была создана структура `Loader`.

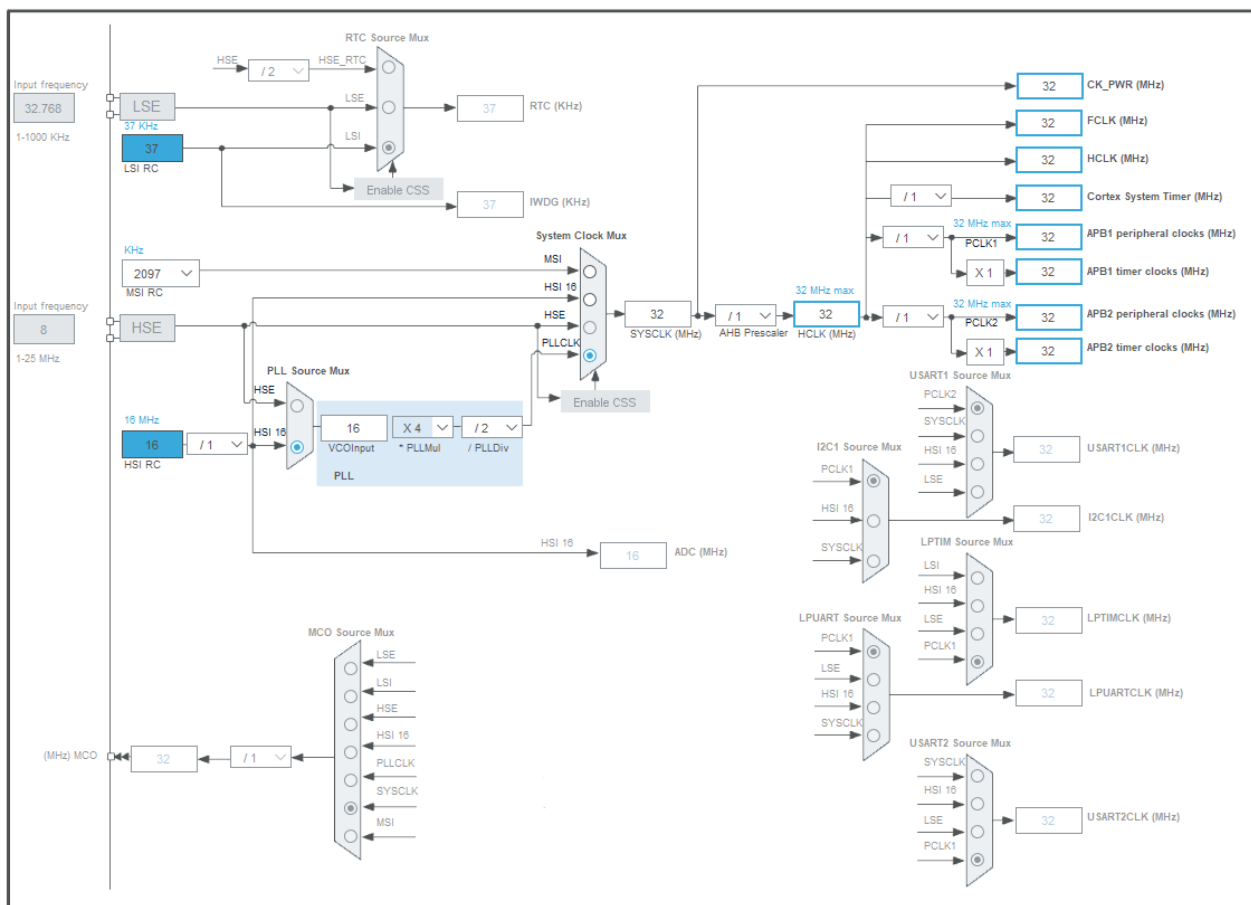


Рисунок 9 – Настройка тактирования

Для передачи и приема данных по LPUSART была создана структура USART. Она содержит Прием и передача данных осуществлялась по-байтово с помощью прерываний `UART_IT_RXNE`, `UART_IT_TC`, `UART_IT_ORE`. Обработка прерываний осуществлялась в функции `LPUART1_IRQHandler`.

Анализ полученного пакета данных был реализован в виде функции `Parser`. В данной функции определяется вид данных (тип полученной команды или блок защищенного образа) и производится запись данных в соответствующие переменные структуры `Loader`.

Обработка команд и данных, полученных в результате работы функции `Parser`, была реализована в виде функции `LoadProc`.

Обработка блок защищенного образа была реализована в виде функции `ProcBlock`. В данной функции выполняется преобразование типа данных, их

расшифровывание и обработка функциями `MGM_CipherBlock` и `MGM_ProcBlock`, описанными в параграфе 3.2.

Обработка расшифрованной записи ПО было реализовано в виде функции `ParserHex`. В данной функции выполняется анализ записи и запись фрагментов ПО в память микроконтроллера. Запись фрагментов полученного ПО производилась по блокам, размерами 32 байта, что соответствует половине размера страницы флеш-памяти. Для записи была использована функция `FLASH_If_Write`.

Для анализа строки использовалась функция `ihex_parse_single_record`. Она выполняет разделения записи ПО на фрагменты в соответствии со стандартном Intel HEX, определяет тип записи, проверяет длину записи и контрольную сумму. В результате работы функции заполняется структура `ihex_record_t`.

В программе был применен защитный механизм RDP (Code Read Out Protection) [8].

Весь алгоритм был реализован в виде файлов. Полный текст программы приведен в приложении Г.

7 Тестирование программного комплекса

7.1 Тестирование реализации алгоритма шифрования ГОСТ Р 34.12—2015

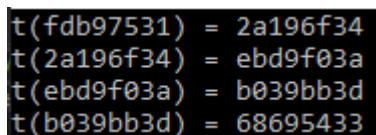
«Магма»

Было выполнено тестирование реализации шифрования ГОСТ Р 34.12—2015 «Магма». Для этого были использованы контрольные примеры из стандарта [5].

Было выполнено тестирование работы функции преобразования *t*. Для проведения тестирования была написана программа. Фрагмент программы, обеспечивающий вызов функции и вывод результатов её работы приведен ниже.

```
printf("t(%08x) = %08x\n", 0xfdb97531, Magma_T(0xfdb97531));  
printf("t(%08x) = %08x\n", 0x2a196f34, Magma_T(0x2a196f34));  
printf("t(%08x) = %08x\n", 0xebd9f03a, Magma_T(0xebd9f03a));  
printf("t(%08x) = %08x", 0xb039bb3d, Magma_T(0xb039bb3d));
```

Результат тестирования работы функции преобразования *t* приведен на рисунке 10.

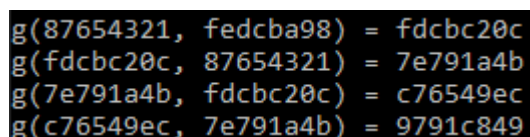


```
t(fdb97531) = 2a196f34  
t(2a196f34) = ebd9f03a  
t(ebd9f03a) = b039bb3d  
t(b039bb3d) = 68695433
```

Рисунок 10 - Результат тестирования работы функции преобразования *t*

Результат работы функции преобразования *t* совпал с описанным в стандарте. Был сделан вывод, что функция реализована верно.

Аналогично было выполнено тестирование работы функции преобразования *g*. Результат работы функции приведен на рисунке 11.



```
g(87654321, fedcba98) = fdcbc20c  
g(fdcbc20c, 87654321) = 7e791a4b  
g(7e791a4b, fdcbc20c) = c76549ec  
g(c76549ec, 7e791a4b) = 9791c849
```

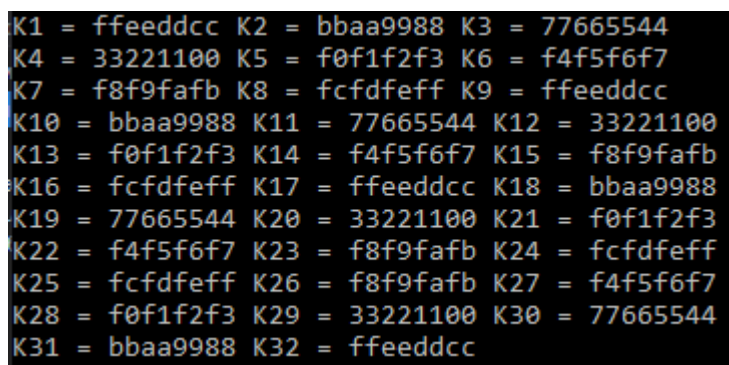
Рисунок 11 - Результат тестирования работы функции преобразования *g*

Результат работы функции преобразования g совпал с описанным в стандарте. Был сделан вывод, что функция реализована верно.

Было выполнено тестирование работы функции развертывания ключа. На вход функции был подан ключ, значение которого приведено ниже.

$K = \text{ffeeddccbbaa99887766554433221100f0f1f2f3f4f5f6f7f8f9fafbfcdfefff}$

Результат работы функции приведен на рисунке 12.



```
K1 = ffeeddcc K2 = bbaa9988 K3 = 77665544  
K4 = 33221100 K5 = f0f1f2f3 K6 = f4f5f6f7  
K7 = f8f9fafb K8 = fcdfefff K9 = ffeeddcc  
K10 = bbaa9988 K11 = 77665544 K12 = 33221100  
K13 = f0f1f2f3 K14 = f4f5f6f7 K15 = f8f9fafb  
K16 = fcdfefff K17 = ffeeddcc K18 = bbaa9988  
K19 = 77665544 K20 = 33221100 K21 = f0f1f2f3  
K22 = f4f5f6f7 K23 = f8f9fafb K24 = fcdfefff  
K25 = fcdfefff K26 = f8f9fafb K27 = f4f5f6f7  
K28 = f0f1f2f3 K29 = 33221100 K30 = 77665544  
K31 = bbaa9988 K32 = ffeeddcc
```

Рисунок 12 - Результат тестирования работы функции развертывания ключа

Результат работы функции развертывания ключа совпал с описанным в стандарте. Был сделан вывод, что функция реализована верно.

Было выполнено тестирование реализации алгоритма зашифрования. Для проведения тестирования в функцию шифрования был добавлен вывод значений промежуточных вычислений. Для зашифрования использовались итерационные ключи, полученные при тестировании работы функции развертывания ключа. На вход функции был подан открытый текст, значение которого приведено ниже.

$a = \text{fedcba9876543210}$

Результат работы функции приведен на рисунке 13.

Результат работы функции зашифрования совпал с описанным в стандарте. Был сделан вывод, что функция реализована верно.

Аналогичным методом было выполнено тестирование реализации алгоритма расшифрования. Шифртекст, подлежащий расшифрованию, был равен шифртексту, полученному при тестировании работы функции зашифрования.

```

(fedcba98, 76543210)
G[K2] = (76543210, 28da3b14)
G[K3] = (28da3b14, b14337a5)
G[K4] = (b14337a5, 633a7c68)
G[K5] = (633a7c68, ea89c02c)
G[K6] = (ea89c02c, 11fe726d)
G[K7] = (11fe726d, ad0310a4)
G[K8] = (ad0310a4, 37d97f25)
G[K9] = (37d97f25, 46324615)
G[K10] = (46324615, ce995f2a)
G[K11] = (ce995f2a, 93c1f449)
G[K12] = (93c1f449, 4811c7ad)
G[K13] = (4811c7ad, c4b3edca)
G[K14] = (c4b3edca, 44ca5ce1)
G[K15] = (44ca5ce1, fef51b68)
G[K16] = (fef51b68, 2098cd86)
G[K17] = (2098cd86, 4f15b0bb)
G[K18] = (4f15b0bb, e32805bc)
G[K19] = (e32805bc, e7116722)
G[K20] = (e7116722, 89cadf21)
G[K21] = (89cadf21, bac8444d)
G[K22] = (bac8444d, 11263a21)
G[K23] = (11263a21, 625434c3)
G[K24] = (625434c3, 8025c0a5)
G[K25] = (8025c0a5, b0d66514)
G[K26] = (b0d66514, 47b1d5f4)
G[K27] = (47b1d5f4, c78e6d50)
G[K28] = (c78e6d50, 80251e99)
G[K29] = (80251e99, 2b96eca6)
G[K30] = (2b96eca6, 05ef4401)
G[K31] = (05ef4401, 239a4577)
G[K32] = (239a4577, c2d8ca3d)

```

Рисунок 13 - Результат тестирования работы функции зашифрования

Для расшифрования использовались итерационные ключи, полученные при тестировании работы функции развертывания ключа. Результат работы функции приведен на рисунке 14.

Результат работы функции расшифрования совпал с описанным в стандарте. Был сделан вывод, что функция реализована верно.

В результате проведенного тестирования реализации шифрования ГОСТ Р 34.12—2015 «Магма» был сделан вывод, что программная реализация работает верно.

```

(4ee901e5, c2d8ca3d)
G[K31] = (c2d8ca3d, 239a4577)
G[K30] = (239a4577, 05ef4401)
G[K29] = (05ef4401, 2b96eca6)
G[K28] = (2b96eca6, 80251e99)
G[K27] = (80251e99, c78e6d50)
G[K26] = (c78e6d50, 47b1d5f4)
G[K25] = (47b1d5f4, b0d66514)
G[K24] = (b0d66514, 8025c0a5)
G[K23] = (8025c0a5, 625434c3)
G[K22] = (625434c3, 11263a21)
G[K21] = (11263a21, bac8444d)
G[K20] = (bac8444d, 89cadf21)
G[K19] = (89cadf21, e7116722)
G[K18] = (e7116722, e32805bc)
G[K17] = (e32805bc, 4f15b0bb)
G[K16] = (4f15b0bb, 2098cd86)
G[K15] = (2098cd86, fef51b68)
G[K14] = (fef51b68, 44ca5ce1)
G[K13] = (44ca5ce1, c4b3edca)
G[K12] = (c4b3edca, 4811c7ad)
G[K11] = (4811c7ad, 93c1f449)
G[K10] = (93c1f449, ce995f2a)
G[K9] = (ce995f2a, 46324615)
G[K8] = (46324615, 37d97f25)
G[K7] = (37d97f25, ad0310a4)
G[K6] = (ad0310a4, 11fe726d)
G[K5] = (11fe726d, ea89c02c)
G[K4] = (ea89c02c, 633a7c68)
G[K3] = (633a7c68, b14337a5)
G[K2] = (b14337a5, 28da3b14)
G[K1] = (28da3b14, 76543210)
A = fedcba9876543210

```

Рисунок 14 - Результат тестирования работы функции расшифрования

7.2 Тестирование реализации режима работы блочных шифров Multilinear Galois Mode

Было выполнено тестирование реализации режима работы блочных шифров Multilinear Galois Mode. Для этого были использованы контрольные примеры из соответствующих рекомендаций [6].

Для проведения тестирования были использованы параметры, приведенные ниже.

K: FF EE DD CC BB AA 99 88 77 66 55 44 33 22 11 00 F0 F1 F2
F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF;

```

nonce: 12 DE F0 6B 3C13 0A5 9;
A: 01 01 01 01 01 01 01 01 02 02 02 02 02 02 02 02 03 03 03
03 03 03 03 03 04 04 04 04 04 04 04 04 05 05 05 05 05 05 05 EA;
P: FF EE DD CC BB AA 99 88 11 22 33 44 55 66 77 00 88 99 AA
BB CC EE FF 0A 00 11 22 33 44 55 66 77 99 AA BB CC EE FF 0A00 11
22 33 44 55 66 77 88 AA BB CC EE FF 0A 00 11 22 33 44 55 66 77 88
99 AA BB CC.

```

Было выполнено тестирование работы функции зашифрования. Для проведения тестирования в функцию шифрования MGM_CipherBlock был добавлен вывод значений промежуточных вычислений. Предварительно была вызвана функция инициализации MGM_Init. Открытый текст был дополнен нулевыми значениями до длины кратной длине блока. Результат тестирования работы функции приведен на рисунке 15.

```

Y_1 = 5623890162de31bf E_K(Y_1) = 387bdba0e43439b3
Y_2 = 5623890162de31c0 E_K(Y_2) = 9433000610f7f2ae
Y_3 = 5623890162de31c1 E_K(Y_3) = 97b7aa6d73c58757
Y_4 = 5623890162de31c2 E_K(Y_4) = 9415528bffc9e80a
Y_5 = 5623890162de31c3 E_K(Y_5) = 03f768bff182d670
Y_6 = 5623890162de31c4 E_K(Y_6) = fd05f84e9b09d2fe
Y_7 = 5623890162de31c5 E_K(Y_7) = da4d908a95b175c4
Y_8 = 5623890162de31c6 E_K(Y_8) = 65997396dac24bd7
Y_9 = 5623890162de31c7 E_K(Y_9) = a900504a148dee26
C = c795066c5f9ea03b85113342459185ae1f2e00d6bf2b785d
940470b8bb9c8e7d9a5dd3731f7ddc70ec27cb0ace6fa57670f6
5c646abb75d547aa37c3bcb5c34e03bb9c0000000000

```

Рисунок 15 - Результат тестирования работы функции зашифрования

Результат работы функции зашифрования совпал с ожидаемым. Был сделан вывод, что функция реализована верно.

Было выполнено тестирование работы функции выработки значения имитовставки T от дополнительных имитозащищаемых данных A и шифртекста C. Для проведения тестирования в функции MGM_ProcBlock и MGM_GenerateT был добавлен вывод значений промежуточных вычислений. Дополнительные имитозащищаемые данные были дополнены нулевыми значениями до длины кратной длине блока. Результат тестирования работы функции приведен на рисунке 16.

```

Z_01 = 2b073f0494f372a0 H_01 = 708a78191cdd22aa
Z_02 = 2b073f0594f372a0 H_02 = 6f02cc464b2fa0a3
Z_03 = 2b073f0694f372a0 H_03 = 9f81f226fd196f05
Z_04 = 2b073f0794f372a0 H_04 = b9c2ac9be5b5dff9
Z_05 = 2b073f0894f372a0 H_05 = 74b5ec96551bf888
Z_06 = 2b073f0994f372a0 H_06 = 7eb021a4035b04c3
Z_07 = 2b073f0a94f372a0 H_07 = c2a9c3a8704d9bb0
Z_08 = 2b073f0b94f372a0 H_08 = f5d505a87b8383b5
Z_09 = 2b073f0c94f372a0 H_09 = f795e75fdeb8933c
Z_10 = 2b073f0d94f372a0 H_10 = 65a1a3e680f08145
Z_11 = 2b073f0e94f372a0 H_11 = 1c74a5764cb0d595
Z_12 = 2b073f0f94f372a0 H_12 = dc8447a514e783e7
Z_13 = 2b073f1094f372a0 H_13 = a7e3afe004ee16e3
Z_14 = 2b073f1194f372a0 H_14 = a5aabb0b7980d071
Z_15 = 2b073f1294f372a0 H_15 = 6e104cc933525c5d
len(A)||len(C) = 0000014800000218
T = a7928069aa10fd10

```

Рисунок 16 - Результат тестирования работы выработки значения имитовставки

Результат работы функции выработки значения имитовставки совпал с ожидаемым. Был сделан вывод, что функция реализована верно.

Аналогичным методом было выполнено тестирование реализации алгоритма расшифрования. Шифртекст, подлежащий расшифрованию, был равен шифртексту, полученному при тестирование работы функции зашифрования. предварительно была вызвана функция инициализации MGM_Init. Шифртекст был дополнен нулевыми значениями до длины кратной длине блока.

