

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт
Факультет приборостроения,
информационных технологий и
электроники

Кафедра
«Информационная безопасность
систем и технологий»

Специальность 100503 –

Информационная безопасность
автоматизированных систем

Специализация -

Защищенные автоматизированные
системы управления

ДИПЛОМНЫЙ ПРОЕКТ

на тему

МОДУЛЬ СРЕДСТВА КРИПТОГРАФИЧЕСКОЙ ЗАЩИТЫ ИНФОРМАЦИИ
НА БАЗЕ ПРОЦЕССОРА MULTICLET R1

Студент

Улитин Владислав Васильевич

Руководитель

Варламов В.Б.

Нормоконтролёр

Иванов А.П.

Рецензент

Петрушков В.С.

Работа допущена к защите (протокол заседания кафедры от 2006/17 № 19)

Заведующий кафедрой

С.Л. Зефилов

Работа защищена с отметкой
от 23.06.17 № 15)

хорошо

(протокол заседания ГЭК

Секретарь ГЭК

А.П. Иванов

Иванов
10.06.
21.06.17г.

Пенза 2017

РЕФЕРАТ

Пояснительная записка содержит 108 страниц, 15 рисунков, 10 таблиц, 7 источников.

СРЕДСТВО КРИПТОГРАФИЧЕСКОЙ ЗАЩИТЫ ИНФОРМАЦИИ, КОНТРОЛЬ ЦЕЛОСТНОСТИ, ФУНКЦИЯ ХЭШИРОВАНИЯ, ГОСТ Р 34.11-2012, ГОСТ 34.12-2015, ГОСТ 34.13-2015, ЭТАЛОННЫЕ ЗНАЧЕНИЯ, ПРОГРАММНОЕ СРЕДСТВО.

Объектом исследования являются принципы разработки модуля СКЗИ и осуществление контроля целостности файловой системы модуля.

Целью дипломного проекта является исследование основных возможностей процессора MultiClet R1 и элементной базы российских производителей, и разработка модуля средства криптографической защиты информации на базе данного процессора с использованием вышеупомянутой базы.

В процессе работы был проведен обзор существующей элементной базы и российских механизмов контроля целостности информации, осуществлен их анализ, выбран один из блочных алгоритмов, работающий в режиме имитовставки, для реализации контроля целостности программного обеспечения.

В результате были разработаны структурная и принципиальная схемы модуля криптографической защиты информации на базе процессора MultiClet R1 и реализовано программное средство для контроля целостности программного обеспечения, хранящегося в ППЗУ программ, контроллере хранения ключевой информации, абонентском модуле и контроллере управления и индикации, с использованием алгоритма ГОСТ 34.12-2015 «Магма» в режиме имитовставки по ГОСТ 34.13-2015. Проверка целостности ПО производится на ключе НСД, отличным от основной ключевой информации модуля. Данный модуль планируется использовать в одной из новых разработок АО «ПНИЭИ».

СОДЕРЖАНИЕ

Введение	6
1 Средства криптографической защиты информации	8
1.1 Описание средств криптографической защиты информации.	8
1.2 Сфера применения криптографических средств информации.....	11
2 Процессор MultiClet R1	14
2.1 Описание процессора MultiClet R1	14
2.2 Организация вычислительного процесса в MultiClet R1	20
2.3 Устройство памяти MultiClet R1	25
2.4 Отладчик для процессора MultiClet R1	26
2.5 Сравнение процессора MultiClet R1 с аналогами.....	27
2.6 Исследования процессора MultiClet R1 в НИС-321 АО «ПНИЭИ»	30
3 Разработка схем для модуля СКЗИ.....	33
3.1 Разработка структурной схемы модуля.....	33
3.1.1 Анализ требований для построения модуля средства криптографической защиты информации.....	33
3.1.2 Описание структурной схемы модуля криптографической защиты информации	38
3.2 Разработка принципиальной схемы модуля СКЗИ.....	41
3.2.1 Выбор элементной базы для построения модуля СКЗИ.....	41
3.2.2 Построение принципиальной схемы модуля СКЗИ.....	46
4 Проверка целостности модуля криптографической защиты информации	47
4.1 ГОСТ Р 34.11-2012	47
4.2 ГОСТ Р 34.12-2015.....	50
4.3 ГОСТ Р 34.13-2015.....	54
4.4 Реализация ГОСТ 34.12-2015 в режиме имитовставки ГОСТ 34.13- 2015.....	56
Заключение.....	64
Список использованных источников.....	65

Приложение А (обязательное) Структурная схема блока СКЗИ.....	66
Приложение Б (обязательное) Перечень элементов, предназначенный для создания модуля СКЗИ	68
Приложение В (обязательное) Принципиальная схема модуля СКЗИ.	78
Приложение Г (обязательное) Алгоритм ГОСТ 34.12-2015 «Магма», работающий в режиме имитовставки по ГОСТ 34.13-2015.....	87
Приложение Д (обязательное) Код программы проверки целостности основанный на алгоритме ГОСТ 34.12-2015 «Магма», работающий в режиме имитовставки по ГОСТ 34.13-2015	89

ОБОЗНАЧЕНИЯ И СКОРАЩЕНИЯ

В настоящем отчете применяют следующие обозначения и сокращения:

НСД – несанкционированный доступ;

ОС – операционная система;

ПО – программное обеспечение;

ЦПУ – центральное процессорное устройство;

УК – устройство контроля;

АЛУ – арифметико-логические устройства;

ФР – файл регистров;

УДП – устройство доступа в память;

СОЗУ – сверхоперативное запоминающее устройство;

ОЗУ – оперативное запоминающее устройство;

ПЗУ – постоянное запоминающее устройство;

ППЗУ – последовательное постоянное запоминающее устройство;

СКЗИ – средство криптографической защиты информации;

РПУ – речепреобразующее устройство;

РПЗУ – репрограммируемое постоянное запоминающее устройство;

ПЛИС – программируемая логическая интегральная схема;

ЭРЭ – электрорадиоэлемент;

ИМС – интегральная микросхема.

ВВЕДЕНИЕ

Защита информации — это деятельность по предотвращению утечки защищаемой информации, несанкционированных и непреднамеренных воздействий на защищаемую информацию. [1]

Эта деятельность необходима для защиты конфиденциальности персональных данных, государственной тайны и другого рода информации подлежащей защите от доступа к ней неограниченного круга лиц, с целью нанесения вреда интересам личности, общества, государства.