Результат тестирования работы функции приведен на рисунке 17.

Результат работы функции расшифрования совпал с ожидаемым. Был сделан вывод, что функция реализована верно.

```

Y_1 = 5623890162de31bf E_K(Y_1) = 387bdba0e43439b3
Y_2 = 5623890162de31c0 E_K(Y_2) = 9433000610f7f2ae
Y_3 = 5623890162de31c1 E_K(Y_3) = 97b7aa6d73c58757
Y_4 = 5623890162de31c2 E_K(Y_4) = 9415528bffc9e80a
Y_5 = 5623890162de31c3 E_K(Y_5) = 03f768bff182d670
Y_6 = 5623890162de31c4 E_K(Y_6) = fd05f84e9b09d2fe
Y_7 = 5623890162de31c5 E_K(Y_7) = da4d908a95b175c4
Y_8 = 5623890162de31c6 E_K(Y_8) = 65997396dac24bd7
Y_9 = 5623890162de31c7 E_K(Y_9) = a900504a148dee26
P = ffeeddccbbaa998811223344556677008899aabbccceeff0a
001122334455667799aabbccceeff0a001122334455667788aabb
ccceeff0a00112233445566778899aabbcc0000000000

```

Рисунок 17 - Результат тестирования работы функции расшифрования

Было выполнено тестирование работы функции проверки значения имитовставки. Результат тестирования работы функции приведен на рисунке 18.

```
Z_01 = 2b073f0494f372a0 H_01 = 708a78191cdd22aa
Z_02 = 2b073f0594f372a0 H_02 = 6f02cc464b2fa0a3
Z_03 = 2b073f0694f372a0 H_03 = 9f81f226fd196f05
Z_04 = 2b073f0794f372a0 H_04 = b9c2ac9be5b5dff9
Z_05 = 2b073f0894f372a0 H_05 = 74b5ec96551bf888
Z_06 = 2b073f0994f372a0 H_06 = 7eb021a4035b04c3
Z_07 = 2b073f0a94f372a0 H_07 = c2a9c3a8704d9bb0
Z_08 = 2b073f0b94f372a0 H_08 = f5d505a87b8383b5
Z_09 = 2b073f0c94f372a0 H_09 = f795e75fdeb8933c
Z_10 = 2b073f0d94f372a0 H_10 = 65a1a3e680f08145
Z_11 = 2b073f0e94f372a0 H_11 = 1c74a5764cb0d595
Z_12 = 2b073f0f94f372a0 H_12 = dc8447a514e783e7
Z_13 = 2b073f1094f372a0 H_13 = a7e3afe004ee16e3
Z_14 = 2b073f1194f372a0 H_14 = a5aabb0b7980d071
Z_15 = 2b073f1294f372a0 H_15 = 6e104cc933525c5d
len(A)||len(C) = 0000014800000218
TRUE
```

Рисунок 18 - Результат тестирования работы функции проверки значения имитовставки

Результат работы функции проверки значения имитовставки совпал с ожидаемым — проверка имитовставки была пройдена успешно. Был сделан вывод, что функция реализована верно.

7.3 Тестирование генератора защищенного образа

Был составлен набор тестов для проведения функционального тестирования генератора защищенного образа.

Для составления перечня тестов использовались следующие методы:

- анализ граничных значений;
- метод предположения об ошибке.

Полученный набор тестов приведен в таблице 9.

Таблица 9 — Набор тестов для тестирования генератора защищенного образа

№	Описание	Ожидаемый результат
1	Ввод файла, содержащего ключ размером 31 байт	Сообщение об ошибке
2	Ввод файла, содержащего ключ размером 32 байта	Успешная проверка файла
3	Ввод файла, содержащего ключ размером 33 байта	Сообщение об ошибке
4	Ввод файла ключа, имеющего расширение, отличное от «bin»	Ввод невозможен
5	Ввод номер версии 255	Значение введено
6	Ввод номер версии 256	Значение не введено
7	Ввод номер версии -1	Значение не введено
8	Ввести номер версии 0	Значение введено
9	Ввод идентификатора устройства длиной 16 символов	Значение введено
10	Ввод идентификатора устройства длиной 17 символов	Значение не будет введено
11	Ввод идентификатора устройства, содержащего не разрешенные символы	Сообщение об ошибке
12	Ввод файла встроенного ПО с измененным значением длины записи Intel HEX	Сообщение об ошибке
13	Ввод файла встроенного ПО, содержащего в записи Intel HEX символы, не используемые для шестнадцатеричных цифр	Сообщение об ошибке
14	Ввод файла встроенного ПО, имеющего расширение, отличное от «hex»	Ввод невозможен
15	Ввод файла встроенного ПО, имеющего размер 20 480 Байт	Успешная проверка файла
16	Ввод файла встроенного ПО, имеющего размер 20 481 Байт	Сообщение об ошибке

В соответствие с разработанным набором тестов было проведено тестирование программы. Для этого был подготовлен файл, содержащий ключ, и несколько файлов встроенного ПО. Результаты тестирования приведены в таблице 10.

Таблица 10 — Результаты тестирования генератора защищенного образа

№	Описание	Результат
1	Введен файл, содержащий ключ размером 31 байт	Сообщение об ошибке «Некорректная длина ключа»
2	Введен файл, содержащий ключ размером 32 байта	Успешная проверка файла
3	Введен файл, содержащий ключ размером 33 байта	Сообщение об ошибке «Некорректная длина ключа»
4	Введен файла ключа, имеющего расширение, отличное от «bin»	Ввод невозможен
5	Введен номер версии 255	Значение введено
6	Введен номер версии 256	Значение версии изменено на 255
7	Введен номер версии -1	Значение версии изменено на 0
8	Введен номер версии 0	Значение введено
9	Введен идентификатор устройства длиной 16 символов	Значение введено
10	Произведена попытка ввести идентификатор устройства длиной 17 символов	Значение не введено
11	Введен идентификатор устройства «A\B/»	Сообщение об ошибке «Некорректный идентификатор устройства»

Продолжение таблицы 10

№	Описание	Результат
12	Введен файл встроенного ПО с измененным значением длины в первой записи Intel HEX (значение изменено с «00» на «01»)	Сообщение об ошибке «Файл встроенного ПО не соответствует формату Intel HEX»
13	Введен файл встроенного ПО, содержащий в записи Intel HEX символы, не использующиеся для шестнадцатеричных цифр	Сообщение об ошибке «Файл встроенного ПО не соответствует формату Intel HEX»
14	Произведена попытка ввести файла встроенного ПО, имеющего расширение, отличное от «hex»	Ввод невозможен
15	Введен файл встроенного ПО, имеющий размер 20 480 Байт	Успешная проверка файла
16	Введен файл встроенного ПО, имеющий размер 20 481 Байт	Сообщение об ошибке «Некорректный идентификатор устройства»

Таким образом, результаты прогонов программы совпали с ожидаемыми, Был сделан вывод, что программа работает верно.

7.4 Тестирование программатора и загрузчика

Был составлен набор тестов для проведения тестирования программатора и загрузчика.

Для составления перечня тестов использовались следующие методы:

— анализ граничных значений;

— метод предположения об ошибке.

Полученный набор тестов приведен в таблице 11.

Таблица 11 — Набор тестов для тестирования программатора и загрузчика

№	Описание	Ожидаемый результат
1	Ввод файла защищенного образа, имеющего расширение, отличное от «sfi»	Ввод невозможен
2	Ввод файла защищенного образа с измененным значением длины блока (значение длины больше изначального)	Сообщение об ошибке
3	Ввод файла защищенного образа с измененным значением длины блока (значение длины меньше изначального)	Сообщение об ошибке
4	Ввод файла защищенного образа с измененным значением длины в заголовке (значение длины больше изначального)	Сообщение об ошибке
5	Ввод файла защищенного образа с измененным значением длины в заголовке (значение длины меньше изначального)	Сообщение об ошибке
6	Ввод файла защищенного образа, имеющего идентификатор устройства, отличный от идентификатора, записанного в памяти микроконтроллера (идентификатор изменен после создания защищенного образа)	Сообщение об ошибке
7	Ввод файла защищенного образа, созданного с использованием идентификатора устройства, отличного от идентификатора, записанного в памяти микроконтроллера, но имеющего в заголовке правильный идентификатор (заголовок изменен после создания образа)	Сообщение об ошибке

Продолжение таблицы 11

№	Описание	Ожидаемый результат
8	Ввод файла защищенного образа, имеющего более раннюю версию ПО, чем записанная в памяти микроконтроллера (версия ПО изменена после создания защищенного образа)	Сообщение об ошибке
9	Ввод файла защищенного образа, созданного с использованием ранней версии ПО, но имеющего в заголовке более позднюю версию ПО, чем записанная в памяти микроконтроллера (заголовок изменен после создания образа)	Сообщение об ошибке
10	Ввод файла защищенного образа с измененным порядком блоков	Сообщение об ошибке
11	Ввод файла защищенного образа, созданного с использованием ключа, отличного от ключа, записанного в памяти микроконтроллера	Сообщение об ошибке
12	Отключение питания устройства во время процесса загрузки	Сообщение об ошибке
13	Прекращение работы программы программатора во время процесса загрузки	Возможно повторно начать загрузку после перезагрузки
14	Начать процесс загрузки через 2 секунды после включения устройства	Успешное выполнение загрузки
15	Начать процесс загрузки через 4 секунды после включения устройства (основная программа присутствует в памяти микроконтроллера)	Сообщение об ошибке
16	Начать процесс загрузки через 4 секунды после включения устройства (основная программа отсутствует в памяти микроконтроллера)	Успешное выполнение загрузки

В соответствие с разработанным набором тестов было проведено тестирование программы. Результаты тестирования приведены в таблице 12.

Таблица 12 — Результаты тестирования программатора и загрузчика

№	Описание	Результат
1	Произведена попытка ввода файла защищенного образа, имеющего расширение, отличное от «txt»	Ввод невозможен
2	Введен файл защищенного образа с измененным значением длины блока (значение длины в первом блоке изменено с 2 на 4)	Сообщение об ошибке «Файл защищенного образа не соответствует требуемому формату»
3	Введен файл защищенного образа с измененным значением длины блока (значение длины в первом блоке изменено с 2 на 1)	Сообщение об ошибке «Файл защищенного образа не соответствует требуемому формату»
4	Введен файл защищенного образа с измененным значением длины в заголовке (значение длины изменено с 231 на 232)	Сообщение об ошибке «Файл защищенного образа не соответствует требуемому формату»
5	Введен файл защищенного образа с измененным значением длины в заголовке (значение длины изменено с 231 на 230)	Сообщение об ошибке «Ошибка проверки имитовставки»
6	Введен файл защищенного образа, имеющего идентификатор устройства, отличный от идентификатора, записанного в памяти микроконтроллера (идентификатор изменен на с «TestProg» на «Test»)	Сообщение об ошибке «Защищенный образ встроенного ПО не предназначен для данного устройства»

Продолжение таблицы 12

№	Описание	Результат
7	Введен файл защищенного образа, созданный с использованием идентификатора устройства, отличного от идентификатора, записанного в памяти микроконтроллера, но имеющего в заголовке правильный идентификатор (был создан файл защищенного образа с идентификатором устройства «Test», после чего идентификатор был изменен на «TestProg»)	Сообщение об ошибке «Ошибка проверки имитовставки»
8	Введен файл защищенного образа, имеющий более раннюю версию ПО, чем записанная в памяти микроконтроллера (была введена версия ПО 1, на устройстве версия ПО 2)	Сообщение об ошибке «Устройство использует более новую версию ПО: 2»
9	Введен файл защищенного образа, созданный с использованием ранней версии ПО, но имеющего в заголовке более позднюю версию ПО, чем записанная в памяти микроконтроллера (версия ПО в заголовке изменена с 1 на 3)	Сообщение об ошибке «Ошибка проверки имитовставки»
10	Введен файл защищенного образа с измененным порядком блоков	Сообщение об ошибке «Ошибка проверки имитовставки»
11	Введен файл защищенного образа, созданный с использованием ключа, отличного от ключа, записанного в памяти микроконтроллера	Сообщение об ошибке «Ошибка записи встроенного ПО»
12	Питания устройства было отключено во время процесса загрузки	Сообщение об ошибке «Потеря связи»

Продолжение таблицы 12

№	Описание	Результат
13	Было симитировано прекращение работы программы во время процесса загрузки, после чего устройство было перезагружено и загрузка была начата повторно	Успешное выполнение загрузки
14	Процесс загрузки был начат через 2 секунды после включения устройства	Успешное выполнение загрузки
15	Процесс загрузки был начат через 4 секунды после включения устройства (основная программа присутствует в памяти микроконтроллера)	Сообщение об ошибке «Потеря связи»
16	Процесс загрузки был начат через 4 секунды после включения устройства (основная программа отсутствует в памяти микроконтроллера)	Успешное выполнение загрузки

Таким образом, результаты прогонов программ совпали с ожидаемыми, Был сделан вывод, что программы работают верно.

8 Вредные психофизиологические факторы, влияющие на органы зрения специалиста по защите информации

8.1 Обзор вредных психофизиологических факторов

Работа специалиста по защите информации сопровождается влиянием на его организм вредных психофизиологических факторов.

Психофизиологические факторы — физические нервно-психические перегрузки, перенапряжение анализаторов, монотонность труда [9].

Профессиональная деятельность специалиста по защите информации в большинстве случаев предполагает продолжительную работу на персональном компьютере.

При выполнении работ на персональном компьютере (ПК) согласно ГОСТу 12.0.003-74 “ССБТ. Опасные и вредные производственные факторы. Классификация” могут иметь место следующие факторы [10]:

- повышенная температура поверхностей ПК;
- повышенная или пониженная температура воздуха рабочей зоны;
- выделение в воздух рабочей зоны ряда химических веществ;
- повышенная или пониженная влажность воздуха;
- повышенный или пониженный уровень отрицательных и положительных аэроионов;
- повышенное значение напряжения в электрической цепи, замыкание;
- повышенный уровень статического электричества;
- повышенный уровень электромагнитных излучений;
- повышенная напряженность электрического поля;
- отсутствие или недостаток естественного света;

- недостаточная искусственная освещенность рабочей зоны;
- повышенная яркость света;
- повышенная контрастность;
- прямая и отраженная блескость;
- зрительное напряжение;
- монотонность трудового процесса;
- нервно-эмоциональные перегрузки.

В частности, к факторам, наиболее сильно влияющим на зрение можно отнести следующие группы [11]:

- несовершенство способов создания изображения на экране монитора;
- непродуманная организация рабочего места.

Первая группа факторов включает в себя:

- неоптимальные параметры схем развертки мониторов;
- несовместимость параметров монитора и графического адаптера;
- недостаточно высокое разрешение монитора, расфокусировка;
- несведение лучей и низкий уровень других его технических характеристик;
- избыточная или недостаточная яркость изображения.

Непродуманная организация рабочего места может стать причиной:

- наличия бликов на лицевой панели экрана;
- отсутствия необходимого уровня освещенности рабочих мест;
- несоблюдения расстояния от глаз до экрана.

Работа на ПК сопровождается постоянным и значительным напряжением функций зрительного анализатора. Одной из основных особенностей является иной принцип чтения информации, чем при обычном чтении. При обычном

чтении текст на бумаге, расположенный горизонтально на столе, считывается работником с наклоненной головой при падении светового потока на текст. При работе на ПК оператор считывает текст, почти не наклоняя голову, глаза смотрят прямо или почти прямо вперед, текст (источник — люминесцирующее вещество экрана) формируется по другую сторону экрана, поэтому пользователь не считывает отраженный текст, а смотрит непосредственно на источник света, что вынуждает глаза и орган зрения в целом работать в несвойственном ему стрессовом режиме длительное время [12].

Расстройство органов зрения резко увеличивается при работе более четырех часов в день. Всемирная организация здравоохранения (ВОЗ) ввела понятие “компьютерный зрительный синдром” (КЗС), типовыми симптомами которого являются жжение в глазах, покраснение век и конъюнктивы, чувство инородного тела или песка под веками, боли в области глазниц и лба, затуманивание зрения, замедленная перефокусировка с ближних объектов на дальние [12].

Таким образом, были определены вредные психофизиологические факторы, влияющие на органы зрения специалиста по защите информации.

8.2 Методы и средства физической культуры, снижающие воздействие вредных психофизиологических факторов

Для снижения воздействия вредных психофизиологических факторов, влияющих на органы зрения специалиста по защите информации применяется специальная гимнастика. Данная гимнастика включает набор упражнений, способствующих минимизации вредного воздействия работ на персональном компьютере [13].

Упражнения необходимо выполнять каждые два часа при работе с компьютером. Движения должны быть плавные, без рывков, также полезно между упражнениями поморгать.

Первое упражнение выполняется сидя. Необходимо крепко зажмурить глаза на 5 секунд, а затем открыть глаза на 5 секунд, данную последовательность действий повторить 8 раз. Описанное упражнение укрепляет мышцы век, способствует кровообращению и расслаблению мышц глаз.

Следующее упражнение выполняется стоя. Необходимо смотреть прямо перед собой в течение 3 секунд, поставить палец правой руки на средней линии лица на расстоянии 30 сантиметров от глаза, перевести взгляд на конец пальца и смотреть на него 5 секунд, опустить руки. Данную последовательность действий повторить 10 раз. Описанное упражнение снимает утомление, облегчает зрительную работу на близком расстоянии.

Следующее упражнение выполняется сидя. Необходимо быстро моргать в течение 2 минут. Описанное упражнение способствует улучшению кровообращения.

Следующее упражнение выполняется стоя. Необходимо вытянуть руки вперед, смотря на конец пальца вытянутой руки, положенной на средней линии лица, медленно приближать палец, не сводя с него глаз до тех пор, пока палец не начнет двоиться. Данное действие повторить 8 раз. Описанное упражнение облегчает работу на близком расстоянии.

Следующее упражнение выполняется сидя. Необходимо закрыть веки, массировать их с помощью круговых движений пальца в течение 1 минуты. Упражнение расслабляет мышцы и улучшает кровообращение.

Следующее упражнение выполняется стоя. Необходимо отвести руку в правую сторону, медленно передвигать палец полусогнутой руки справа налево и при неподвижной голове следить глазами за пальцем, медленно передвигать палец полусогнутой руки слева направо и при неподвижной голове следить глазами за пальцем. Данную последовательность действий повторить 12 раз. Описанное упражнение укрепляет мышцы глаз горизонтального действия и совершенствует их координацию.

Следующее упражнение выполняется сидя. Необходимо тремя пальцами каждой руки легко нажать на верхнее веко, спустя 2 секунд снять пальцы с век. Данную последовательность действий повторить 4 раза. Описанное упражнение улучшает циркуляцию внутриглазной жидкости.

Регулярное выполнение описанной гимнастики способствует снижению воздействия вредных психофизиологических факторов, влияющих на органы зрения специалиста по защите информации.

Заключение

В ходе выполнения выпускной квалификационной работы был проанализирован процесс загрузки программного обеспечения в микроконтроллерах STM32. В результате анализа было составлено описание процесса, определены угрозы процесса, возможности, которыми может обладать нарушитель, в рамках модели нарушителя, определены защитные меры, направленные на нейтрализацию выявленных угроз. Определенный набор защитных мер был реализован в виде программного комплекса для защищенного обновления встроенного программного обеспечения микроконтроллеров STM32. Разработка программного комплекса была выполнена в несколько этапов. На первом этапе были разработаны алгоритмы программного комплекса. Был определен формат защищенного образа ПО. Определен протокол взаимодействия программных компонентов комплекса и составлено описание используемых в нем команд. На третьем этапе была выполнена реализация программного комплекса. На последнем этапе было проведено тестирование программного комплекса. Для тестирования компонентов комплекса был составлен набор тестов на основе методов предположения об ошибке и анализа граничных значений.

Разработанный комплекс может применять для защищенного обновления встроенного программного обеспечения устройств под управлением микроконтроллеров STM32L0x1. Благодаря использованию высокоуровневых библиотек комплекс может быть легко преобразован для работы на микроконтроллерах STM32 других серий.

Таким образом, задание на выпускную квалификационную работу было выполнено в полном объеме.

Список используемых источников

- 1 Методический документ ФСТЭК России «Методика определения угроз безопасности информации в информационных системах» – [Электронный ресурс]. – Режим доступа: <https://fstec.ru/component/%20attachments/download/812> (дата последнего обращения 15.02.2020).
- 2 Криптографические методы защиты информации: классическая криптография :учеб. пособие / С.Н. Борисова, О.В. Липилин. – Пенза : Изд-во ПГУ.
- 3 Средства противодействия угрозам безопасности в микроконтроллерах STM32G0. [Электронный ресурс]. – URL: <https://www.compel.ru/lib/131887> (дата обращения 15.02.2021).
- 4 Proprietary code read-out protection on STM32L4, STM32L4+, STM32G4 and STM32WB Series MCUs [Электронный ресурс]. – URL: https://www.st.com/resource/en/application_note/dm00226247-proprietary-code-readout-protection-on-stm32l4-stm32l4-stm32g4-and-stm32wb-series-mcus-stmicroelectronics.pdf (дата обращения 15.02.2021).
- 5 ГОСТ Р 34.12-2015 Информационная технология. Криптографическая защита информации. Блочные шифры. – М. : Стандартинформ, 2015.
- 6 Режимы работы блочных шифров, реализующие аутентифицированное шифрование. [Электронный ресурс]. – URL: <https://tc26.ru/standarts/rekomendatsii-po-standartizatsii/r-1323565-1-026-2019-informatsionnaya-tekhnologiya-kriptograficheskaya-zashchita-informatsii-rezhimy-raboty-blochnykh-shifrov-realizuyushchie-autentifitsirovannoe-shifrovanie.html> (дата обращения 15.03.2021).
- 7 Intel HEX-record Format. [Электронный ресурс]. – URL: <https://www.lucidtechnologies.info/intel.htm> (дата обращения 05.04.2021).

8 Reference manual RM0377 [Электронный ресурс]. – URL: https://www.st.com/resource/en/reference_manual/dm00108282-ultralowpower-stm32l0x1-advanced-armbased-32bit-mcus-stmicroelectronics.pdf (дата обращения 15.04.2021).

9 Классификация опасных и вредных производственных факторов [Электронный ресурс]. – URL: <https://asout.ru/klassifikatsiya-opasnyih-i-vrednyih-proizvodstvennyih-faktorov> (дата обращения 20.05.2021).

10 Вредные и опасные факторы при работе с компьютером / [Электронный ресурс]. – URL: <http://www.grandars.ru/shkola/bezopasnost-zhiznedeyatelnosti/vrednye-factory-pri-rabote-na-pk.html> (дата обращения: 20.05.2021).

11 Негативные факторы воздействия компьютера на здоровье человека [Электронный ресурс]. – URL: http://e-ikt.uginfo.sfedu.ru/lectures/lecture15/lecture15_2.html (дата обращения 20.05.2021).

12 Вредные и опасные факторы при работе с компьютером [Электронный ресурс]. – URL: <https://www.grandars.ru/shkola/bezopasnost-zhiznedeyatelnosti/vrednye-factory-pri-rabote-na-pk.html> (дата обращения 20.05.2021).

13 Упражнения для глаз [Электронный ресурс]. – URL: <https://moscow.krugozor-clinic.ru/poleznaya-informaciya/zaryadka-dlya-glaz/> (дата обращения 22.05.2021).

Приложение А

(обязательное)

Блок-схемы алгоритмов программного комплекса

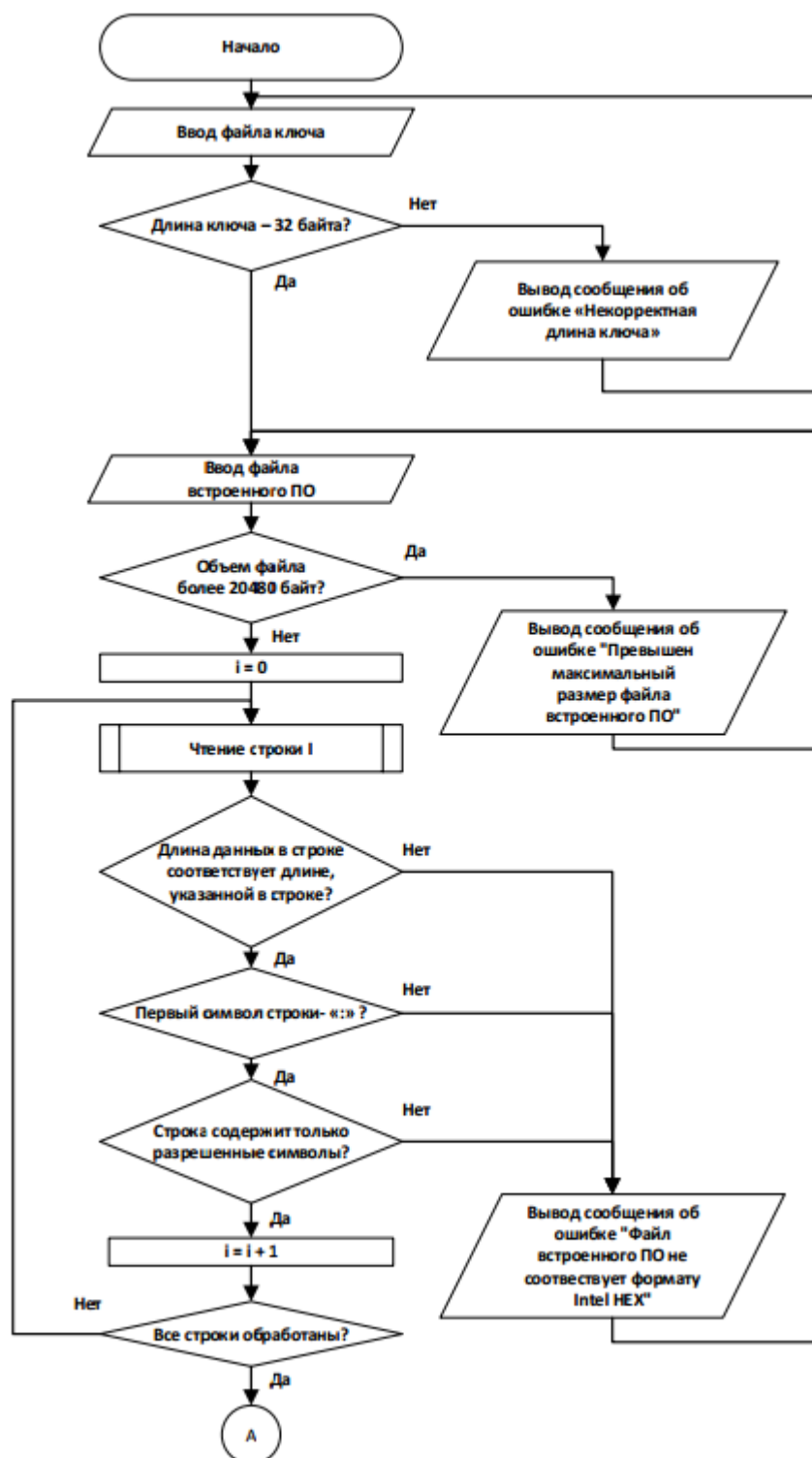


Рисунок А.1 – Алгоритм проверки входных данных генератором защищенного образа (первая часть)

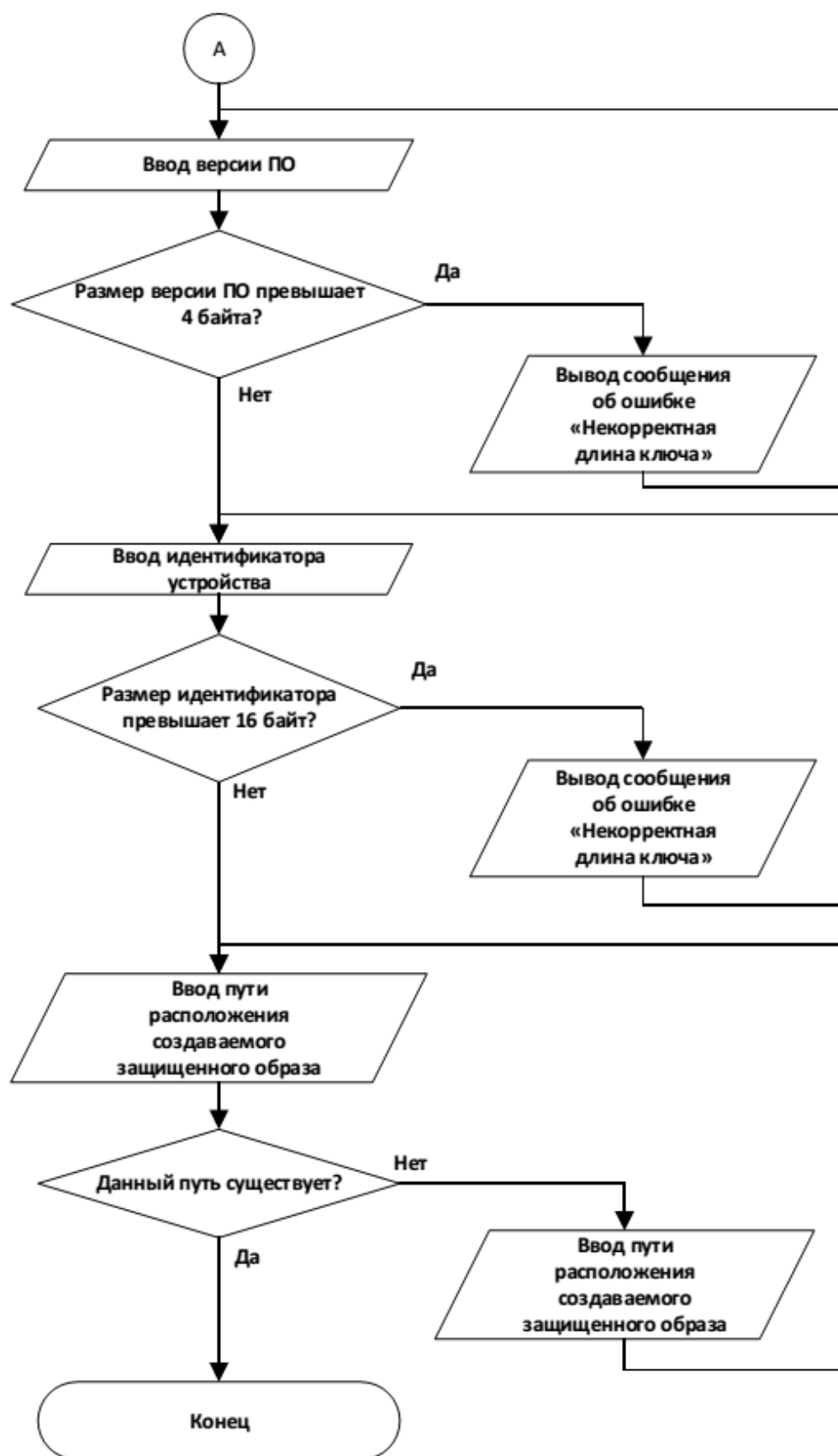


Рисунок А.2 – Алгоритм проверки входных данных генератором защищенного образа (вторая часть)