Ввиду быстро развивающихся и уже развитых технических средств защита информации в современном мире становится не возможной без использования специальных средств защиты информации, таких как средства криптографической защиты информации.

Объектом исследования являются принципы разработки модуля СКЗИ и осуществление контроля целостности файловой системы модуля.

Целью данного дипломного проекта является исследование основных возможностей процессора MultiClet R1 и элементной базы российских производителей, и разработка модуля средства криптографической защиты информации на базе данного процессора с использованием вышеупомянутой базы.

Для достижения цели выпускной квалификационной работы были поставлены следующие задачи:

- анализ предназначения и сферы применения СКЗИ;
- исследование возможностей процессора MultiClet R1;
- разработка структурной схемы модуля СКЗИ;
- анализ отечественной элементной базы, необходимой для построения модуля СКЗИ;
- разработка принципиальной схемы модуля СКЗИ;
- анализ отечественных стандартов по обеспечению целостности информации;

- реализация механизма контроля целостности ПО модуля на языке C.

1 Средства криптографической защиты информации

1.1 Описание средств криптографической защиты информации

В ГОСТ Р 50922-2006. «Защита информации. Основные термины и определения» определены термины средство криптографической информации, средство защиты информации, защита информации.

Средства криптографической защиты информации (СКЗИ) – средство защиты информации, реализующее алгоритмы криптографического преобразования информации.

Средство защиты информации – техническое, программное, программно-техническое средство, вещество и (или) материал, предназначенные или используемые для защиты информации.

Защита информации – деятельность, направленная на предотвращение утечки защищаемой информации, несанкционированных и непреднамеренных воздействий на защищаемую информацию. [1]

В дополнении к этим терминам определён термин криптографического преобразования по ГОСТ 28147-89.

Криптографическое преобразование – преобразование данных при помощи шифрования и (или) выработки имитовставки. [2]

Криптографическое преобразование информации позволяет реализовать два из трёх основных свойств информации подлежащей защите: обеспечение конфиденциальности и целостности. Не задействуется только свойство доступности информации. Два вышеперечисленных свойства очень тесно взаимосвязаны и нередко при реализации обеспечения одного свойства информации обеспечивается и второе свойство.

Существует много классификаций методов криптографического преобразования информации, одна из таких это по виду воздействия на исходную информацию. Они приведены на рисунке 1.

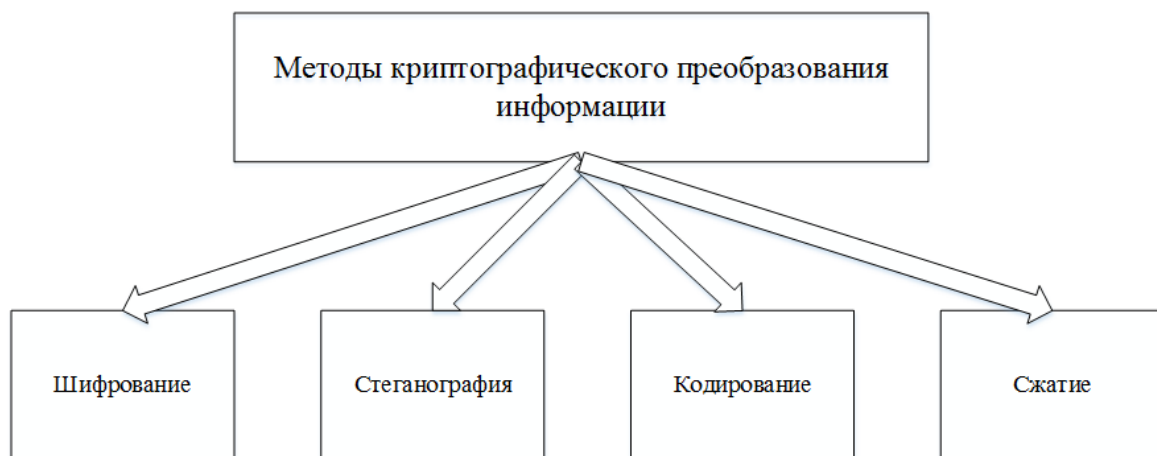


Рисунок 1 – Классификация методов криптографического преобразования информации

На рисунке 1 представлены следующие методы:

- шифрование;
- стеганография;
- кодирование;
- сжатие.

Шифрование заключается в проведении обратимых математических, логических и других операций над первоначальной информацией, в результате которых эта информация становится случайным набором букв и символов. Как правило для метода шифрования необходим алгоритм преобразования информации и ключ, на котором непосредственно и будет производиться шифрование и обратная ей операция – расшифрование.

Стеганография заключается не только в процессе сокрытия передаваемой информации, но и самого факта передачи информации. В основе всех этих методов лежит маскирование закрытой информации среди открытых файлов. Стеганографию также можно использовать совместно с другими методами, например, с шифрованием, для повышения вероятности того, что информация не будет перехвачена.

Кодирование заключается в замене исходных слов, предложений заранее оговорёнными кодами, в качестве которых могут использоваться сочетания

букв, цифр, знаков. В информационных сетях кодирование применяется для повышения достоверности передаваемой информации.

Сжатие заключается в сокращении объёма информации. Сжатая информация не может быть прочитана или использована без обратного преобразования. Так как средства сжатия и обратного преобразования доступны, эти методы нельзя рассматривать как надежные средства криптографического преобразования информации.

Исходя из всего вышеперечисленного становится ясно, что основное предназначение средств криптографической защиты информации — это не допустить получение защищаемой информации лицами, для которых она не предназначена. Эта защита осуществляется с помощью криптографических преобразований над информацией в специальных системах как односторонних, таких как преобразования с помощью хеш-функций, и обратимых, таких как потоковые шифры и другие.

Современные средства криптографической защиты информации решают следующие задачи:

- генерация и использование ключевой информации;
- шифрование и расшифровывание данных;
- электронная цифровая подпись и все связанные с ней задачи;
- идентификация и аутентификация.