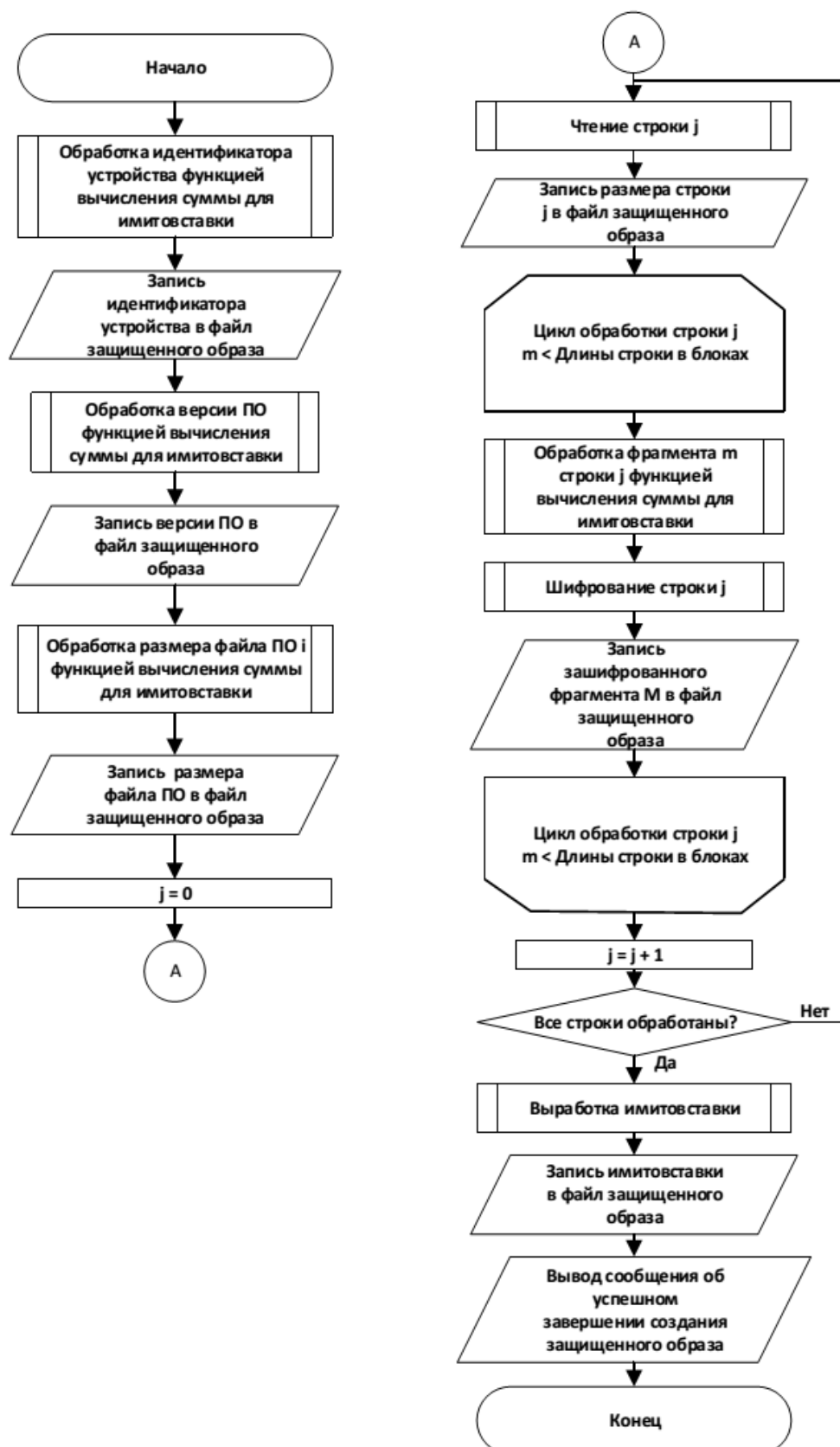


Рисунок А.3 – Алгоритм генерации защищенного образа встроенного ПО

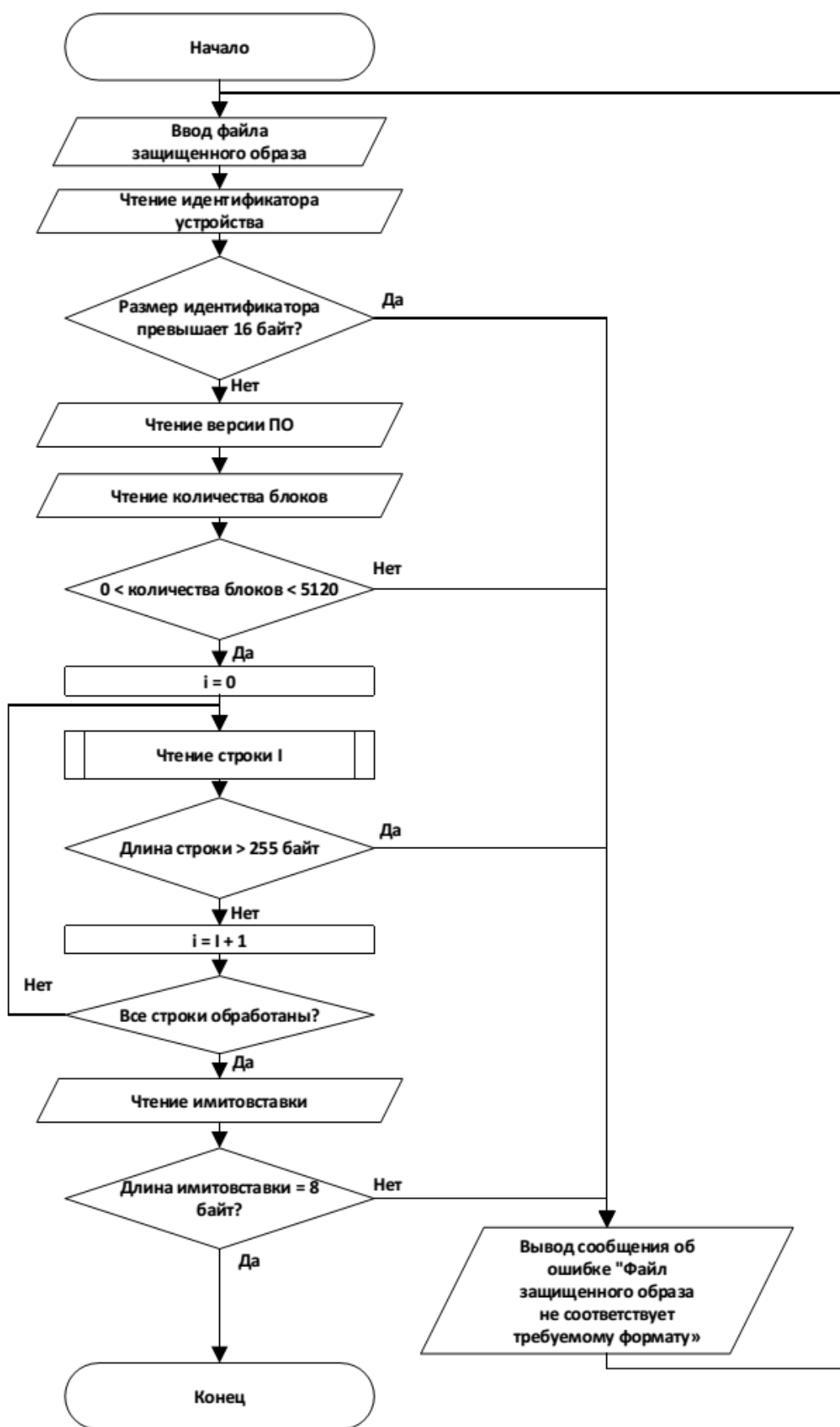


Рисунок А.4 – Алгоритм проверки входных данных программатором

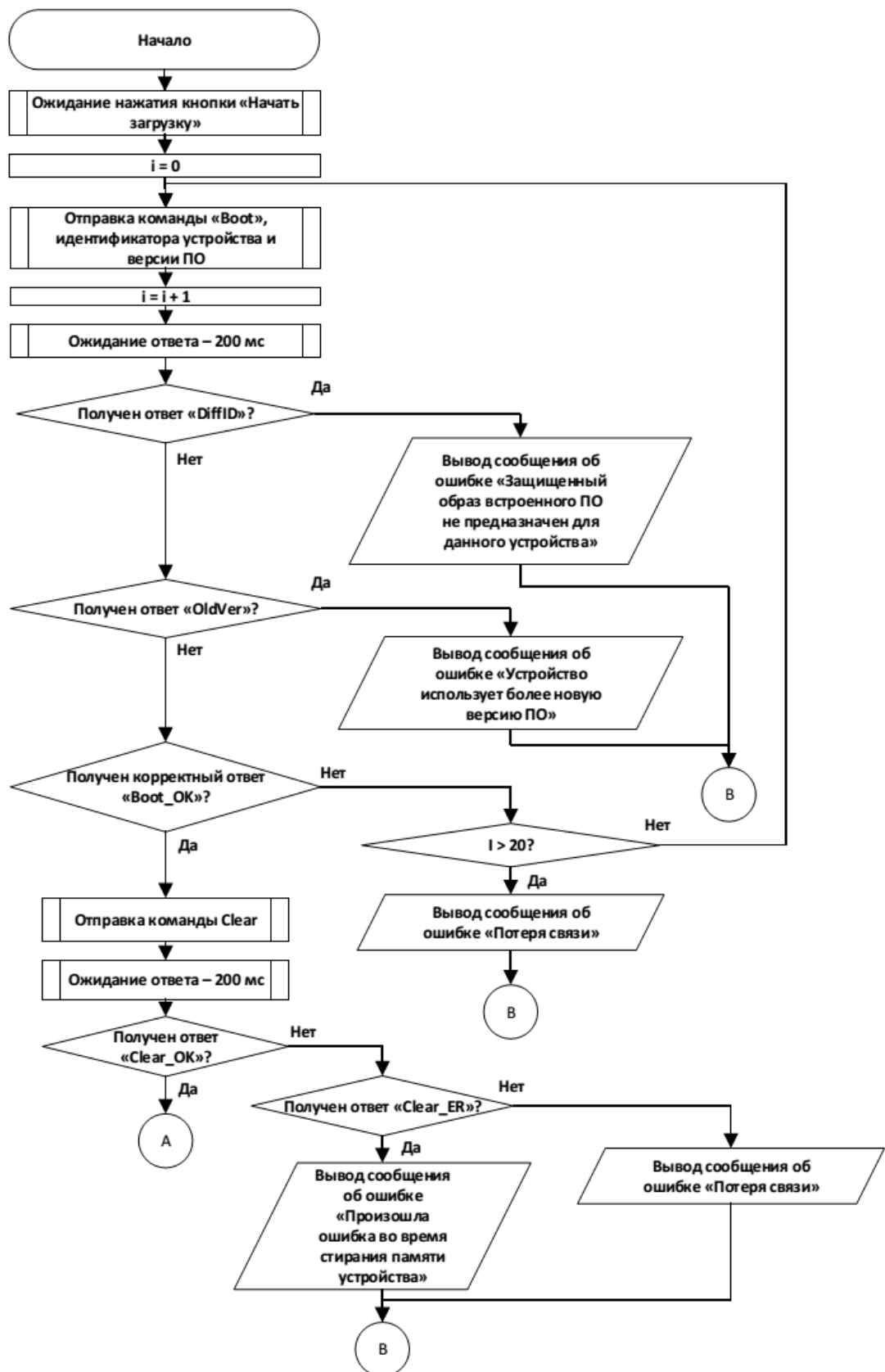


Рисунок А.5 – Алгоритм реализации процесса загрузки со стороны программатора (первая часть)

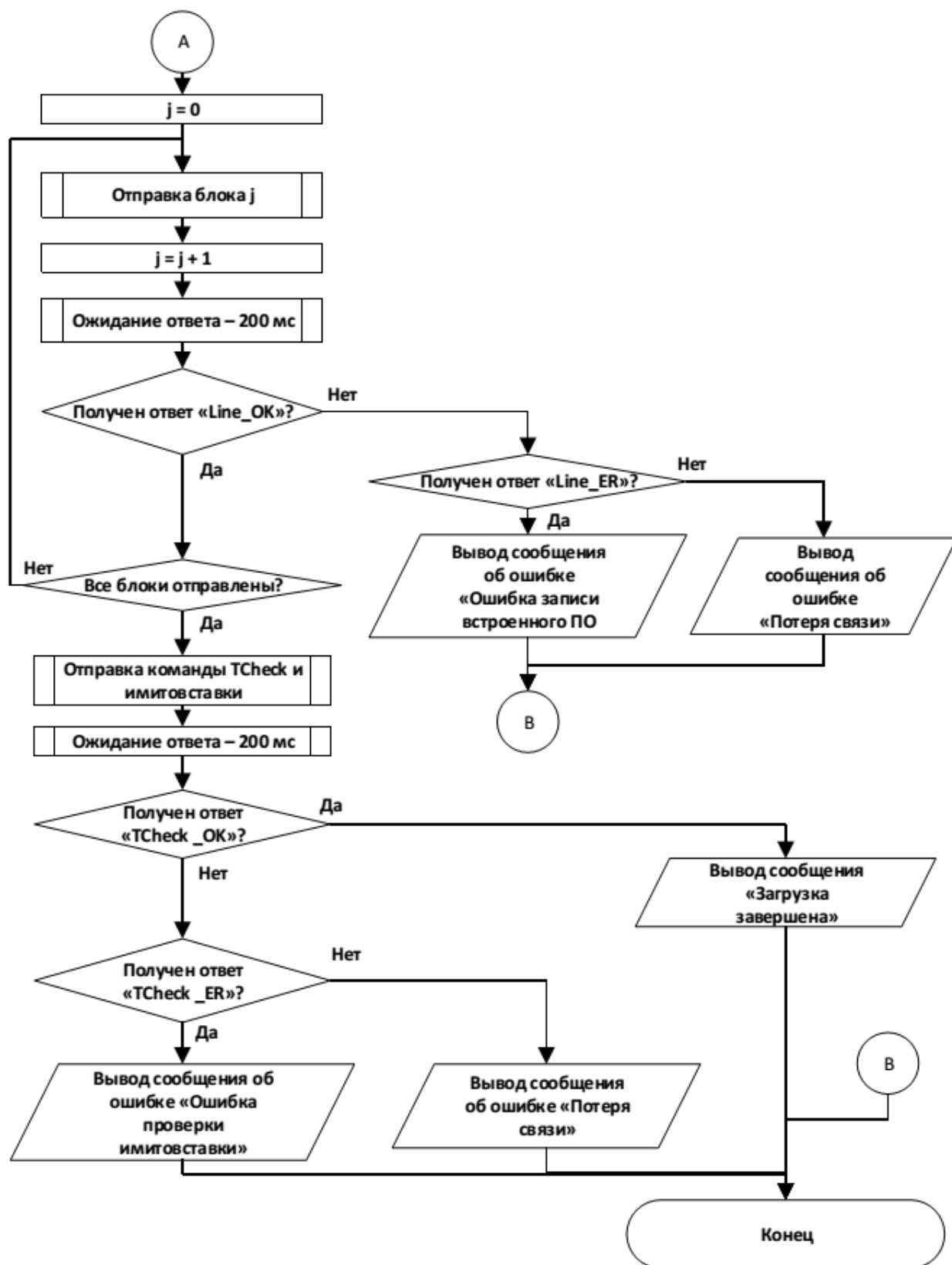


Рисунок А.6 – Алгоритм реализации процесса загрузки со стороны программатора (вторая часть)

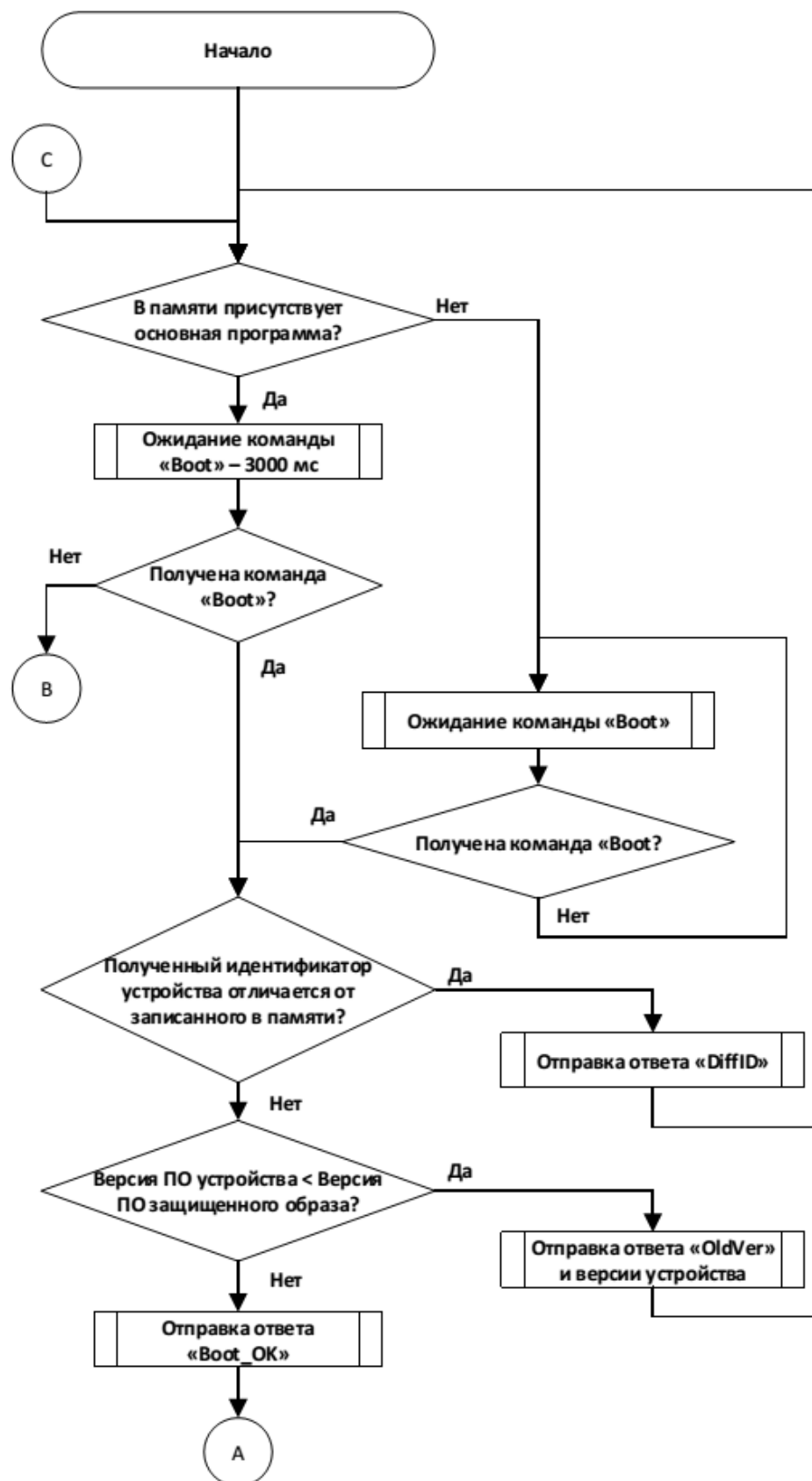


Рисунок А.7 – Алгоритм реализации процесса загрузки со стороны загрузчика (первая часть)

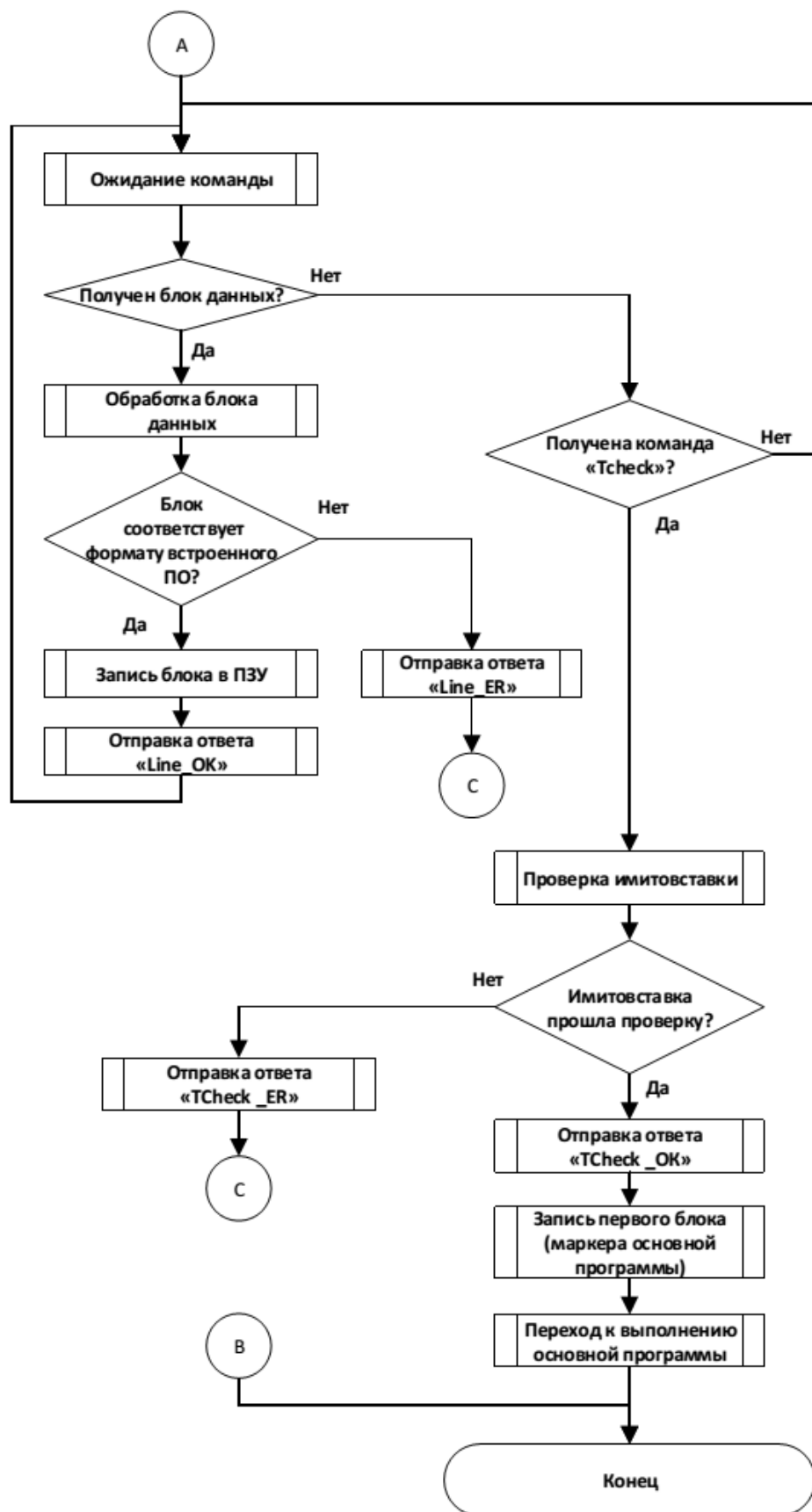


Рисунок А.8 – Алгоритм реализации процесса загрузки со стороны загрузчика (вторая часть)

Приложение Б

(обязательное)

Текст программы генератора защищенного образа

```
// Содержимое файла Magma.cs
namespace Firmware_generator
{
    class Magma
    {
        private static byte[][] Pi =
        {
            new byte[] {1, 7, 14, 13, 0, 5, 8, 3, 4, 15, 10, 6, 9, 12, 11, 2},
            new byte[] {8, 14, 2, 5, 6, 9, 1, 12, 15, 4, 11, 0, 13, 10, 3, 7},
            new byte[] {5, 13, 15, 6, 9, 2, 12, 10, 11, 7, 8, 1, 4, 3, 14, 0},
            new byte[] {7, 15, 5, 10, 8, 1, 6, 13, 0, 9, 3, 14, 11, 4, 2, 12},
            new byte[] {12, 8, 2, 1, 13, 4, 15, 6, 7, 0, 10, 5, 3, 14, 9, 11},
            new byte[] {11, 3, 5, 8, 2, 15, 10, 13, 14, 1, 7, 4, 12, 9, 6, 0},
            new byte[] {6, 8, 2, 3, 9, 10, 5, 12, 1, 14, 4, 7, 11, 13, 0, 15},
            new byte[] {12, 4, 6, 2, 10, 5, 11, 9, 14, 8, 13, 7, 0, 3, 15, 1}
        };

        private uint[] RoundKey = new uint[32];
        private uint Magma_T(uint input)
        {
            return (uint)((uint)(Pi[0][(input & 0xf0000000) >> 28]) << 28 |
                (uint)(Pi[1][(input & 0x0f000000) >> 24]) << 24 |
                (uint)(Pi[2][(input & 0x00f00000) >> 20]) << 20 |
                (uint)(Pi[3][(input & 0x000f0000) >> 16]) << 16 |
                (uint)(Pi[4][(input & 0x0000f000) >> 12]) << 12 |
                (uint)(Pi[5][(input & 0x00000f00) >> 8]) << 8 |
                (uint)(Pi[6][(input & 0x000000f0) >> 4]) << 4 |
                (uint)(Pi[7][input & 0x0000000f]));
        }

        private uint Magma_g(uint RoundKey, uint input)
        {
            return (uint)((Magma_T(input + RoundKey) << 11) | (Magma_T(input +
RoundKey) >> 21));
        }

        private uint[] Magma_G(uint RoundKey, uint[] InputBlock)
        {
            return new uint[2] { InputBlock[1], InputBlock[0] ^
Magma_g(RoundKey, InputBlock[1]) };
        }

        private uint[] Magma_G_Fin(uint RoundKey, uint[] InputBlock)
        {
            return new uint[2] { InputBlock[0] ^ Magma_g(RoundKey,
InputBlock[1]), InputBlock[1] };
        }

        public void Init(uint[] key)
        {
            if (key == null) return;
            Magma_CalcRoundKey(key);
        }

        private void Magma_CalcRoundKey(uint[] key)
        {
            for (int j = 0; j < 3; j++)
                for (int i = 0; i < 8; i++)
```

```

        RoundKey[j * 8 + i] = key[i]; // Формируется 24 итерационных
        ключа с использованием части ключа в прямом порядке

        for (int i = 0; i < 8; i++)
            RoundKey[24 + i] = key[7 - i]; // В обратном порядке
    }

    public uint[] EncryptBlock(uint[] InputBlock)
    {
        uint[] OutputBlock = new uint[2];
        OutputBlock = Magma_G(RoundKey[0], InputBlock); // Первое
        преобразование G
        for (int i = 1; i < 31; i++)
            OutputBlock = Magma_G(RoundKey[i], OutputBlock); // Последующие
        (со второго по тридцать первое) преобразования G
        OutputBlock = Magma_G_Fin(RoundKey[31], OutputBlock); // Последнее
        (тридцать второе) преобразование G
        return OutputBlock;
    }

    public uint[] DecryptBlock(uint[] InputBlock)
    {
        uint[] OutputBlock = new uint[2];
        OutputBlock = Magma_G(RoundKey[31], InputBlock);
        for (int i = 30; i > 0; i--)
            OutputBlock = Magma_G(RoundKey[i], OutputBlock);
        OutputBlock = Magma_G_Fin(RoundKey[0], OutputBlock);
        return OutputBlock;
    }
}

// Содержимое файла MGM.cs
using System;
namespace Firmware_generator
{
    class MGM
    {
        private const uint MGM_MAX_LENGTH = 0x04000000;
        private uint[] k = new uint[8];
        private uint[] nonce = new uint[2];
        private uint[] sum = new uint[2];
        private uint[] Y_i = new uint[2];
        private uint[] Z_i = new uint[2];
        private uint Zcount;
        private uint Pcount;
        private Magma Magma1 = new Magma();
        private uint[] XOR64(uint[] inp1, uint[] inp2)
        {
            return new uint[2] { inp1[0] ^ inp2[0], inp1[1] ^ inp2[1] };
        }

        private uint[] gf64_mul(uint[] inp1, uint[] inp2)
        {
            uint inp1_l = inp1[0];
            uint inp1_r = inp1[1];
            uint inp2_l = inp2[0];
            uint inp2_r = inp2[1];
            uint x_MSB;
            uint[] output = new uint[2];
            for (int i = 0; i < 64; i++)