В свою очередь шифрование и расшифрование средствами криптографической защиты информации осуществляется с помощью одного или нескольких методов защиты информации:

- программного, состоящего из реализации криптографических алгоритмов на языках высокого или низкого уровня в виде функций, библиотек или программ;
- аппаратного, составляющего реализацию крипто-алгоритмов или их отдельных участков в микросхемах, процессорах, модулях, совмещёнными со

средствами вычислительной техники или встроенными в автоматизированные системы;

– программно-аппаратного, состоящего из комплексов взаимосвязанных программных и аппаратных частей с модулями защиты информации.

1.2 Сфера применения средств криптографической защиты информации

На сегодняшний день сфера применения СКЗИ затрагивает два основных направления: гражданское и военное.

Первое направление характеризуется в первую очередь применением СКЗИ для обеспечения безопасности ПДн.

Для их выполнения требований по защите персональных данных руководители компаний, сталкиваются с необходимостью формирования модели угроз персональным данным и разработки на ее основе системы защиты персональных данных, в состав которой должно входить средство криптографической защиты информации.

Согласно документу, утверждённому руководством 8 Центра ФСБ России 31 марта 2015 года под названием: «Методические рекомендации по разработке нормативных правовых актов, определяющих угрозы безопасности персональных данных, актуальные при обработке персональных данных в информационных системах персональных данных, эксплуатируемых при осуществлении соответствующих видов деятельности», использование СКЗИ для обеспечения безопасности персональных данных необходимо в следующих случаях:

– если персональные данные подлежат криптографической защите в соответствии с законодательством Российской Федерации;

– если в информационной системе существуют угрозы, которые могут быть нейтрализованы только с помощью СКЗИ.

Государство четко указывает, что при передаче персональных данных по каналам связи заинтересованные в этом стороны должны применять сертифицированные ФСБ России и ФСТЭК России средства криптографической защиты информации.

В военной сфере средства криптографической защиты информации применяются для скрытого управления войсками. Скрытое управление ставит перед собой цель сохранить в тайне от противника информацию, используемую при руководстве войсками в мирное и военное время. Основным техническим средством скрытого управления войсками является шифровальная аппаратура связи.

Шифровальная аппаратура связи может быть представлена четырьмя конструктивными решениями.

Первое заключается в том, что аппаратура создается в виде автономного блока криптографической защиты, к которому подключаются аналоговые средства связи и цифровые, хранящие данные без собственных защитных компонентов. При передаче речевой информации в блоке производится речепреобразование, затем информация подвергается криптографическому преобразованию, кодируется от помех и передаётся по проводам или радиоканалу. При приеме речевой информации операции выполняются в обратном порядке.

Второе решение – это построение аппаратуры в виде криптомодуля, который вставляется в любое внешнее приемопередающее или шифровальное оборудование. Если внедрить единый стандарт во все конструктивные и информационные интерфейсы модуля, то возможно применять криптомодуль во многих средствах проводной и радиосвязи, а также в технике предварительного шифрования.

Третье решение – схемы засекречивания, которые располагаются на отдельной плате и становятся составной частью шифровального средства связи. Функционально этот вариант мало чем отличается от второго. Если плата

унифицирована, то достаточно просто может использоваться в различных цифровых средствах, как во втором решении. Но если она не извлекается, то каждый экземпляр внешнего оборудования подлежит поэкземплярному учёту как средство засекреченной связи или шифрования.

Четвёртое решение – это, когда схемные и программные криптокомпоненты совмещены с другими функциональными узлами прибора, так, что даже нельзя отделить их в платы и модули. Такая схема делает аппаратуру более компактной и, применяется в корпоративных системах с консервативным составом средств связи. Каждый экземпляр такой аппаратуры подлежит индивидуальному учёту. В аппаратуре криптографического преобразования информации в качестве алгоритмов шифрования/расшифрования применяются блочные алгоритмы.

2 Процессор MultiClet R1

2.1 Описание процессора MultiClet R1

Мультиклеточное процессорное ядро (МСс) – первое процессорное ядро с принципиально новой (пост-неймановской) мультиклеточной архитектурой. Предназначено для решения задач управления и цифровой обработки сигналов в приложениях, требующих минимального энергопотребления и высокой производительности. [3]

Клетка — это аналог ядра в привычном ЦПУ. Она содержит своё УК, АЛУ. У клеток есть доступ к общим ФР и УДП, или, как альтернатива, они могут иметь свои собственные, которые могут работать в режиме зеркал или RAID-5. Хотя в таких режимах ФР будет работать заметно медленней, производительность МСс существенно это не снизит, так как основной обмен данными в ходе вычисления идёт не через ФР и шунты, а через коммутатор, не являющееся памятью устройства.

Главная особенность клеток – это то, что их УК могут, работая по особому протоколу и с особым представлением программы, вместе составлять одно распределённое УК, которое может выполнять один поток управления. Выполнение этого потока происходит в параллельном, асинхронном, совмещённом режиме, когда одновременно происходят: выборка инструкций, работа с памятью и ФР, арифметические преобразования, вычисление цели перехода. Вследствие чего эти УК намного более сильно существеннее и проще, чем УК суперскалярных процессоров в очередном исполнении современными фирмами производителями. Они тоже способны на такое параллельное исполнение программы, но за счёт сложности и централизованности.

Вследствие всего вышеперечисленного вытекает два важных преимущества данного вида процессоров:

- энергоэффективность, так как мультиклеточный процессор способен выполнять за 1 такт 4 инструкции без использования сложных и больших УК;

- отказоустойчивость, если в традиционном процессоре УК выйдет из строя, то и весь процессор прекратит свою работу, если это произойдет в мультиклеточном процессоре, то выйдет из строя только одна из клеток, но процесс вычисления может продолжаться на оставшихся клетках, но заметно медленнее;

Количество клеток, из которых может состоять ядро равно 4, 8, 16 или 32, объединенных специальной интеллектуальной коммутационной средой. Клетки имеют систему команд, построенную на базе языка триад.