```

```

    {
        if ((inp2_r & 0x1) != 0)
        {
            output[0] ^= inp1_l;
            output[1] ^= inp1_r;
        }
        //Сдвиг inp2 на 1 бит вправо
        inp2_r = (uint)((inp2_r >> 1) | ((inp2_l & 0x1) << 31));
        inp2_l = (uint)(inp2_l >> 1);
        x_MSB = inp1_l & 0x80000000;
        //Сдвиг inp1 на 1 бит влево
        inp1_l = (uint)((inp1_l << 1) | ((inp1_r & 0x80000000) >> 31));
        inp1_r = (uint)(inp1_r << 1);
        if (x_MSB != 0)
        {
            inp1_r ^= 0x1B;
        }
    }
    return output;
}
private uint[] incr_l(uint[] input)
{
    input[0]++;
    return input;
}

private uint[] incr_r(uint[] input)
{
    input[1]++;
    return input;
}

private uint[] MSB(uint[] InpBlock, uint InpBlockLengh)
{
    if (InpBlockLengh < 8)
    {
        if (InpBlockLengh <= 4)
            return new uint[2] { (uint)((InpBlock[0] >> (int)((4 - InpBlockLengh) * 8)) << (int)((4 - InpBlockLengh) * 8)), 0 };
        else
            return new uint[2] { InpBlock[0], (uint)((InpBlock[1] >> (int)((8 - InpBlockLengh) * 8)) << (int)((8 - InpBlockLengh) * 8)) };
    }
    return InpBlock;
}

public void Init(uint[] NewK, uint[] NewNonce)
{
    Array.Copy(NewK, k, k.Length);
    Array.Copy(NewNonce, nonce, nonce.Length);
    Array.Clear(sum, 0, sum.Length);
    Zcount = 0;
    Pcount = 0;
    Magma1.Init(k);
}

public void ProcBlock(uint[] InpBlock)
{
    uint[] temp = new uint[2];
    uint[] H_i = new uint[2];
    if (Zcount == 0)
    {

```

```

        Array.Copy(nonce, temp, nonce.Length);
        temp[0] |= 0x80000000;
        Z_i = Magma1.EncryptBlock(temp);
        H_i = Magma1.EncryptBlock(Z_i);
    }
    else
    {
        Z_i = incr_l(Z_i);
        H_i = Magma1.EncryptBlock(Z_i);
    }
    temp = gf64_mul(H_i, InpBlock);
    sum = XOR64(sum, temp);
    Zcount++;
}
public uint[] CipherBlock(uint[] InpBlock, uint Block_length = 8)
{
    uint[] temp = new uint[2];
    uint[] H_i = new uint[2];
    if (Pcount == 0)
    {
        Y_i = Magma1.EncryptBlock(nonce);
        temp = Magma1.EncryptBlock(Y_i);
    }
    else
    {
        Y_i = incr_r(Y_i);
        temp = Magma1.EncryptBlock(Y_i);
    }
    Pcount++;
    return MSB(XOR64(temp, InpBlock), Block_length);
}
public uint[] GenerateT(uint A_length, uint P_length)
{
    uint[] temp = new uint[2];
    uint[] temp2 = new uint[2];
    uint[] H_i = new uint[2];
    Z_i = incr_l(Z_i);
    H_i = Magma1.EncryptBlock(Z_i);
    temp[0] = A_length * 8;
    temp[1] = P_length * 8;
    temp2 = gf64_mul(H_i, temp);
    return Magma1.EncryptBlock(XOR64(sum, temp2));
}
private bool CheckT(uint A_length, uint P_length, uint[] T)
{
    uint[] NewT = GenerateT(A_length, P_length);
    return ((NewT[0] == T[0]) && (NewT[1] == T[1]));
}
}
}

// Содержимое файла SFI_Generator.cs
using System.Collections.Generic;
using System;
using System.IO;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Firmware_generator

```

```

{
class SFI_Generator
{
    public string FirmwareFileName, ImageDirName;
    public bool KeyFileSelected, FirmwareFileSelected, ImageDirSelected;
    private uint CounterLine, Cblock_length;
    private uint[] Key = new uint[8];
    private static byte[] StrHexToByteArray(string str)
    {
        Dictionary<string, byte> hexindex = new Dictionary<string, byte>();
        for (int i = 0; i <= 255; i++)
            hexindex.Add(i.ToString("X2"), (byte)i);

        List<byte> hexres = new List<byte>();
        for (int i = 0; i < str.Length; i += 2)
            hexres.Add(hexindex[str.Substring(i, 2)]);

        return hexres.ToArray();
    }
    private static UInt32 ByteToUInt32(byte[] inp)
    {
        return (uint)(inp[0] << 24) |
            (uint)(inp[1] << 16) |
            (uint)(inp[2] << 8) |
            (uint)(inp[3]);
    }
    private static UInt32 ByteToUInt32(byte[] inp, int position)
    {
        return (uint)(inp[position] << 24) |
            (uint)(inp[position + 1] << 16) |
            (uint)(inp[position + 2] << 8) |
            (uint)(inp[position + 3]);
    }
    private static UInt32[] ByteToUInt32Array(byte[] inp)
    {
        int LengthInUInt32 = ((inp.Length - 1) / 4 + 1);
        byte[] temp_b4 = new byte[LengthInUInt32 * 4];
        Array.Copy(inp, temp_b4, inp.Length);
        UInt32[] output = new UInt32[LengthInUInt32];
        for (int i = 0; i < LengthInUInt32; i++)
            output[i] = ByteToUInt32(temp_b4, i * 4);
        return output;
    }
    private static UInt32[] StringToUInt32Array(string inp)
    {
        return ByteToUInt32Array(Encoding.ASCII.GetBytes(inp));
    }
    private static byte[] UInt32ToByte(UInt32 inp)
    {
        return new byte[4] { (byte)((inp & 0xFF000000)>>24),
            (byte)((inp & 0x00FF0000) >> 16),
            (byte)((inp & 0x0000FF00) >> 8),
            (byte)((inp & 0x000000FF))};
    }
    public void ReadKey(string NewKeyFileName)
    {
        FileInfo fileInf = new FileInfo(NewKeyFileName);
        if (fileInf.Length != 32)
            throw new Exception("Некорректная длина ключа");
        byte[] KeyB = new byte[32];
        fileInf.OpenRead().Read(KeyB, 0, 32);
    }
}
}

```

```

        for (int i = 0; i < 8; i++)
            Key[i] = ByteToUInt32(KeyB, i * 4);
        KeyFileSelected = true;
    }
    public void ReadFirmware(string NewFirmwareFileName)
    {
        FileInfo FirmwareFileInfo = new FileInfo(@NewFirmwareFileName);
        if (FirmwareFileInfo.Length > 20480) throw new Exception("Превышен
максимальный размер файла встроенного ПО");
        System.IO.StreamReader InputFs = new
System.IO.StreamReader(@NewFirmwareFileName);
        CounterLine = 0;
        while (true)
        {
            string InputLine = InputFs.ReadLine();
            if (String.IsNullOrEmpty(InputLine) || (InputLine == "\r\n"))
break;

            if (InputLine[0] != ':')
            {
                InputFs.Close();
                throw new Exception("Файл встроенного ПО не соответствует формату
Intel HEX");
            }
            if (((InputLine.Length - 11) / 2) !=
StrHexToByteArray(InputLine.Substring(1, 2))[0])
            {
                InputFs.Close();
                throw new Exception("Файл встроенного ПО не соответствует формату
Intel HEX");
            }
            char[] AllowedSymbols = { '0', '1', '2', '3', '4', '5', '6', '7',
'8', '9', 'A', 'B', 'C', 'D', 'E', 'F' };
            for (int i = 1; i < InputLine.Length; i++)
                if (!AllowedSymbols.Contains(InputLine[i]))
                {
                    InputFs.Close();
                    throw new Exception("Файл встроенного ПО не соответствует
формату Intel HEX");
                }
            CounterLine++;
        }
        if (CounterLine < 2)
        {
            InputFs.Close();
            throw new Exception("Файл встроенного ПО не соответствует формату
Intel HEX");
        }
        FirmwareFileName = NewFirmwareFileName;
        FirmwareFileSelected = true;
        InputFs.Close();
    }
    private void CheckInput(string NewDeviceID)
    {
        if (!KeyFileSelected) throw new Exception("Не выбран файл ключа");
        if (!FirmwareFileSelected) throw new Exception("Не выбран файл
встроенного ПО");
        if (!ImageDirSelected) throw new Exception("Не выбрано расположение
создаваемого защищенного образа");
        if (NewDeviceID.Length == 0) throw new Exception("Не выбран
идентификатор устройства");
        foreach (char ch in NewDeviceID)

```

```

        {
            if (!(((ch > 'A') && (ch < 'Z')) ||
                ((ch > 'a') && (ch < 'z')) ||
                ((ch > '0') && (ch < '9'))
                || ch == '-' || ch == '_')) throw new Exception("Некорректный
идентификатор устройства");
        }
    }

    private void ProcSupportInfo(string NewDeviceIDstr, UInt32
NewFirmwareVersion, ref BinaryWriter writer, ref MGM MGM_obj)
    {
        byte[] NewDeviceID = Encoding.ASCII.GetBytes(NewDeviceIDstr);
        byte[] DeviceID_b4 = new byte[16];
        Array.Copy(NewDeviceID, DeviceID_b4, NewDeviceID.Length);
        UInt32[] DeviceID = new UInt32[4];
        for (int i = 0; i < 4; i++)
            DeviceID[i] = ByteToUInt32(DeviceID_b4, i * 4);

        MGM_obj.ProcBlock(new uint[] { DeviceID[0], DeviceID[1] });
        MGM_obj.ProcBlock(new uint[] { DeviceID[2], DeviceID[3] });
        MGM_obj.ProcBlock(new uint[] { CounterLine, CounterLine });
        writer.Write(NewDeviceIDstr);
        writer.Write(NewFirmwareVersion);
        writer.Write(CounterLine);
    }

    private void ProcLine(string InputLine, ref BinaryWriter writer, ref
MGM MGM_obj)
    {
        uint[] cb = new uint[2];
        string InputLineData = InputLine.Substring(1, InputLine.Length - 1);
        byte[] InputLineData_b = StrHexToByteArray(InputLineData);
        UInt32[] InputLineData_uint = ByteToUInt32Array(InputLineData_b);
        UInt32[] InputLineData_block = new
UInt32[((InputLineData_uint.Length + 1) / 2) * 2];
        Array.Copy(InputLineData_uint, InputLineData_block,
InputLineData_uint.Length);
        writer.Write(InputLineData_b.Length);
        for (int i = 0; i < (InputLineData_block.Length / 2 - 1); i++)
        {
            MGM_obj.ProcBlock(new uint[] { InputLineData_block[i * 2],
InputLineData_block[i * 2 + 1] });
            cb = MGM_obj.CipherBlock(new uint[] { InputLineData_block[i * 2],
InputLineData_block[i * 2 + 1] });
            writer.Write(cb[0]);
            writer.Write(cb[1]);
        }
        MGM_obj.ProcBlock(new uint[]
{ InputLineData_block[InputLineData_block.Length - 2],
InputLineData_block[InputLineData_block.Length - 1] });
        cb = MGM_obj.CipherBlock(new uint[]
{ InputLineData_block[InputLineData_block.Length - 2],
InputLineData_block[InputLineData_block.Length - 1] },
(((UInt32)InputLineData_b.Length - 1) % 8) + 1);
        writer.Write(cb[0]);
        if (((((UInt32)InputLineData_b.Length - 1) % 8) + 1) > 4)
writer.Write(cb[1]);

        Cblock_length += (UInt32)InputLineData_b.Length * 8;
    }
}

```

```

        public void EncryptFirmware(string NewDeviceIDstr, UInt32
NewFirmwareVersion, ref ToolStripStatusLabel StatusLabel) //
Encoding.ASCII.GetBytes(textBox4.Text); Convert.ToUInt32(numericUpDown1.Value)
    {
        CheckInput(NewDeviceIDstr);
        Cblock_length = 0;
        System.IO.StreamReader InputFs = new
System.IO.StreamReader(@FirmwareFileName);

        uint[] Nonce = new uint[2];
        Nonce[0] = NewFirmwareVersion;

        MGM MGM1 = new MGM();
        MGM1.Init(Key, Nonce);
        BinaryWriter writer = new
BinaryWriter(File.Open(System.IO.Path.Combine(@ImageDirName, @NewDeviceIDstr +
".sfi"), FileMode.Create));
        ProcSupportInfo(NewDeviceIDstr, NewFirmwareVersion, ref writer, ref
MGM1);
        for (int i = 0; i < CounterLine; i++)
        {
            string InputLine = InputFs.ReadLine();
            if (String.IsNullOrEmpty(InputLine) || (InputLine == "\r\n"))
break;

            ProcLine(InputLine, ref writer, ref MGM1);
            StatusLabel.Text = Convert.ToString((int)((double)i /
(double)CounterLine) * 100) + '%';
        }
        uint[] T = MGM1.GenerateT(192, Cblock_length);
        writer.Write(T[0]);
        writer.Write(T[1]);
        StatusLabel.Text = "Создание защищенного образа завершено";
        writer.Close();
        InputFs.Close();
    }
}
}

```

// Содержимое файла Form1.cs

```

using System;
using System.IO;
using System.Windows.Forms;

namespace Firmware_generator
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        SFI_Generator SFI_Gen = new SFI_Generator();

        private void button3_Click(object sender, EventArgs e)
        {
            SFI_Gen.FirmwareFileSelected = false;
            openFileDialog1.FileName = textBox3.Text;
            openFileDialog1.Filter = "Файлы встроенного ПО|*.HEX";
            openFileDialog1.Title = "Выбор файла встроенного ПО";
            if (openFileDialog1.ShowDialog() == DialogResult.OK)

```

```

        {
            textBox3.Text = openFileDialog1.FileName;
            try
            {
                SFI_Gen.ReadFirmware(openFileDialog1.FileName);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Ошибка!");
                return;
            }
            if (!SFI_Gen.ImageDirSelected)
            {
                textBox5.Text = Path.GetDirectoryName(openFileDialog1.FileName);
                SFI_Gen.ImageDirName = textBox5.Text;
                SFI_Gen.ImageDirSelected = true;
            }
        }
    }

    private void button2_Click(object sender, EventArgs e)
    {
        SFI_Gen.KeyFileSelected = false;
        openFileDialog1.FileName = textBox2.Text;
        openFileDialog1.Filter = "Ключ|*.bin";
        openFileDialog1.Title = "Выбор файла ключа";
        if (openFileDialog1.ShowDialog() == DialogResult.OK)
        {
            textBox2.Text = openFileDialog1.FileName;
            try
            {
                SFI_Gen.ReadKey(@openFileDialog1.FileName);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Ошибка!");
            }
        }
    }

    private void button5_Click(object sender, EventArgs e)
    {
        SFI_Gen.ImageDirSelected = false;
        if (folderBrowserDialog1.ShowDialog() == DialogResult.OK)
        {
            textBox5.Text = folderBrowserDialog1.SelectedPath;
            SFI_Gen.ImageDirName = folderBrowserDialog1.SelectedPath;
            SFI_Gen.ImageDirSelected = true;
        }
    }

    private void button4_Click(object sender, EventArgs e)
    {
        try
        {
            SFI_Gen.EncryptFirmware(textBox4.Text,
            Convert.ToUInt32(numericUpDown1.Value), ref toolStripStatusLabel1);
        }
        catch (Exception ex)
        {

```

```
        MessageBox.Show(ex.Message, "Ошибка!");  
    }  
    return;  
}  
  
}  
}
```

Приложение В

(обязательное)

Текст программы программатора

```
// Содержимое файла BootLoader.cs
using System.Collections.Generic;
using System;
using System.IO;
using Microsoft.VisualBasic;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace BootLoader_STM32L051
{
    class SFI_Reader
    {
        public string DeviceIDstr;
        public UInt32 FirmwareVersion, LineNumber, LineCounter;
        private bool FileCheked;
        public SFI_Reader()
        {
            LineCounter = 0;
        }
        private static byte[] UInt32ToByte(UInt32 inp)
        {
            return new byte[4] { (byte)((inp & 0xFF000000)>>24),
                                (byte)((inp & 0x00FF0000) >> 16),
                                (byte)((inp & 0x0000FF00) >> 8),
                                (byte)((inp & 0x000000FF))};
        }
        private static byte[] UInt32ToByteArray(UInt32[] inp)
        {
            byte[] output = new byte[inp.Length * 4];
            for (int i = 0; i < inp.Length; i++)
            {
                byte[] output_tmp = UInt32ToByte(inp[i]);
                output[i * 4] = output_tmp[0];
                output[i * 4 + 1] = output_tmp[1];
                output[i * 4 + 2] = output_tmp[2];
                output[i * 4 + 3] = output_tmp[3];
            }
            return output;
        }

        public void CheckFile(ref BinaryReader reader)
        {
            FileCheked = false;
            GetSupportInfo(ref reader);
            for (int i = 0; i < LineNumber; i++)
            {
                byte[] inpstr = GetSFILine(ref reader);
            }
            FileCheked = true;
            reader.Close();
        }

        public byte[] GetSFILine(ref BinaryReader reader)
        {

```

```

        if (LineCounter >= LineNumber) return null;
        UInt32 InputLineLengh = reader.ReadUInt32();
        if (InputLineLengh > 255)
        {
            throw new Exception("Файл защищенного образа не соответствует
требуемому формату");
        }
        UInt32[] InputLineData_block = new UInt32[((InputLineLengh + 7) / 8)
* 2];
        for (int i = 0; i < (InputLineData_block.Length / 2 - 1); i++)
        {
            InputLineData_block[i * 2] = reader.ReadUInt32();
            InputLineData_block[i * 2 + 1] = reader.ReadUInt32();
        }
        InputLineData_block[InputLineData_block.Length - 2] =
reader.ReadUInt32();
        if (((UInt32)InputLineLengh - 1) % 8) + 1 > 4)
        InputLineData_block[InputLineData_block.Length - 1] = reader.ReadUInt32();
        byte[] InputLineData_b8 = UInt32ToByteArray(InputLineData_block);
        byte[] InputLineData_b = new byte[InputLineLengh];
        Array.Copy(InputLineData_b8, InputLineData_b,
InputLineData_b.Length);
        LineCounter++;
        return InputLineData_b;
    }
    public void GetSupportInfo(ref BinaryReader reader)
    {
        LineCounter = 0;
        DeviceIDstr = reader.ReadString();
        FirmwareVersion = reader.ReadUInt32();
        LineNumber = reader.ReadUInt32();
        if ((DeviceIDstr.Length == 0) || (DeviceIDstr.Length > 16) ||
(LineNumber == 0))
            throw new Exception("Файл защищенного образа не соответствует
требуемому формату");
    }
    public byte[] GetSFI_T(ref BinaryReader reader)
    {
        UInt32[] T_uint = new UInt32[2];
        T_uint[0] = reader.ReadUInt32();
        T_uint[1] = reader.ReadUInt32();
        return UInt32ToByteArray(T_uint);
    }
}
class SFI_Loader
{
    public enum States
    {
        Stop,
        StartLoad,
        ReciveID,
        StartClear,
        ClearAck,
        SendLine,
        SendLineAck,
        SendAuthTag,
        SendAuthTagAck,
        StartApp,
        StartAppAck
    }
    public States State;

```

```

private DateTime Time_Start;
private SFI_Reader SFIreader = new();
private BinaryReader reader;
private Receiver receiver = new();
private int CountReplay;
private static byte[] StrHexToByteArray(string str)
{
    Dictionary<string, byte> hexindex = new Dictionary<string, byte>();
    for (int i = 0; i <= 255; i++)
        hexindex.Add(i.ToString("X2"), (byte)i);
    List<byte> hexres = new List<byte>();
    for (int i = 0; i < str.Length; i += 2)
        hexres.Add(hexindex[str.Substring(i, 2)]);
    return hexres.ToArray();
}
private static byte[] Uint32ToByte(UInt32 inp)
{
    return new byte[4] { (byte)((inp & 0xFF000000)>>24),
        (byte)((inp & 0x00FF0000) >> 16),
        (byte)((inp & 0x0000FF00) >> 8),
        (byte)((inp & 0x000000FF))};
}
private static UInt32 ByteToUInt32(byte[] inp)
{
    return (uint)(inp[0] << 24) |
        (uint)(inp[1] << 16) |
        (uint)(inp[2] << 8) |
        (uint)(inp[3]);
}
public void LoadProc(ref ProgressBar progressBar1, ref
System.IO.Ports.SerialPort serialPort1, ref ToolStripStatusLabel StatusLabel,
ref Timer DelayTimer, ref Timer ProcTimer)
{
    ProcTimer.Stop();
    switch (State)
    {
        case States.StartLoad:
        {
            StatusLabel.Text = "Начало загрузки";
            string DeviceID_hex =
BitConverter.ToString(Encoding.ASCII.GetBytes(SFIreader.DeviceIDstr)).Replace("-",
string.Empty);
            byte[] DeviceID_32b = new byte[32];
            Array.Copy(Encoding.ASCII.GetBytes(DeviceID_hex),
DeviceID_32b, DeviceID_hex.Length);
            string sendID = Encoding.Default.GetString(DeviceID_32b);
            serialPort1.Write(":Boot" + sendID +
BitConverter.ToString(Uint32ToByte(SFIreader.FirmwareVersion)).Replace("-",
string.Empty) + "\n");
            DelayTimer.Interval = 200;
            DelayTimer.Enabled = true;
            State = States.ReciveID;
            break;
        }
        case States.ReciveID:
        {
            if (DelayTimer.Enabled == false)
            {
                CountReplay++;
                if (CountReplay > 20)
                {

```

```

        LoadException("Потеря связи", ref serialPort1, ref
progressBar1);
    }
    State = States.StartLoad;
}
if (receiver.Receive(serialPort1))
{
    if (receiver.Data == ":Boot_OK\n")
    {
        State = States.StartClear;
    }
    else if (receiver.Data.Substring(0, 7) == ":OldVer")
    {
        uint Version =
ByteToUInt32(StrHexToByteArray(receiver.Data.Substring(7, 8)));
        LoadException("Устройство использует более новую версию
ПО: " + Version.ToString(), ref serialPort1, ref progressBar1);
    }
    else if (receiver.Data == ":DiffID\n")
    {
        LoadException("Защищенный образ встроенного ПО не
предназначен для данного устройства", ref serialPort1, ref progressBar1);
    }
}
break;
}
case States.StartClear:
{
    StatusLabel.Text = "Стирание памяти";
    serialPort1.Write(":Clear\n");
    DelayTimer.Interval = 2000;
    DelayTimer.Enabled = true;
    State = States.ClearAck;
    break;
}
case States.ClearAck:
{
    if (receiver.Receive(serialPort1))
    {
        if (receiver.Data == ":Clear_OK\n")
        {
            StatusLabel.Text = "Запись и расшифровка ПО";
            State = States.SendLine;
        }
        if (receiver.Data == ":Clear_ER\n")
        {
            LoadException("Произошла ошибка во время стирания памяти
устройства", ref serialPort1, ref progressBar1);
        }
    }
    if (DelayTimer.Enabled == false)
    {
        LoadException("Потеря связи", ref serialPort1, ref
progressBar1);
    }
    break;
}
case States.SendLine:
{
    if (SFIREader.LineCounter < SFIREader.LineNumber)
    {

```

```

        byte[] inpstr = SFIREader.GetSFILine(ref reader);
        serialPort1.Write(": " +
BitConverter.ToString(inpstr).Replace("-", string.Empty) + "\n");
        DelayTimer.Interval = 200;
        DelayTimer.Start();
        if (progressBar1.Value < progressBar1.Maximum)
progressBar1.Value++;
        State = States.SendLineAck;
    }
    else
    {
        DelayTimer.Enabled = false;
        State = States.SendAuthTag;
    }
    break;
}
case States.SendLineAck:
{
    if (DelayTimer.Enabled == false)
        LoadException("Потеря связи", ref serialPort1, ref
progressBar1);
    if (receiver.Receive(serialPort1))
    {
        if (receiver.Data == ":Line_OK\n")
        {
            State = States.SendLine;
            DelayTimer.Stop();
        }
        else if (receiver.Data == ":Line_ER\n")
        {
            LoadException("Ошибка записи встроенного ПО", ref
serialPort1, ref progressBar1);
        }
        else
        {
            LoadException("Потеря связи", ref serialPort1, ref
progressBar1);
        }
        receiver.Data = "";
    }
    break;
}
case States.SendAuthTag:
{
    StatusLabel.Text = "Проверка имитовставки";
    byte[] T = SFIREader.GetSFI_T(ref reader);
    serialPort1.Write(":CheckT" +
BitConverter.ToString(T).Replace("-", string.Empty) + "\n");
    DelayTimer.Interval = 200;
    DelayTimer.Enabled = true;
    State = States.SendAuthTagAck;
    break;
}
case States.SendAuthTagAck:
{
    if (DelayTimer.Enabled == false)
        LoadException("Потеря связи", ref serialPort1, ref
progressBar1);
    if (receiver.Receive(serialPort1))
    {
        if (receiver.Data == ":CheckT_OK\n")

```

```

        {
            DelayTimer.Stop();
            TimeSpan Time_Delta = DateTime.Now - Time_Start;
            MessageBox.Show("Загрузка завершена. Время загрузки: " +
Time_Delta.Minutes.ToString().PadLeft(2, '0') + ":" +
            Time_Delta.Seconds.ToString().PadLeft(2, '0') + ":" +
Time_Delta.Milliseconds.ToString().PadLeft(3, '0'));
            State = States.Stop;
            serialPort1.Close();
            reader.Close();
        }
        else if (receiver.Data == ":CheckT_ER\n")
        {
            LoadException("Ошибка проверки имитовставки", ref
serialPort1, ref progressBar1);
        }
    }
    break;
}

}
ProcTimer.Start();
}
private void LoadException(string msg, ref System.IO.Ports.SerialPort
serialPort1, ref ProgressBar progressBar1)
{
    State = States.Stop;
    serialPort1.Close();
    reader.Close();
    Program.formobj.button1.Enabled = true;
    Program.formobj.button2.Enabled = true;
    progressBar1.Value = 0;
    progressBar1.Visible = false;
    throw new Exception(msg);
}
public void StartLoad(string FileName, ref ProgressBar progressBar1,
ref System.IO.Ports.SerialPort serialPort1, ref List<COMPortInfo> ComList, ref
System.Windows.Forms.ComboBox comboBoxPortName, ref ToolStripStatusLabel
StatusLabel, ref Timer DelayTimer, ref Timer ProcTimer)
{
    try
    {
        SFIReader = new();
        reader = new BinaryReader(File.Open(FileName, FileMode.Open));
        SFIReader.GetSupportInfo(ref reader);
    }
    catch (Exception ex)
    {
        Program.formobj.button1.Enabled = true;
        Program.formobj.button2.Enabled = true;
        MessageBox.Show(ex.Message, "Ошибка чтения файла",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }
    try
    {
        serialPort1.PortName =
ComList[comboBoxPortName.SelectedIndex].Name;
        serialPort1.Open();
        serialPort1.DiscardInBuffer();
        serialPort1.DiscardOutBuffer();
    }
}

```

```

    }
    catch (Exception ex)
    {
        Program.formobj.button1.Enabled = true;
        Program.formobj.button2.Enabled = true;
        MessageBox.Show(ex.Message, "Не удалось открыть порт",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }
    progressBar1.Step = 1;
    progressBar1.Maximum = (int)SFIreader.LineNumber;
    progressBar1.Value = 0;
    progressBar1.Visible = true;
    try
    {
        State = States.StartLoad;
        receiver = new Receiver();
        Time_Start = DateTime.Now;
        CountReplay = 0;
        LoadProc(ref progressBar1, ref serialPort1, ref StatusLabel, ref
        DelayTimer, ref ProcTimer);
    }
    catch (Exception ex)
    {
        Program.formobj.button1.Enabled = true;
        Program.formobj.button2.Enabled = true;
        State = States.Stop;
        MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
        MessageBoxIcon.Error);
        return;
    }
    ProcTimer.Enabled = true;
    return;
}
}
class Receiver
{
    private enum States
    {
        NewLine,
        ContinueLine
    }
    private States State = States.NewLine;
    public string Data;
    public Receiver()
    {
        State = States.NewLine;
        Data = "";
    }
    public bool Receive(System.IO.Ports.SerialPort serialPort1)
    {
        byte[] buf = new byte[1024];
        int Count = serialPort1.BytesToRead;
        if (Count != 0)
        {
            serialPort1.Read(buf, 0, Count);
            for (int i = 0; i < Count; i++)
            {
                switch (State)
                {
                    case States.NewLine:

```

```

        {
            if (buf[i] == ':')
            {
                Data = "";
                Data += Strings.Chr(buf[i]);
                State = States.ContinueLine;
            }
            break;
        }
        case States.ContinueLine:
        {
            if (buf[i] == '\n')
            {
                Data += Strings.Chr(buf[i]);
                State = States.NewLine;
                return true;
            }
            else Data += Strings.Chr(buf[i]);
            break;
        }
    }
    }
    return false;
}
else return false;
}
}
}
// Содержимое файла ComPortName.cs
using System;
using System.Collections.Generic;
using System.Management;

namespace BootLoader_STM32L051
{
    internal class ProcessConnection
    {
        public static ConnectionOptions ProcessConnectionOptions()
        {
            ConnectionOptions options = new ConnectionOptions();
            options.Impersonation = ImpersonationLevel.Impersonate;
            options.Authentication = AuthenticationLevel.Default;
            options.EnablePrivileges = true;
            return options;
        }
        public static ManagementScope ConnectionScope(string machineName,
        ConnectionOptions options, string path)
        {
            ManagementScope connectScope = new ManagementScope();
            connectScope.Path = new ManagementPath(@"\\" + machineName +
path);

            connectScope.Options = options;
            connectScope.Connect();
            return connectScope;
        }
    }
    public class COMPortInfo
    {
        public string Name { get; set; }
        public string Description { get; set; }
        public COMPortInfo() { }
    }
}

```

```

        public static List<COMPortInfo> GetCOMPortsInfo()
        {
            List<COMPortInfo> comPortInfoList = new List<COMPortInfo>();
            ConnectionOptions options =
ProcessConnection.ProcessConnectionOptions();
            ManagementScope connectionScope =
ProcessConnection.ConnectionScope(Environment.MachineName, options,
@"\root\CIMV2");
            ObjectQuery objectQuery = new ObjectQuery("SELECT * FROM
Win32_PnPEntity WHERE ConfigManagerErrorCode = 0");
            ManagementObjectSearcher comPortSearcher = new
ManagementObjectSearcher(connectionScope, objectQuery);
            using (comPortSearcher)
            {
                string caption = null;
                foreach (ManagementObject obj in comPortSearcher.Get())
                {
                    if (obj != null)
                    {
                        object captionObj = obj["Caption"];
                        if (captionObj != null)
                        {
                            caption = captionObj.ToString();
                            if (caption.Contains("(COM)")
                                {
                                    COMPortInfo comPortInfo = new
COMPortInfo();
                                    comPortInfo.Name =
caption.Substring(caption.LastIndexOf("(COM)").Replace("(",
string.Empty).Replace(")", string.Empty));
                                    comPortInfo.Description = caption;
                                    comPortInfoList.Add(comPortInfo);
                                }
                            }
                        }
                    }
                }
            }
            return comPortInfoList;
        }
    }

// Содержимое файла Form1.cs
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.IO;
using System.IO.Ports;

namespace BootLoader_STM32L051
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private string SFI_FileName;
        private void button1_Click(object sender, EventArgs e)
        {
            openFileDialog1.FileName = textBox1.Text;
        }
    }
}

```

```

        openFileDialog1.Filter = "Файлы защищенного образа встроенного
ПО|*.sfi";
        openFileDialog1.Title = "Выбор файла защищенного образа
встроенного ПО";
        if (openFileDialog1.ShowDialog() == DialogResult.OK)
        {
            SFI_FileName = textBox1.Text = openFileDialog1.FileName;
            try
            {
                BinaryReader reader = new
BinaryReader(File.Open(openFileDialog1.FileName, FileMode.Open));
                SFI_Reader SFIreader = new SFI_Reader();
                SFIreader.CheckFile(ref reader);
                reader.Close();
                label3.Text = "Идентификатор устройства: " +
SFIreader.DeviceIDstr;
                label4.Text = "Версия ПО: " +
SFIreader.FirmwareVersion.ToString();
                if (comboBoxPortName.Items.Count != 0) button2.Enabled
= true;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Ошибка чтения файла",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
        }
    }

    private void folderBrowserDialog1_HelpRequest(object sender,
EventArgs e)
    {
    }

    public List<COMPortInfo> ComList = null;
    private void Form1_Load(object sender, EventArgs e)
    {
        progressBar1.Visible = false;
        button2.Enabled = false;
        ComList = COMPortInfo.GetCOMPortsInfo();
        foreach (COMPortInfo comPort in ComList)
        {
            comboBoxPortName.Items.Add(comPort.Description);
        }
        if (comboBoxPortName.Items.Count != 0)
comboBoxPortName.SelectedIndex = 0;
    }

    private void timer1_Tick(object sender, EventArgs e)
    {
        timer1.Enabled = false;
    }

    SFI_Loader Loader;
    SerialPort serialPort1;
    private void timer2_Tick(object sender, EventArgs e)
    {
        if (Loader.State != SFI_Loader.States.Stop)
        {
            try

```

```

        {
            Loader.LoadProc(ref progressBar1, ref serialPort1, ref
toolStripStatusLabel1, ref timer1, ref timer2);
        }
        catch (Exception ex)
        {
            Loader.State = SFI_Loader.States.Stop;
            MessageBox.Show(ex.Message, "Ошибка загрузки",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }
    }
    else
    {
        button1.Enabled = true;
        button2.Enabled = true;
        timer2.Enabled = false;
        progressBar1.Visible = false;
        toolStripStatusLabel1.Text = "";
    }
}

private void button2_Click(object sender, EventArgs e)
{
    Loader = new SFI_Loader();
    serialPort1 = new SerialPort();
    serialPort1.BaudRate = 115000;
    serialPort1.DataBits = 8;
    serialPort1.DiscardNull = false;
    serialPort1.DtrEnable = false;
    serialPort1.Handshake = Handshake.None;
    serialPort1.Parity = Parity.None;
    serialPort1.ParityReplace = 63;
    serialPort1.ReadBufferSize = 1024;
    serialPort1.ReadTimeout = -1;
    serialPort1.ReceivedBytesThreshold = 1;
    serialPort1.RtsEnable = false;
    serialPort1.StopBits = StopBits.One;
    serialPort1.WriteBufferSize = 1024;
    serialPort1.WriteTimeout = -1;
    button1.Enabled = false;
    button2.Enabled = false;
    try
    {
        Loader.StartLoad(SFI_FileName, ref progressBar1, ref
serialPort1, ref ComList, ref comboBoxPortName, ref toolStripStatusLabel1, ref
timer1, ref timer2);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Ошибка загрузки",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        button1.Enabled = true;
        button2.Enabled = true;
        return;
    }
}
}
}

```

Приложение Г

(обязательное)