Архитектурные особенности данного вида процессорных ядер заключается в следующем:

- отсутствию в мультиклеточной архитектуре перед фон-неймоновской необходимости упорядоченного размещения описаний операций в программе, то есть необходимо непосредственно указывать информационные связи в программе, что в свою очередь приводит к облегчению проектирования процессора и инструментальных программных средств;

- мультиклеточная архитектура в сравнении с известными не фон-неймоновскими архитектурами использует традиционные императивные языки программирования, использует для указания информационных связей не адреса команд, а динамически формируемые значения и последовательным способом выборки команд;

- система команд клетки является аппаратной реализацией входного языка программирования;

- неупорядоченность промежуточных представлений компилируемых программ обеспечивает получение после каждой компиляции индивидуального объектного кода для каждого процессора. Это, а также замкнутость подмножеств промежуточных представлений, резко ограничивают

возможности незаметного и несанкционированного вмешательства злоумышленников в работу системного программного обеспечения;

- промежуточные представления компилируемых программ обеспечивают «естественную» реализацию параллелизма, т.е. возможность одновременного чтения и исполнения нескольких команд без анализа очередности выполнения и их информационной связности;

- интеллектуальная полносвязная коммутационная среда, которая работает ширококлеточно, не нарушает работу межклеточного обмена данными;

- программа, прошедшая процедуру компиляции может быть выполнена на любом количестве клеток, что обеспечивает непрерывную работу как при отказе клеток процессора, так и при подключении дополнительных вычислительных ресурсов;

- организация данного процессора является асинхронной и децентрализованной, как на системном уровне (между клетками процессора), так и на внутриклеточном уровне (между блоками клетки при реализации команд).

Микропроцессор МСр042R100102-LQ256 имеет в своем составе мультিকлеточное процессорное ядро. Ядро мультиклеточного процессора содержит 4 клетки – когерентных процессорных блоков, объединенных интеллектуальной коммутационной средой. МП имеют динамически реконфигурируемое ядро, распределение вычислительных ресурсов (клеток) по задачам производится согласно алгоритму заложенному в исполняемой программе.

При этом один и тот же код может выполняться любым количеством и комбинацией клеток. Мультиклеточный процессор предназначен для решения широкого круга задач управления и цифровой обработки сигналов в приложениях, требующих минимального энергопотребления и высокой производительности.

Внешний вид процессора представлен на рисунке 2.



Рисунок 2 – Внешний вид процессора MultiClet R1

Основные технические характеристики процессора приведены в таблице 1.

Таблица 1 – Технические характеристики

Условное обозначение	MCp042R100102-LQ256
Основное функциональное назначение	64-разрядная микро-ЭВМ
Максимальная тактовая частота ядра	100 МГц
Мультиклеточное ядро	4 клетки 32/64 разряда
Блок вычисления чисел с плавающей точкой (соответствует ieee754)	двойной точности
Производительность пиковая	24 MFLOPS/МГц
Встроенная память СОЗУ, Кбайт	512

Продолжение таблицы 1

Условное обозначение	МСр042R100102-LQ256
Внешняя шина памяти, бит	32
Внешняя шина памяти, типы памяти	PROM, SRAM, SDRAM, IO
USB 2.0 HS device, шт.	1
UART, шт.	4
SPI, шт	2
I2C	1 ведущий, 1 ведомый
I2S	1 ведущий/ведомый (прием/передача)
PWM, каналов	4
Ethernet 10/100, шт.	1
ЦАП, 12 бит, 125Msps, каналов	1
АЦП, 16 бит, 48Ksps, каналов	8
Пользовательские входы-выходы, шт.	159

МП имеет следующие основные части:

- ЦПУ;
- периферийные устройства, в качестве шины периферийных устройств, использующие шину AMBA 2.0;

- оперативные запоминающие устройства типа СОЗУ.

В свою очередь ЦПУ состоит из:

- ядра с 4 вычислительными клетками;
- коммутатора памяти;
- системных устройств;
- коммутатора системных регистров;
- аппаратной системы отладки ПО.

Структурная схема ЦПУ представлена на рисунке 3.