Текст программы загрузчика

```
// Содержимое файла cipher_magma.h
#include <stdio.h>
#include <string.h>

#define MAGMA_BLOCK_SIZE_INT32 2
typedef uint32_t magma_block_t[MAGMA_BLOCK_SIZE_INT32];

#define MAGMA_KEY_SIZE_INT32 8
typedef uint32_t magma_key_t[MAGMA_KEY_SIZE_INT32];

void Magma_Init(const magma_key_t *key);
void MagmaEncryptBlock(const uint32_t *InputBlock, uint32_t *OutputBlock);
void MagmaDecryptBlock(const uint32_t *InputBlock, uint32_t *OutputBlock);

// Содержимое файла cipher_magma.c
#include "main.h"
uint32_t RoundKey[32]; // Итерационные ключи шифрования
void Magma_CalcRoundKey(const uint32_t *key);
uint32_t Magma_T(const uint32_t input);
uint32_t Magma_g(const uint32_t RoundKey, const uint32_t input);
void Magma_G(const uint32_t RoundKey, const uint32_t *InputBlock, uint32_t
*output);
void Magma_G_Fin(const uint32_t RoundKey, uint32_t *InputBlock, uint32_t
*output);

static unsigned char Pi[8][16] =
{
{ 1,7,14,13,0,5,8,3,4,15,10,6,9,12,11,2 },
{ 8,14,2,5,6,9,1,12,15,4,11,0,13,10,3,7 },
{ 5,13,15,6,9,2,12,10,11,7,8,1,4,3,14,0 },
{ 7,15,5,10,8,1,6,13,0,9,3,14,11,4,2,12 },
{ 12,8,2,1,13,4,15,6,7,0,10,5,3,14,9,11 },
{ 11,3,5,8,2,15,10,13,14,1,7,4,12,9,6,0 },
{ 6,8,2,3,9,10,5,12,1,14,4,7,11,13,0,15 },
{ 12,4,6,2,10,5,11,9,14,8,13,7,0,3,15,1 }
};

void Magma_Init(const magma_key_t *key)
{
if (key == NULL) return;
Magma_CalcRoundKey((uint32_t *)key);
}

inline void Magma_CalcRoundKey(const uint32_t *key)
{
for (int j = 0; j < 3; j++)
for (int i = 0; i < 8; i++)
RoundKey[j * 8 + i] = key[i]; // Формируется 24 итерационных
ключа с использованием части ключа в прямом порядке
for (int i = 0; i < 8; i++)
RoundKey[24 + i] = key[7 - i]; // В обратном порядке
}

inline uint32_t Magma_T(const uint32_t input)
```

```

{
return ((uint32_t)(Pi[0][(input & 0xf0000000) >> 28]) << 28 |
        (uint32_t)(Pi[1][(input & 0x0f000000) >> 24]) << 24 |
        (uint32_t)(Pi[2][(input & 0x00f00000) >> 20]) << 20 |
        (uint32_t)(Pi[3][(input & 0x000f0000) >> 16]) << 16 |
        (uint32_t)(Pi[4][(input & 0x0000f000) >> 12]) << 12 |
        (uint32_t)(Pi[5][(input & 0x00000f00) >> 8]) << 8 |
        (uint32_t)(Pi[6][(input & 0x000000f0) >> 4]) << 4 |
        (uint32_t)(Pi[7][input & 0x0000000f]));
}

inline uint32_t Magma_g(const uint32_t RoundKey, const uint32_t input)
{
uint32_t temp32 = Magma_T(input + RoundKey); // Сложение по модулю 32
правой части блока с итерационным ключом
return (temp32 << 11) | (temp32 >> 21); // Циклически сдвиг на 11 разрядов
влево
}

inline void Magma_G(const uint32_t RoundKey, const uint32_t *InputBlock,
uint32_t *output)
{
uint32_t temp = InputBlock[0]; // для сохранения значения InputBlock[0]
output[0] = InputBlock[1];
output[1] = temp ^ Magma_g(RoundKey, InputBlock[1]);
}

inline void Magma_G_Fin(const uint32_t RoundKey, uint32_t *InputBlock,
uint32_t *output)
{
output[0] = InputBlock[0] ^ Magma_g(RoundKey, InputBlock[1]);
output[1] = InputBlock[1];
}

void MagmaEncryptBlock(const uint32_t *InputBlock, uint32_t *OutputBlock)
{
Magma_G(RoundKey[0], InputBlock, OutputBlock); // Первое преобразование G
for (int i = 1; i < 31; i++)
    Magma_G(RoundKey[i], OutputBlock, OutputBlock); // Последующие (со
второго по тридцать первое) преобразования G
Magma_G_Fin(RoundKey[31], OutputBlock, OutputBlock); // Последнее
(тридцать второе) преобразование G
}

void MagmaDecryptBlock(const uint32_t *InputBlock, uint32_t *OutputBlock)
{
Magma_G(RoundKey[31], InputBlock, OutputBlock);
for (int i = 30; i > 0; i--)
    Magma_G(RoundKey[i], OutputBlock, OutputBlock);
Magma_G_Fin(RoundKey[0], OutputBlock, OutputBlock);
}

// Содержимое файла mgm.h
#include "main.h"
#include "cipher_magma.h"

#define MGM_MAX_LENGTH 0x04000000

#define FALSE 0
#define TRUE 1

```

```

void MGM_Init(const uint32_t *K, // ключ шифрования
const uint32_t *nonce); // уникальный вектор

void MGM_ProcBlock(const uint32_t *InpBlock);
uint8_t MGM_CheckT(uint32_t A_length, uint32_t P_length, uint32_t *T);
void MGM_CipherBlock(const uint32_t *InpBlock, uint32_t *OutpBlock,
uint32_t Block_length);

// Содержимое файла mgm.c
#include "mgm.h"

uint32_t *MGM_XOR64(const uint32_t *inp1, const uint32_t *inp2, uint32_t
*output);
uint32_t *MGM_incr_l(uint32_t *input);
uint32_t *MGM_incr_r(uint32_t *input);

uint32_t *MGM_MSB(uint32_t *InpBlock, uint32_t InpBlockLength);

uint32_t *MGM_gf64_mul(const uint32_t *inp1, const uint32_t *inp2,
uint32_t *output);

typedef struct {
magma_key_t k;
magma_block_t nonce;
magma_block_t sum;
magma_block_t Y_i;
magma_block_t Z_i;
uint32_t Zcount;
uint32_t Pcount;
} T_MGM;
T_MGM MGM_context;

inline uint32_t *MGM_XOR64(const uint32_t *inp1, const uint32_t *inp2,
uint32_t *output)
{
output[0] = inp1[0] ^ inp2[0];
output[1] = inp1[1] ^ inp2[1];
return output;
}

inline uint32_t *MGM_gf64_mul(const uint32_t *inp1, const uint32_t *inp2,
uint32_t *output)
{
uint32_t inp1_l = inp1[0];
uint32_t inp1_r = inp1[1];
uint32_t inp2_l = inp2[0];
uint32_t inp2_r = inp2[1];
uint32_t x_MSB;
output[0] = 0;
output[1] = 0;
for (int i = 0; i < 64; i++) {
if (inp2_r & 0x1)
{
output[0] ^= inp1_l;
output[1] ^= inp1_r;
}
//Сдвиг inp2 на 1 бит вправо
inp2_r = ((inp2_r >> 1) | ((inp2_l & 0x1) << 31));
inp2_l = inp2_l >> 1;
x_MSB = inp1_l & 0x80000000;
//Сдвиг inp1 на 1 бит влево

```

```

        inpl_l = ((inpl_l << 1) | ((inpl_r & 0x80000000) >> 31));
        inpl_r = inpl_r << 1;
        if (x_MSB) inpl_r ^= 0x1B;
    }
    return output;
}

inline uint32_t *MGM_incr_l(uint32_t *input)
{
    input[0]++;
    return input;
}

inline uint32_t *MGM_incr_r(uint32_t *input)
{
    input[1]++;
    return input;
}

void MGM_Init(const uint32_t *K, // ключ шифрования
const uint32_t *nonce) // уникальный вектор
{
    memcpy(MGM_context.k, K, 32);
    memcpy(MGM_context.nonce, nonce, 8);
    memset(MGM_context.sum, 0, 8);
    MGM_context.Zcount = 0;
    MGM_context.Pcount = 0;
    Magma_Init((magma_key_t *)MGM_context.k);
}

inline uint32_t *MGM_MSB(uint32_t *InpBlock, uint32_t InpBlockLengh)
{
    if (InpBlockLengh < 8)
    {
        if (InpBlockLengh <= 4)
        {
            uint32_t tmp = (4 - InpBlockLengh) * 8;
            InpBlock[1] = 0;
            InpBlock[0] = (InpBlock[0] >> tmp) << tmp;
        }
        else
        {
            uint32_t tmp = (8 - InpBlockLengh) * 8;
            InpBlock[1] = (InpBlock[1] >> tmp) << tmp;
        }
    }
    return InpBlock;
}

void MGM_ProcBlock(const uint32_t *InpBlock)
{
    uint32_t temp[2], H_i[2];
    if (MGM_context.Zcount >= MGM_MAX_LENGTH) return; //ошибка
    if (MGM_context.Zcount++ == 0)
    {
        memcpy(temp, MGM_context.nonce, 8); // temp = 0 || попсе
        temp[0] |= 0x80000000; // 1 || nonce
        MagmaEncryptBlock(temp, MGM_context.Z_i); //Z_1 = E_K(1 || ICN)
        MagmaEncryptBlock(MGM_context.Z_i, H_i); //H_i = E_K(Z_i)
    }
}

```

```

else
{
    MGM_incr_l(MGM_context.Z_i); // Z_{i+1} = incr_l(Z_i)
    MagmaEncryptBlock(MGM_context.Z_i, H_i); //H_i = E_K(Z_i)
}
MGM_gf64_mul(H_i, InpBlock, temp);
MGM_XOR64(MGM_context.sum, temp, MGM_context.sum);
MGM_context.Zcount++;
}

void MGM_CipherBlock(const uint32_t *InpBlock, uint32_t *OutpBlock,
uint32_t Block_length)
{
    uint32_t temp[2];
    if (MGM_context.Pcount >= MGM_MAX_LENGTH) return; //ошибка
    if (MGM_context.Pcount == 0)
    {
        MagmaEncryptBlock(MGM_context.nonce, MGM_context.Y_i); //y1 = Ek(0
|| nonce)
        MagmaEncryptBlock(MGM_context.Y_i, temp); // E_K(Y_1)
    }
    else
    {
        MGM_incr_r(MGM_context.Y_i); // Y_i = incr_r(Y_{i-1})
        MagmaEncryptBlock(MGM_context.Y_i, temp); //E_K(Y_i)
    }
    MGM_context.Pcount++;
    MGM_MSB(MGM_XOR64(temp, InpBlock, OutpBlock), Block_length);
}

void MGM_GenerateT(uint32_t A_length, uint32_t P_length, uint32_t *T)
{
    uint32_t temp[2], temp2[2], H_i[2];
    MGM_incr_l(MGM_context.Z_i); // Z_{i+1} = incr_l(Z_i)
    MagmaEncryptBlock(MGM_context.Z_i, H_i); //H_{h+q+1}
    temp[0] = A_length * 8;
    temp[1] = P_length * 8;
    MGM_gf64_mul(H_i, temp, temp2); // H_{h+q+1} (x) || (len(A) || len(C))
    MagmaEncryptBlock(MGM_XOR64(MGM_context.sum, temp2, MGM_context.sum), T);
}

uint8_t MGM_CheckT(uint32_t A_length, uint32_t P_length, uint32_t *T)
{
    uint32_t NewT[2];
    MGM_GenerateT(A_length, P_length, NewT);
    if ((NewT[0] == T[0]) && (NewT[1] == T[1])) return TRUE;
    else return FALSE;
}

// Содержимое файла support.h
uint8_t Hex8ToUint(uint8_t i);
uint8_t Hex16ToUint(uint8_t *i);
uint16_t Hex32ToUint(uint8_t *i);
void HexToUintArray(const char * Hex_str, uint8_t *Out, size_t
SizeInChar);
void ByteToHex(char *HexStr, const uint8_t *Bytes, int HexLen);
int CompareArrays(char* arr1, char* arr2, int len);
void Uint32ToChar(uint32_t inp, char* out);
uint32_t CharToUint32(char* inp);

// Содержимое файла support.c

```

```

#include "main.h"

inline uint8_t Hex8ToUint(uint8_t i)
{
    if (i >= 0x61 && i <= 0x66) return i - 0x61 + 10;
    else if (i >= 0x41 && i <= 0x46) return i - 0x41 + 10;
    else if (i >= 0x30 && i <= 0x39) return i - 0x30;
    else return 0;
}

uint8_t Hex16ToUint(uint8_t *i)
{
    return (Hex8ToUint(i[0]) << 4) + Hex8ToUint(i[1]);
}

uint16_t Hex32ToUint(uint8_t *i)
{
    return (Hex8ToUint(i[0]) << 12) + (Hex8ToUint(i[1]) << 8) +
        (Hex8ToUint(i[2]) << 4) + Hex8ToUint(i[3]);
}

void HexToUintArray(const char * Hex_str, uint8_t *Out, size_t SizeInChar)
{
    for (int i=0; i<SizeInChar; i++)
        Out[i] = Hex16ToUint((uint8_t*)Hex_str + i*2);
}

void ByteToHex(char *HexStr, const uint8_t *Bytes, int HexLen)
{
    const char xx[] = "0123456789ABCDEF";
    while (--HexLen >= 0) HexStr[HexLen] = xx[(Bytes[HexLen]>>1] >> ((1 -
(HexLen&1)) << 2)) & 0xF];
}

int CompareArrays(char* arr1, char* arr2, int len)
{
    for (int i = 0; i < len; i++)
        if (arr1[i] != arr2[i]) return 0;
    return 1;
}

void Uint32ToChar(uint32_t inp, char* out)
{
    out[0] = (char)((inp & 0xFF000000)>>24);
    out[1] = (char)((inp & 0x00FF0000)>>16);
    out[2] = (char)((inp & 0x0000FF00)>>8);
    out[3] = (char)((inp & 0x000000FF));
}

uint32_t CharToUint32(char* inp)
{
    uint32_t out = (uint32_t)inp[0] << 24 |
        (uint32_t)inp[1] << 16 |
        (uint32_t)inp[2] << 8 |
        (uint32_t)inp[3];
    return out;
}

// Содержимое файла LoadProc.h
#define IS_APP_IN_MEM (((*(__IO uint32_t*) APPLICATION_ADDRESS) &
0x2FFE0000) == 0x20000000)

```

```

#define BOOTLOADER_TIMEOUT 3000
#define ERROR 0
#define SEND_ID 1
#define CLEAR 2
#define TCHECK 3
#define APP_START 4
#define READ_STRING 5

void LoadProc();

extern UART_HandleTypeDef hlpuart1;
typedef struct
{
    uint16_t RxCounter;
    uint16_t TxCounter;
    volatile uint16_t TimeoutCounter;
    volatile uint16_t StaterRX;
    unsigned char FlagPacketEnable;
    char RxBuffer[260];
    uint16_t TxCounterTransmit;
    volatile unsigned char FlagPacketTransmitEnable;
    char TxBuffer[100];
} T_USART;
extern T_USART USART;

typedef struct
{
    uint8_t FirstBlock[64];
    uint16_t WaitCounter;
    uint32_t P_length;
    uint8_t AuthTag[8];
    char ReciveID[16];
    uint32_t ReciveVer;
    char AllowLoad;
    uint8_t Block_ui8[128];
    uint32_t DecryptBlock_ui32[32];
    uint32_t* Block_ui32;
    uint8_t* DecryptBlock_ui8;
    uint8_t FirstBlockFlag;
} T_Loader;
extern T_Loader Loader;

// Содержимое файла LoadProc.c

#include "main.h"

T_Loader Loader;
T_USART USART;
UART_HandleTypeDef UartHandle;

typedef enum ParserOut
{
    Error,
    CheckID,
    Clear,
    TCheck,
    AppStart,
    ReadString
} ParserOut;

```

```

#define IHEX_CHR_RECORDMARK 0x3A

#define IHEX_OK 0x00
#define IHEX_ERR_INCORRECT_CHECKSUM 0x01

#define IHEX_ERR_PARSE_ERROR 0x03
#define IHEX_ERR_WRONG_RECORD_LENGTH 0x04

typedef unsigned int uint_t;
typedef uint8_t ihex_rdata_t;
typedef uint8_t ihex_rlen_t;
typedef uint8_t ihex_rchks_t;
typedef unsigned int ihex_error_t;
typedef uint16_t ihex_addr_t;
typedef enum
{
    IHEX_DATA = 0x00, IHEX_EOF = 0x01, IHEX_ESA = 0x02,
    IHEX_SSA = 0x03, IHEX_ELA = 0x04, IHEX_SLA = 0x05
} ihex_rtype_t;
typedef struct ihex_record
{
    unsigned int ihr_length;
    ihex_rtype_t ihr_type;
    ihex_addr_t ihr_address;
    ihex_rdata_t ihr_data[200];
    ihex_rchks_t ihr_checksum;
} ihex_record_t;

typedef void (*pFunction)(void);
pFunction JumpToApplication;
uint32_t JumpAddress;

const char DeviceID[] = "TestProg\0";
const uint32_t Key[] = {0x31323334, 0x35363738, 0x39303132, 0x33343536,
0x37383930, 0x31323334, 0x35363738, 0x39303132};

void SendMsg(char* msg);
ParserOut Parser(void);
void StartMainApp();
void TransmitLPUSART();
void ProcBlock(void);
void ParserHex(void);
int ihex_parse_single_record(uint8_t* data, unsigned int length,
ihex_record_t* record);
int ihex_check_record(ihex_record_t *r);

void LoadProc()
{
    if (Loader.WaitCounter >= BOOTLOADER_TIMEOUT)
    {
        if (IS_APP_IN_MEM) StartMainApp();
        Loader.WaitCounter = 0;
    }
    switch (Parser())
    {
        case Error:
        {
            break;
        }
        case CheckID:

```

```

{
    char DeviceIDbuf[16] = {0};
    memcpy(DeviceIDbuf, DeviceID, strlen(DeviceID));
    if (CompareArrays(DeviceIDbuf, Loader.ReciveID, 16) == 0)
    {
        strcpy(USART.TxBuffer, ":DiffID\n");
        USART.TxCOUNTER = 8;
        Loader.AllowLoad = 0;
    }
    else
    {
        //uint32_t Version = VERSION_EPR
        if ((VERSION_EPR > Loader.ReciveVer) && ((VERSION_KEY_EPR ==
VERSION_KEY)))
        {
            strcpy(USART.TxBuffer, ":OldVer");
            char Ver_c[4];
            Uint32ToChar(VERSION_EPR, Ver_c);
            char VerHEXbuf[8];
            ByteToHex(VerHEXbuf, (uint8_t*)Ver_c, 8);
            memcpy(USART.TxBuffer + 7, VerHEXbuf, 8);
            USART.TxBuffer[15] = '\n';
            USART.TxCOUNTER = 16;
            Loader.AllowLoad = 0;
        }
        else
        {
            Loader.WaitCounter = 0;
            strcpy(USART.TxBuffer, ":Boot_OK\n");
            USART.TxCOUNTER = 9;
            Loader.AllowLoad = 1;
            uint32_t Nonce[2] = {0};
            Nonce[0] = Loader.ReciveVer;
            MGM_Init(Key, Nonce);
            uint32_t* DeviceID_uint32 = (uint32_t*)Loader.ReciveID;
            for (int i = 0; i < 4; i++) DeviceID_uint32[i] =
__REV(DeviceID_uint32[i]);
            MGM_ProcBlock(DeviceID_uint32);
            MGM_ProcBlock(DeviceID_uint32 + 1);
            Loader.P_length = 0;
        }
    }
    TransmitLPUSART();
    while (USART.FlagPacketTransmitEnable == 0) continue;
    break;
}
case Clear:
{
    Loader.FirstBlockFlag = 1;
    FLASH_If_Init();
    if (FLASH_If_GetWriteProtectionStatus() != 0)
    {
        strcpy(USART.TxBuffer, ":Clear_ER\n");
        USART.TxCOUNTER = strlen(":Clear_ER\n");
        TransmitLPUSART();
        while (USART.FlagPacketTransmitEnable == 0)
        {
            continue;
        }
        break;
    }
}