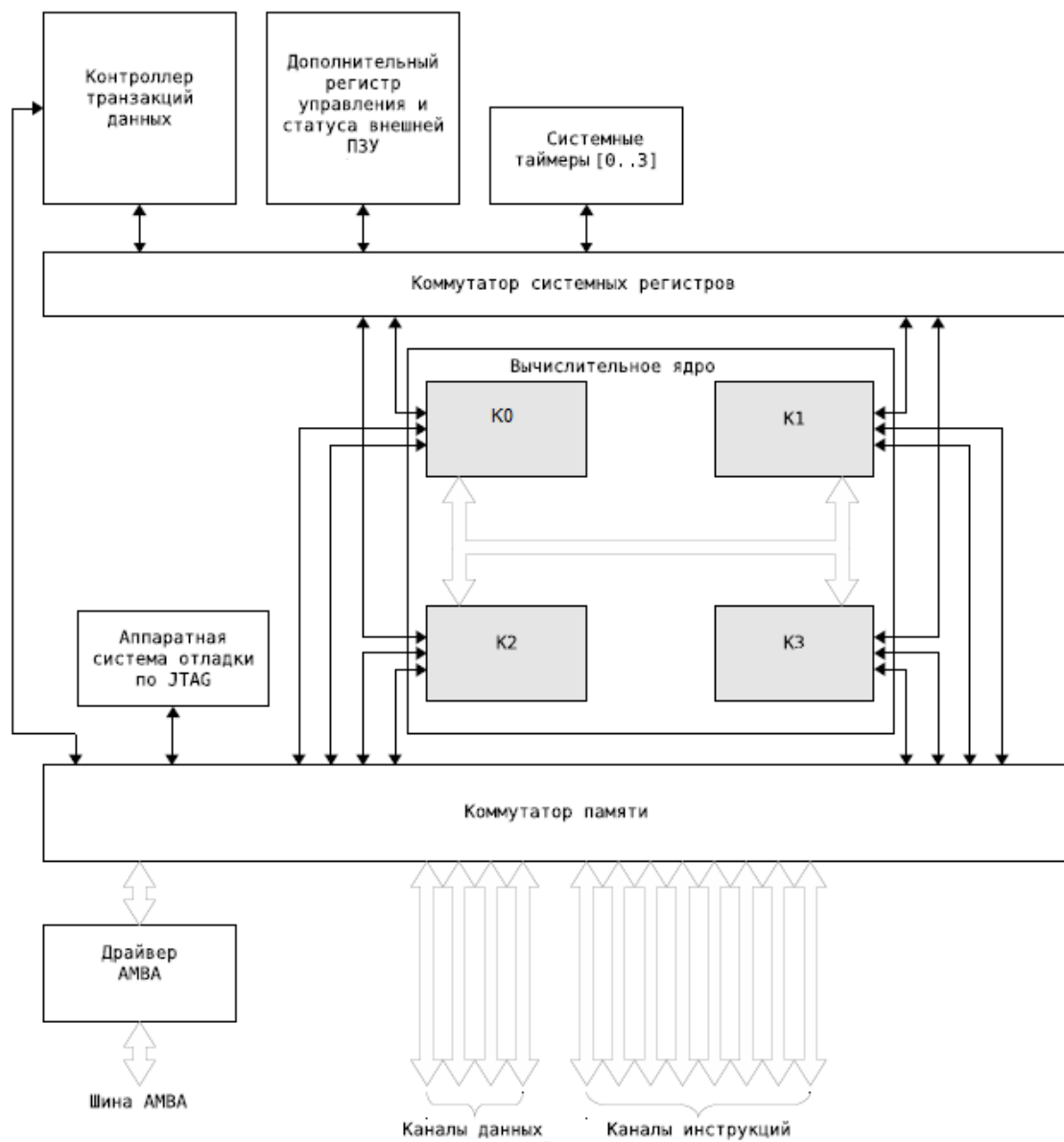


Рисунок 3 – Структурная схема ЦПУ MultiClet R1

2.2 Организация вычислительного процесса в MultiClet R1

Результатом работы процессора по выполнению некоторой команды является формирование некоторого результата. Использование этого результата командами, которые будут выполнены в будущем, образует информационную связь между командой–источником и командами–приемниками. Эта информационная связь может быть, как прямой, так и косвенной.

Все современные цифровые процессоры, кроме процессоров с мультиклеточной архитектурой, используют косвенную связь, при которой работа с результатом возможна только после его отчуждения. Возможно всего два способа реализации связи:

- запись результата командой–источником в общедоступные регистры и/или память, имя которых при написании программы задают в поле операнда команды–приемника (классическая фон-неймановская модель), поэтому процесс изменения состояний строго задаётся исполнением последовательности команд, определённых на этапе программирования;

- адресная рассылка, то есть запись результата в память по адресу в который, при реализации программы на одном из языков программирования, должна быть помещена команда–приемник. Типичная команда потокового процессора содержит: поле кода операции, поля двух операндов, одно или несколько полей адресов команд, которым передается результат выполненной операции, и поля номеров операндов, в которых он будет размещен. Команда выбирается и выполняется тогда, когда в ее поля операндов будут записаны все предшествующие результаты выполнения других команд, необходимых для ее выполнения. Следовательно, команды в программе могут располагаться независимо от местоположения. Этот принцип исполнения команд потокового процессора «по готовности» позволяет реализовать параллелизм без необходимости решения задачи распараллеливания.

Прямая связь – принцип, отличающийся от вышеперечисленных. При использовании такой связи команды идентифицируются и при этом имена

должны однозначно указывать на команду, а не ее местоположение или другие особенности реализации. Доступ к результатам может осуществляться по идентификаторам как команд–источников, так команд–приемников.

При использовании идентификации по командам-источникам используется широковещательная рассылка с последующим отбором каждого элемента, при которой в поле операнда команды–приемника задают имя команды–источника.

При использовании идентификации по командам-приёмникам выполняется поименная рассылка, которая в команде–источнике задаёт имя команды–приемника результата. Самый распространённый пример кодирования программы с использованием прямых информационных связей между командами является язык триад.

Строки программ представляются в виде триад, образующих записи с тремя полями:

$\langle op \rangle \langle arg1 \rangle \langle arg2 \rangle$, где op – поле кода операции; $arg1$ – поле первого операнда; $arg2$ – поле второго операнда.

В качестве операндов используются либо указатели на таблицу символов (имена, определенные программистом ранее, или константы), либо указатели на триады–источники выполненных ранее команд.

Последовательность операторов, написанной на языке высокого уровня и образующей линейный участок, представлена на рисунке 4.

$$a = b * c ;$$
$$d = a / e ;$$
$$f = (d + a) / 10 ;$$

Рисунок 4 – Линейная последовательность операторов

Представление на уровне триад указанного на рисунке 4 участка приведено в таблице 2.

Таблица 2 – Представление программы на уровне триад

Номер триады	Op	Arg1	Arg2
0	*	b	c
1	=	a	(0)
2	/	(1)	e
3	=	d	(2)
4	+	(3)	(1)
5	/	(4)	10
6	=	f	(6)

Именованные константы – это указатели на таблицу символов, а числа в скобках – это указатели на триады.

Развитием языка триад является представление программы в котором она делится на совокупность параграфов. Внутри параграфа информационная связь между его командами только прямая. При этом команды параграфа образуют частично упорядоченную и логически замкнутую линейную последовательность, т.е., команды-источники предшествуют своим командам-приемникам, а команды-приемники имеют ссылки только на команды-источники своего параграфа. Следовательно, информационная связь между командами различных параграфов осуществляется косвенно, через память.

Ссылки рассчитываются как значение разницы между номерами команды приемника и команды-источника, полученными при последовательной нумерации команд параграфа. Так, например, фрагмент программы, приведенный в таблице 2, будет выглядеть как представлено в таблице 3.

Таблица 3 – Представление программы на уровне триад

Номер триады	Op	Arg1	Arg2
0	rd		b
1	rd		c

Продолжение таблицы 3

Номер триады	Op	Arg1	Arg2
2	rd		e
3	*	(0)	(1)
4	=	a	(2)
5	/	(4)	(2)
6	=	d	(5)
7	+	(6)	(4)
8	/	(7)	10
9	=	f	(8)

Последнюю команду параграфа отмечают управляющим признаком «конец параграфа».

Особенности, определяющие организацию вычислительного процесса и аппаратную реализацию ядра процессора, заключаются в следующем:

- команды, заключённые в последовательность, имеют линейный характер, а каждая команда имеет свой номер, поэтому, по номеру команды и по значениям ссылок можно определить номера команд, результаты которых необходимы данной команде, для чего необходима упорядоченная выборка команд;
- информационные связи между командами указаны явно, это даёт возможность исполнять команды по готовности операндов и, следовательно, обеспечить естественную реализацию параллелизма, но также необходимо обеспечить хранение команды до момента ее выполнения и сохранность полученного результата до его передачи командам-приемникам;
- естественная реализация параллелизма предполагает использование нескольких процессорных устройств, ввиду того, что выполняемая команда не имеет информации о потребителях своих результатов, ее результаты должны

рассылаться всем потенциальным потребителям широковещательно, а команды-приёмники должны отбирать требуемые им результаты из общего потока данных, все эти особенности учитываются в ядре мультиклеточного процессора, состоящего из совокупности идентичных клеток, объединённых одной полносвязной коммутационной средой, в свою очередь каждая клетка включает устройство управления, буфер команд и набор исполнительных устройств.

Процесс исполнения команды включает три этапа:

- а) выборки командных слов;
- б) отбор результатов;
- в) исполнения команд.

Выборка командных слов, этап а), осуществляется устройством управления. При этом выполняются следующие действия:

Начиная с первой команды очередного параграфа, устройства управления всех клеток мультиклеточного процессора, синхронно выбирают и декодируют группу последовательно размещенных команд данного параграфа. При этом клетки группы выбирают команды соответственно – первая клетка группы выбирает первую команду последовательности, вторая – вторую и т.д. Если среди команд, которые находятся на обработке, есть команда с признаком «конец параграфа», то последующие выбранные команды группы не декодируются.

При декодировании каждой команде присваивается индивидуальная идентификационная метка, вычисляются идентификационные метки команд, результаты которых являются операндами декодируемой команды, формируется и передается запрос коммутатору на отбор этих результатов из их общего потока данных.

Если все клетки группы готовы к приему очередной команды, то декодируемая команда записывается в буфер команд, коммутатор запоминает запрос и выборка повторяется. Готовность клеток к приему может определяться

целым рядом факторов, например, занятостью тега, буфера команд и других факторов. Также процесс выборки следующей команды может быть остановлен из-за отсутствия адреса следующего параграфа, до того момента пока данный адрес не будет сформирован.

Отбор результатов, этап б), выполняется одним главным коммутатором, который состоит из совокупности коммутационных устройств, каждое из которых необходимо для обслуживания одной клетки процессора. Коммутационное устройство принимает запросы на отбор результатов, запоминает их идентификационные метки в памяти запросов, и, если среди уже поступивших результатов есть те, значение меток которых записано в памяти запросов, передает их в буфер команд. В памяти запросов соответствующая отметка удаляется, что знаменует о выполнении запроса.

Процесс выполнения команды, заключительный этап в), начинается, если в буфере команд есть хотя бы одна команда, для которой получены все операнды. При этом, если готовых к исполнению команд несколько, то действует принцип «первый пришёл, первый вышел» и выбирается, команда, которая была записана в буфер раньше всех, и эта команда передается в операционное устройство. Операционное устройство выполняет команду и выдает во все коммутационные устройства клеток её результат с меткой, равной метке команды. Этот процесс приостанавливается только тогда, когда в буфере нет готовых к исполнению команд, до тех пор, пока в буфер не поступят новые команды для исполнения.

2.3 Устройство памяти MultiClet R1

Память для мультиклеточного процессора с программной точки зрения представляет собой массив с байтовой адресацией, в диапазоне адресов $[0, 2^{32}]$. Память доступна вся сразу без деления на какие-либо части. Так же нет деления на память программ и память данных, но есть ограничения по размещению данных и программ.

Память процессора можно разделить на несколько составных частей:

- память на кристалле мультиклеточного процессора;
- внешнее ОЗУ;
- внешнее ПЗУ;
- область устройств ввода/вывода;
- шина периферийных устройств;
- область отображения адресов 0x00000000-0x4000000.

Память на кристалле представляет собой СОЗУ. В двух диапазонах 0x00000000-0x00040000 и 0x00040000-0x00080000 память разбита на одинаковые по объёму 4 региона. Различие заключается в физическом представлении, в первом случае 32 блока $33 * 2^{11}$, во втором же случае 16 блоков $64 * 2^{11}$. Оба диапазона позволяют иметь одновременный доступ к данным, расположенным в соседних блоках.

Для доступа к внешней ОЗУ, внешней ПЗУ, области устройств ввода/вывода используется специализированная шина внешней памяти. Диапазон адресов для данных видов памяти находится в пределах 0x20000000-0x80000000. Возможность выполнения и хранения данных в данных зонах предусмотрена, но это существенно снизит производительность процессора, так как максимальная ширина шины внешней памяти – 4 байта, а 4 клетки за один такт могут запрашивать из памяти до 32 байт данных.

Область отображения адресов используется контроллером прямого доступа к памяти на периферийной шине. Диапазон адресов находится в пределах 0x80000000-0xC0000000

В зоне памяти предназначенной для шины периферийных устройств расположены регистры периферийных устройств. В последних же адресах расположена информация о конфигурации периферии процессора.

2.4 Отладчик процессора MultiClet R1.

Для отладки программ, написанных для мультиклеточного процессора, используется отладчик mc-dbg. Отладчик производит отладку в режиме работы с программой-эмулятором мультиклеточного процессора. Данный отладчик

запускается из командной строки. Обязательным аргументом является исполняемый бинарный файл. В качестве аргументов команд допустимо использовать числа в восьмеричной, десятиричной и шестнадцатеричной форме.

Отладчик поддерживает все основные группы команд такие как: команды общего назначения, команды управления процессором, команды управления точками останова и другие.

Отладчик имеет и графический режим в Geany IDE, который в свою очередь даёт возможность написания программ на языках C и ассемблер.

2.5 Сравнение процессора MultiClet R1 с аналогами

Для наглядной оценки производительности и эффективности процессора в таблице 4 приведено сравнение процессора MultiClet R1 с отечественными и зарубежными конкурентами.