```

```

        __disable_irq();
        uint32_t Res = FLASH_If_Erase(APPLICATION_ADDRESS);
        __enable_irq();
        if (Res == 0) SendMsg(":Clear_OK\n");
        else SendMsg(":Clear_ER\n");
        break;
    }
    case TCheck:
    {
        uint32_t* T_ui32 = (uint32_t*)(Loader.AuthTag);
        T_ui32[0] = __REV(T_ui32[0]);
        T_ui32[1] = __REV(T_ui32[1]);
        if (MGM_CheckT(192, Loader.P_lengh, T_ui32))
        {
            if (FLASH_If_Write(APPLICATION_ADDRESS, (uint32_t *)
Loader.FirstBlock, 16) == FLASHIF_OK)
            {
                uint32_t temp[16];
                temp[0] = Loader.ReceiveVer;
                temp[1] = VERSION_KEY;
                FLASH_EraseInitTypeDef desc;
                HAL_FLASH_Unlock();
                uint32_t result = FLASHIF_OK;
                desc.PageAddress = VERSION_ADDRESS;
                desc.TypeErase = FLASH_TYPEERASE_PAGES;
                desc.NbPages = 1;
                if (HAL_FLASHEx_Erase(&desc, &result) == HAL_OK){
                    HAL_FLASH_Lock();
                    if (FLASH_If_Write(VERSION_ADDRESS, (uint32_t *) temp, 2) ==
FLASHIF_OK)
                    {
                        SendMsg(":CheckT_OK\n");
                        StartMainApp();
                    }
                }
            }
            else SendMsg(":CheckT_ER\n");
        }
        else SendMsg(":CheckT_ER\n");
        break;
    }
    case ReadString:
    {
        ProcBlock();
        ParserHex();
        break;
    }
}

ParserOut Parser(void)
{
    if (USART.FlagPacketEnable == 1)
    {
        if (USART.RxBuffer[0] != ':') return Error;
        switch (USART.RxBuffer[2])
        {
            case 'o':
            {
                if (strncmp(USART.RxBuffer, ":Boot", 5) == 0)
                {

```

```

        if (USART.RxCounter != 46) return Error;
        memset(Loader.ReciveID, 0, 16);
        HexToUintArray((char*) (USART.RxBuffer + 5),
(uint8_t*)Loader.ReciveID, 16);
        char ReciveVer_c[4];
        HexToUintArray((char*) (USART.RxBuffer + 37),
(uint8_t*)ReciveVer_c, 4);
        Loader.ReciveVer = CharToUint32(ReciveVer_c);
        USART.TimeoutCounter = 0;
        USART.RxCounter = 0;
        USART.StateRX = 0;
        USART.FlagPacketEnable = 0;
        return CheckID;
    }
    break;
}
case 'l':
{
    if (strncmp(USART.RxBuffer, ":Clear\n", USART.RxCounter) == 0)
    {
        USART.TimeoutCounter = 0;
        USART.RxCounter = 0;
        USART.StateRX = 0;
        USART.FlagPacketEnable = 0;
        return Clear;
    }
    break;
}
case 'h':
{
    if (strncmp(USART.RxBuffer, ":CheckT", 7) == 0)
    {
        if (USART.RxCounter != 24) return Error;
        HexToUintArray(USART.RxBuffer + 7, Loader.AuthTag, 8);
        USART.TimeoutCounter = 0;
        USART.RxCounter = 0;
        USART.StateRX = 0;
        USART.FlagPacketEnable = 0;
        return TCheck;
    }
    break;
}
default:
{
    USART.TimeoutCounter = 0;
    USART.RxCounter = 0;
    USART.StateRX = 0;
    USART.FlagPacketEnable = 0;
    return ReadString;
    break;
}
}
}
return Error;
}

void ProcBlock(void)
{
    // Определение длины блока
    int BlockLenght_hex;
    for (BlockLenght_hex = 0; USART.RxBuffer[++BlockLenght_hex] != '\n';) ;

```

```

    BlockLenght_hex--;
    int BlockLenght_byte = BlockLenght_hex / 2;
    int BlockLenght_uint32 = ((BlockLenght_byte - 2) / 4 + 1);

    //Преобразование в uint32
    uint8_t Block_ui8[128] = {0};
    uint32_t DecryptBlock_ui32[32] = {0};
    HexToUintArray((char*)(USART.RxBuffer + 1), (uint8_t*)Block_ui8,
    BlockLenght_hex);
    uint32_t* Block_ui32 = (uint32_t*)(Block_ui8);
    for (int i = 0; i < BlockLenght_uint32; i++) Block_ui32[i] =
    __REV(Block_ui32[i]);

    //Расшифрование
    int BlockLenght_mgmb1 = ((BlockLenght_uint32 - 2) / 2);
    for (int i = 0; i < BlockLenght_mgmb1; i++)
    {
        MGM_CipherBlock(Block_ui32 + i*2, DecryptBlock_ui32 + i*2, 8);
        MGM_ProcBlock(DecryptBlock_ui32 + i*2);
    }
    MGM_CipherBlock(Block_ui32 + BlockLenght_mgmb1*2, DecryptBlock_ui32 +
    BlockLenght_mgmb1*2, ((BlockLenght_byte - 2) % 8) + 1);
    MGM_ProcBlock(DecryptBlock_ui32 + BlockLenght_mgmb1*2);
    Loader.P_length += BlockLenght_byte*8;

    //Преобразование в hex
    for (int i = 0; i < BlockLenght_uint32; i++) DecryptBlock_ui32[i] =
    __REV(DecryptBlock_ui32[i]);
    uint8_t* DecryptBlock_ui8 = (uint8_t*)DecryptBlock_ui32;
    ByteToHex(USART.RxBuffer + 1, DecryptBlock_ui8, BlockLenght_hex);
}

void ParserHex(void)
{
    static uint32_t Extended_Address = 0;
    static uint32_t Address_Write = 0;
    static uint8_t Data_Write[64];
    static uint16_t CounterDataWrite = 0;
    static uint32_t State_Write = 0;
    static uint32_t State_Extended_Address = 0;
    static ihex_record_t recordHex;
    switch (ihex_parse_single_record((uint8_t*) USART.RxBuffer,
    USART.RxCounter, &recordHex))
    {
        case IHEX_OK:
        {
            switch (recordHex.ihr_type)
            {
                case IHEX_DATA:
                {
                    switch (State_Write)
                    {
                        case 0:
                        {
                            Address_Write = Extended_Address + recordHex.ihr_address;
                            if (Address_Write < APPLICATION_ADDRESS)
                            {
                                SendMsg(":Line_ER\n");
                                return;
                            }
                        }
                        for (uint16_t i = 0; i < recordHex.ihr_length; i++)

```

```

{
    Data_Write[CounterDataWrite++] = recordHex.ihr_data[i];
    if (CounterDataWrite == 64)
    {
        if (Address_Write == APPLICATION_ADDRESS)
        {
            Loader.FirstBlockFlag = 0;
            memcpy(Loader.FirstBlock, Data_Write, 64);
        }
        else
        {
            FLASH_If_Write(Address_Write, (uint32_t *) Data_Write,
16);

            for (uint16_t i = 0; i < 64; i++)
            {
                Data_Write[i] = 0;
            }
            CounterDataWrite = 0;
            Address_Write += 64;
        }
    }
    State_Write = 1;
    break;
}
case 1:
{
    if ((Address_Write - Extended_Address + CounterDataWrite) ==
recordHex.ihr_address)
    {
        for (uint16_t i = 0; i < recordHex.ihr_length; i++)
        {
            Data_Write[CounterDataWrite++] = recordHex.ihr_data[i];
            if (CounterDataWrite == 64)
            {
                if (Address_Write == APPLICATION_ADDRESS)
                {
                    Loader.FirstBlockFlag = 0;
                    memcpy(Loader.FirstBlock, Data_Write, 64);
                }
                else
                {
                    FLASH_If_Write(Address_Write, (uint32_t *) Data_Write,
16);

                    for (uint16_t i = 0; i < 64; i++)
                    {
                        Data_Write[i] = 0;
                    }
                    CounterDataWrite = 0;
                    Address_Write += 64;
                }
            }
        }
    }
    else
    {
        if (Address_Write == APPLICATION_ADDRESS)
        {
            Loader.FirstBlockFlag = 0;
            memcpy(Loader.FirstBlock, Data_Write, 64);
        }
        else
        {

```

```

FLASH_If_Write(Address_Write, (uint32_t *) Data_Write,
16);
    }
    for (uint16_t i = 0; i < 64; i++) Data_Write[i] = 0;
    CounterDataWrite = 0;
    Address_Write = Extended_Address + recordHex.ihr_address;
    if (Address_Write < APPLICATION_ADDRESS)
    {
        SendMsg(":Line_ER\n");
        return;
    }
    for (uint16_t i = 0; i < recordHex.ihr_length; i++)
    {
        Data_Write[CounterDataWrite++] = recordHex.ihr_data[i];
        if (CounterDataWrite == 64)
        {
            if (Address_Write == APPLICATION_ADDRESS)
            {
                Loader.FirstBlockFlag = 0;
                memcpy(Loader.FirstBlock, Data_Write, 64);
            }
            else
            {
                FLASH_If_Write(Address_Write, (uint32_t *) Data_Write,
16);
            }
            for (uint16_t i = 0; i < 64; i++) Data_Write[i] = 0;
            CounterDataWrite = 0;
            Address_Write += 64;
        }
    }
    }
    break;
}
}
SendMsg(":Line_OK\n");
break;
}
case IHEX_EOF:
{
    if (CounterDataWrite != 0)
    {
        if (Address_Write == APPLICATION_ADDRESS)
        {
            Loader.FirstBlockFlag = 0;
            memcpy(Loader.FirstBlock, Data_Write, 64);
        }
        else
        {
            FLASH_If_Write(Address_Write, (uint32_t *) Data_Write,
16);
        }
        for (uint16_t i = 0; i < 64; i++) Data_Write[i] = 0;
        CounterDataWrite = 0;
        Address_Write += 64;
    }
    SendMsg(":Line_OK\n");
    break;
}
case IHEX_ESA:
case IHEX_SSA:

```

```

        SendMsg(":Line_OK\n");
        break;

    case IHEX_ELA:
    {
        switch (State_Extended_Address)
        {
            case 0:
            {
                Extended_Address = (recordHex.ihr_data[0] << 8) +
recordHex.ihr_data[1];
                Extended_Address = Extended_Address << 16;
                State_Extended_Address = 1;
                break;
            }
            case 1:
            {
                if (State_Write != 0)
                {
                    if (Address_Write == APPLICATION_ADDRESS)
                    {
                        Loader.FirstBlockFlag = 0;
                        memcpy(Loader.FirstBlock, Data_Write, 64);
                    }
                    else
                    {
                        FLASH_If_Write(Address_Write, (uint32_t *) Data_Write,
16);
                    }
                    for (uint16_t i = 0; i < 64; i++) Data_Write[i] = 0;
                    CounterDataWrite = 0;
                    State_Write = 0;
                }
                Extended_Address = (recordHex.ihr_data[0] << 8) +
recordHex.ihr_data[1];
                Extended_Address = Extended_Address << 16;
            }
        }
        SendMsg(":Line_OK\n");
        break;
    }
    case IHEX_SLA:
    {
        SendMsg(":Line_OK\n");
        break;
    }
    break;
}

case IHEX_ERR_INCORRECT_CHECKSUM:
case IHEX_ERR_PARSE_ERROR:
case IHEX_ERR_WRONG_RECORD_LENGTH:
    SendMsg(":Line_ER\n");
    break;
}
}

void StartMainApp()
{
    NVIC_DisableIRQ(LPUART1_IRQn);

```

```

    __LPUART1_CLK_DISABLE();
    JumpAddress = *(__IO uint32_t*) (APPLICATION_ADDRESS + 4);
    JumpToApplication = (pFunction) JumpAddress;
    __set_MSP(*(__IO uint32_t*) APPLICATION_ADDRESS);
    JumpToApplication();
}

void SendMsg(char* msg)
{
    strcpy(USART.TxBuffer, msg);
    USART.TxCounter = strlen(msg);
    TransmitLPUSART();
    while (USART.FlagPacketTransmitEnable == 0) continue;
}

int ihex_parse_single_record(uint8_t* data, unsigned int length,
ihex_record_t* record)
{
    if (*(data) != IHEX_CHR_RECORDMARK) return IHEX_ERR_PARSE_ERROR;
    record->ihr_length = (ihex_rtype_t) Hex16ToUint (data + 1);
    record->ihr_address = (ihex_addr_t) Hex32ToUint (data + 3);
    record->ihr_type = (ihex_rtype_t) Hex16ToUint (data + 7);
    record->ihr_checksum = (ihex_rchks_t) Hex16ToUint (data + 9 + record-
>ihr_length * 2);
    if (( data[11 + record->ihr_length * 2] != 0x0D
        || data[12 + record->ihr_length * 2] != 0x0A)
        && (data[11 + record->ihr_length * 2] != 0x0A))
        return IHEX_ERR_WRONG_RECORD_LENGTH;
    for (uint_t i = 0; i < record->ihr_length; i++)
    {
        if (data[9 + i*2] == 0x0A || data[9 + i*2] == 0x0D)
            return IHEX_ERR_WRONG_RECORD_LENGTH;
        record->ihr_data[i] = Hex16ToUint (data + 9 + i*2);
    }
    if (ihex_check_record(record) != 0)
        return IHEX_ERR_INCORRECT_CHECKSUM;
    return 0;
}

int ihex_check_record(ihex_record_t *r)
{
    uint8_t t = 0;
    t += r->ihr_length + ((r->ihr_address >> 8) & 0xFF)
        + (r->ihr_address & 0xFF) + r->ihr_type + r->ihr_checksum;
    for (uint_t i = 0; i < r->ihr_length; i++)
        t += r->ihr_data[i];
    return ((t & 0xFF) == 0) ? 0 : 1;
}

void TransmitLPUSART(void)
{
    USART.FlagPacketTransmitEnable = 0;
    USART.TxCounterTransmit = 1;
    LPUART1->TDR = (uint8_t) USART.TxBuffer[0];
}

void LPUART1_IRQHandler(void)
{
    if (__HAL_UART_GET_IT(&UartHandle, UART_IT_RXNE) != RESET)
    {

```

```

        unsigned char buffer;
        buffer = LPUART1->RDR;
        USART.TimeoutCounter = 1;
        switch (USART.StateRX)
        {
            case 0:
            {
                if (buffer == ':')
                {
                    USART.RxCounter = 0;
                    USART.RxBuffer[USART.RxCounter++] = buffer;
                    USART.StateRX = 1;
                }
                break;
            }
            case 1:
            {
                USART.RxBuffer[USART.RxCounter++] = buffer;
                USART.StateRX = 1;
                if (buffer == '\n')
                {
                    USART.TimeoutCounter = 0;
                    USART.StateRX = 0;
                    USART.FlagPacketEnable = 1;
                }
                break;
            }
        }
    }

    if (__HAL_UART_GET_IT(&UartHandle, UART_IT_TC) != RESET)
    {
        __HAL_UART_CLEAR_IT(&UartHandle, UART_CLEAR_TCF);
        if (USART.TxCouterTransmit < USART.TxCounter)    LPUART1->TDR    =
(uint8_t) USART.TxBuffer[USART.TxCouterTransmit++];
        else USART.FlagPacketTransmitEnable = 1;
    }

    if (__HAL_UART_GET_IT(&UartHandle, UART_IT_ORE) != RESET)
    {
        __HAL_UART_CLEAR_IT(&UartHandle, UART_CLEAR_OREF);
        USART.StateRX = 0;
        USART.RxCounter = 0;
        USART.FlagPacketEnable = 0;
    }
}

// Содержимое файла main.h

#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

#include "stm32l0xx_hal.h"
#include "string.h"
#include "flash_if.h"
#include "mgm.h"

```

```

#include "LoadProc.h"
#include "support.h"

#define VERSION_ADDRESS (uint32_t)(APPLICATION_ADDRESS - FLASH_PAGE_SIZE)
#define VERSION_EPR (*(__IO uint32_t*) VERSION_ADDRESS)

#define VERSION_KEY_ADDRESS (VERSION_ADDRESS + 4)
#define VERSION_KEY_EPR (*(__IO uint32_t*) VERSION_KEY_ADDRESS)

#define VERSION_KEY 0xAB123456

extern UART_HandleTypeDef UartHandle;

void Error_Handler(void);

#ifdef __cplusplus
}
#endif

#endif /* __MAIN_H */

// Содержимое файла main.c
#include "main.h"

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_LPUART1_UART_Init(void);

#define RDP
void SetReadOutProtection(void)
{
    FLASH_OBProgramInitTypeDef Optbyte;
    HAL_FLASHEx_OBGetConfig(&Optbyte);
    if (Optbyte.RDPLevel == OB_RDP_LEVEL_0) {
        Optbyte.OptionType = OPTIONBYTE_RDP;
        Optbyte.RDPLevel = OB_RDP_LEVEL_1;
        HAL_FLASH_OB_Unlock();
        HAL_FLASHEx_OBProgram(&Optbyte);
        HAL_FLASH_OB_Lock();
    }
}

int main(void)
{
    HAL_Init();
    SystemClock_Config();
#ifdef RDP
    SetReadOutProtection();
#endif
    MX_GPIO_Init();
    MX_LPUART1_UART_Init();
    Loader.WaitCounter = 1;
    while (1)
    {
        LoadProc();
    }
}

void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};

```

```

RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};
__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
RCC_OscInitStruct.HSISState = RCC_HSI_ON;
RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
RCC_OscInitStruct.PLL.PLLMUL = RCC_PLLMUL_4;
RCC_OscInitStruct.PLL.PLLDIV = RCC_PLLDIV_2;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
    Error_Handler();
}
RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
{
    Error_Handler();
}
PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_LPUART1;
PeriphClkInit.Lpuart1ClockSelection = RCC_LPUART1CLKSOURCE_PCLK1;
if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
{
    Error_Handler();
}
}

static void MX_LPUART1_UART_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct;
    __GPIOB_CLK_ENABLE();
    __LPUART1_CLK_ENABLE();

    GPIO_InitStruct.Pin = GPIO_PIN_10 | GPIO_PIN_11;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    GPIO_InitStruct.Alternate = GPIO_AF4_LPUART1;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

    __HAL_RCC_LPUART1_CONFIG(RCC_LPUART1CLKSOURCE_HSI);
    UartHandle.Instance = LPUART1;
    UartHandle.Init.BaudRate = 115200;
    UartHandle.Init.WordLength = UART_WORDLENGTH_8B;
    UartHandle.Init.StopBits = UART_STOPBITS_1;
    UartHandle.Init.Parity = UART_PARITY_NONE;
    UartHandle.Init.Mode = UART_MODE_TX_RX;
    UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    UartHandle.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
    UartHandle.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
    if (HAL_UART_Init(&UartHandle) != HAL_OK)
    {
        Error_Handler();
    }
    __HAL_UART_ENABLE_IT(&UartHandle, UART_IT_RXNE);
}

```

```

    __HAL_UART_ENABLE_IT(&UartHandle, UART_IT_TC);
    NVIC_EnableIRQ(LPUART1_IRQn);
}

static void MX_GPIO_Init(void)
{
    __HAL_RCC_GPIOB_CLK_ENABLE();
}

void Error_Handler(void)
{
    __disable_irq();
    while (1)
    {
    }
}

```