Таблица 4 – Сравнение процессоров

	Отечественные процессоры		Зарубежные процессоры		
Параметры	MultiClet R1	Элвис 1892BM5Я(М C-0226)	Texas Instruments TMS320C67 01	Analog Devices ADSP - TS201S	Intel Pentium 4
Архитектура	Мультиклеточная (не фон-неймоновская)	RISC + SIMD с VLIW командой (фон-неймоновская)	VLIW (фон-неймоновская)	Суперскаляр с VLIW командой (фон-неймоновская)	Суперскаляр (фон-неймоновская)
Структура /количество /состав ядер	Однородная / 4 / Мультиклет	Неоднородная /3/ MIPS32 + 2xELcore-26	-/1/C67x	-/1/-	-/1/-
Решение задачи распараллеливания	«Естественное», без использования программных или аппаратных средств	С применением программных средств	С применением программных средств	С применением аппаратных и программных средств	С применением аппаратных и программных средств

Продолжение таблицы 4

	Отечественные процессоры		Зарубежные процессоры		
Параметры	MultiClet R1	Элвис 1892BM5Я(М С-0226)	Texas Instruments TMS320C67 01	Analog Devices ADSP - TS201S	Intel Pentium 4
Количество операций выполняемых ядрами параллельно за один такт	4	1+2x6	8	24	
Время выполнения CFFT-256 с плавающей запятой (такты)	2136	2150		928	
Ресурс потребляемый CFFT-256 (операции)	8544	12900		22272	
Динамическая реконфигурация	есть	нет	нет	нет	нет
Тип корпуса	QFP256	HSBGA416	BGA-352	Ball BGA-576	FC-PGA2-423
Технологический процесс (мкм)	0,18	0,25	0,18	0,13	0,18
Температурный диапазон (в пластиковом корпусе)	от - 60 до +125° С	-от 65 до 85° С	от -40 до 105° С	от -40 до +85° С	от -40 до +85° С
Площадь кристалла (кв.мм)	144	155	НД	НД	217
Разрядность (бит)	32/64	32	32	32/64	32
Тактовая частота (МГц)	100	100	150	600	2000
Производительность (Гфлопс)	2,4	1,2	1	3,6	2

Продолжение таблицы 4

	Отечественные процессоры		Зарубежные процессоры		
Параметры	MultiClet R1	Элвис 1892BM5Я(MC-0226)	Texas Instruments TMS320C670 1	Analog Devices ADSP - TS201S	Intel Pentium 4
Производи- тельность (Гфлопс)	2,4	1,2	1	3,6	2
Память ПЗУ/ОЗУ (Кб)	-(ПД=256, ПП=256)	16/(ПД=128, ПП=16)	64/64	НД/3000	НД/НД
Память ПЗУ/ОЗУ (Кб)	-(ПД=256, ПП=256)	16/(ПД=128, ПП=16)	64/64	НД/3000	НД/НД
Напряжение Ядро/Пере- ферия (В)	1,8/3,3	2,5/3,3	1,8/3,3	1,2/3,3	1,8/НД
Максималь- ная потребля- емая мощность (Вт)	1,05	2,5	1,8	3,4	100
Арифметика с плавающей запятой двойной точности	+	-	-	-	+
SPI	2	-	-	-	-
UART	4	1	-	-	-
USB	2.0 FS (device)	-	-	-	-
АЦП	16бит,48К выб/сек, 8 каналов				
ЦАП	12 бит, 100M выб/сек, 1 канал				
Интерфейс внешней памяти	Шина данных 32 бита, SRAM, SDRAM, PROM, IO	Шина данных 32 бита, SRAM, SDRAM, SBSRAM FLASH,ROM	Шина данных 32 бита, SRAM,SDRA M,SBSRA M, EEPROM	Шина данных 64 бита, SDRAM	
Ethernet	Ethernet MAC 10/100Mб/с	-	-	-	-

2.6 Исследования процессора MultiClet R1 в НИС-321 АО «ПНИЭИ»

В НИС-321 АО «ПНИЭИ» ведущим инженером НИС-321 Короленя А.А. была проведена работа по изучению отечественного процессора «MultiClet R1». Он установил, что процессор разрабатывается в г. Екатеринбург. Сам кристалл процессора изготавливается в Малайзии на фабрике SilTerra.

В основу построения процессора «MultiClet R1» положена мультиклеточная архитектура, когда каждая из клеток независима и параллельно выполняет определённую ей компилятором часть программного обеспечения. В ходе исполнения ПО возможно динамическое перераспределение клеток в группы, при этом у программиста есть возможность перенаправить те или иные подпрограммы в определённую группу клеток. За счёт такой параллельности разработчиком процессора заявляется высокая производительность изделия.

Основные параметры «MultiClet R1» приведены в таблице 5.

Таблица 5 – Основные технические характеристики

Тип корпуса	LQFP-256
Технологический процесс	180 нм
Тактовая частота	100 МГц
Память программ	256 Кбайт
Память данных	256 Кбайт
Напряжение	1,8В - ядро 3,3В - периферия
Количество прерываний	44
АЦП	8 каналов, 16 бит, 48 Кбит/с
ЦАП	1 канал, 12 бит, до 100 Мбит/с

В НИС-321 были проведены следующие виды работ:

- изучена архитектура и периферийные устройства процессора;
- изучена система команд процессора;
- изучены средства разработки исполняемых кодов (С-компилятор, ассемблер, линкер);

- разработан С-модуль ПО речепреобразующего устройства (РПУ) комплекса «Разбег» на скорости 4.8 кбит/с;
- разработан С-проект РПУ;
- произведена отладка С-проекта на тестовом векторе в функциональной модели (симуляторе) интегрированной среды Geany IDE;
- при использовании отладочной платы LDM-HELPER-MB501 и субмодуля LDM-HELPER-MCP042RQ256 (включающего процессор «MultiClet R1») по интерфейсу JTAG был запущен разработанный в процессе отладки исполняемый код РПУ;
- проведено субъективное тестирование качества речи методом самопрослушивания при использовании микрофона и телефона, которые были подключены к соответствующим контактам АЦП и ЦАП процессора. Зацикливание речи в ПО РПУ обеспечивалось непосредственной передачей битового потока, вычисленного в анализаторе, на вход синтезатора;
- профилирование ПО и его последующая оптимизация на данном этапе разработки не производились.

Результаты тестирования РПУ приведены в таблице 7:

Таблица 6 – Результаты тестирования процессора MultiClet R1

Качество речи	Хорошее
Используемая память программ (включая константы)	90Кбайт (35% от имеющихся 256Кбайт в процессоре)
Используемая память данных	10Кбайт (4% от имеющихся 256Кбайт в процессоре)
Производительность РПУ	60 MIPS (54-анализ, 6-синтез) (60% от производительности процессора)

В результате можно сделать вывод, что к недостаткам «MultiClet R1» стоит отнести неполную завершенность разработки (последняя версия С-

компилятора в связи с ошибками в предыдущей версии вышла 18.08.2016 и она по словам разработчиков возможно не последняя), а также отсутствие отладочных средств, от чего во многом зависит время и качество отладки программистами прикладного ПО.

3 Разработка схем модуля СКЗИ

3.1 Разработка структурной схемы

Для наглядного представления основных элементов и узлов схемы и более лучшего понимания работы блока криптографической защиты необходимо разработать структурную схему данного модуля.

Схема состоит из совокупности частей, составляющих логически неделимые элементы, на которые она делится по определённому признаку. На основе разработанной структурной схемы строится всё дальнейшее проектирование блока криптографической защиты информации, включая принципиальную схему, алгоритмы и непосредственно сами программы, выполняющие определённые функции, заложенные в систему.

Схема на разных этапах жизненного цикла предназначена для:

- на этапе проектирования - для определения структуры будущего изделия;
- на этапе производства - для ознакомления с конструкцией изделия, разработки технологических процессов изготовления, монтажа и контроля изделия;
- на этапе эксплуатации - для определения неисправностей, ремонта и технического обслуживания изделия.

Важной составляющей частью разработки структурной схемы является анализ требований для построения модуля средства криптографической защиты информации.

3.1.1 Анализ требований для построения модуля средства криптографической защиты информации

Согласно техническому заданию целью проекта является разработка модуля средства криптографической защиты информации (СКЗИ) на основе процессора Multiclet R1, предназначенного для обеспечения криптографической защиты речевой информации и данных в составе

стационарного криптографического шлюза системы специальной подвижной радиосвязи.

Одно из основных требований, подлежащее первоочередному рассмотрению, является требование к габаритным размерам модуля – 310x170x50. Учитывая, что модуль должен быть разработан в основном на отечественной элементной базе, эти размеры являются довольно «жесткими», так как, к сожалению, размеры отечественных ЭРЭ в большинстве своем на много превосходят размеры импортных элементов. Исходя из этого, при выборе методов реализации составных узлов модуля и выборе элементной базы, следует обратить особое внимание на габаритные размеры комплектующих элементов.

При разработке модуля СКЗИ, помимо необходимости выполнения им своих непосредственных задач по защите конфиденциальной информации, следует учитывать ряд дополнительных требований, а именно:

- обеспечение фиксации фактов несанкционированного доступа к узлам модуля и хранящейся в нем информации;
- обеспечение защиты от проникновения защищаемой открытой, ключевой и криптографически-опасной информации по побочным высокочастотным сигналам и по электромагнитному излучению в цепи вторичного электропитания, сопрягаемые узлы шлюза и подключаемое оборудование.

В связи с этим при разработке модуля необходима установка на все входящие и выходящие цепи гальванической развязки и фильтров низких частот и экранировка модуля и его отсеков. При разработке конструкции и схемы модуля необходимо обеспечить срабатывание кнопок НСД при попытке разборки модуля.

Согласно ТЗ модуль должен обеспечивать ввод и хранение ключа НСД и ключевой информации и их стирание при факте НСД и аварийном стирании с органов управления. Следует учесть, что выполнение данных требований

должно обеспечиваться как при наличии внешнего питания, так и при его отсутствии. Это подразумевает под собой необходимость применение устройства, обеспечивающего хранение информации и возможность ее стирания от батарейного питания, при этом потребляющего минимально возможный ток.

Возможны следующие методы обеспечения выполнения данных требований:

а) применение статического ОЗУ с малым потреблением в режиме хранения информации и обеспечением стирания хранящейся в нем информации с использованием процессора, двух кнопок НСД (с учетом необходимости дублирования) и двух кнопок или сдвоенного тумблера (с учетом необходимости дублирования) аварийного стирания;

б) применение статического ОЗУ с малым потреблением в режиме хранения информации и размыканием его батарейного питания при НСД или при аварийном стирании;

в) применение микроконтроллера с необходимым объемом встроенного ОЗУ и малым потреблением в режиме хранения и работы.

Метод а) не подходит для нашего случая по причине большой потребляемой мощности процессора (порядка 2 Вт) в рабочем режиме.

Для реализации метода б) необходимо применение двух ОЗУ, для разделения режима стирания при НСД от режима аварийного стирания, при этом в одном ОЗУ буде храниться ключ НСД, а во втором – другая ключевая информация. Для реализации размыкания питания необходимо:

- для первого ОЗУ – две кнопки (с учетом необходимости дублирования), срабатывающих при разборке модуля;

- для второго ОЗУ – две кнопки (с учетом необходимости дублирования), срабатывающих при разборке модуля и сдвоенный тумблер (с учетом необходимости дублирования), срабатывающий при аварийном стирании. Применение тумблера обусловлено необходимостью обеспечения

максимально возможного времени снятия питания с ОЗУ для гарантированного рассасывания зарядов ячеек хранения для гарантированного стирания информации.

Итого мы имеем для второго метода две дополнительные кнопки и одно дополнительное ОЗУ.

С учетом достаточно высоких требований к габаритным размерам модуля, по нашему мнению, наиболее целесообразно применение метода в), предполагающего установку микроконтроллера, двух кнопок НСД и двух кнопок или сдвоенного тумблера аварийного стирания. Реализация данного метода позволит в полной мере обеспечить выполнение требований ТЗ в части хранения и стирания информации под батарейным питанием и в части габаритных размеров модуля.

Согласно ТЗ модуль СКЗИ должен обеспечивать возможность ввода ключевой информации (действующей и очередной) объемом не более 1,5 Мбайт. Для выполнения данного требования необходима установка в модуль перепрограммируемого запоминающего устройства (Flash-памяти) объемом не менее 12 Мбит. Для увеличения скорости обмена информацией и, соответственно, уменьшения времени криптографической синхронизации целесообразно применение ППЗУ с параллельным интерфейсом.

Согласно ТЗ модуль СКЗИ должен обеспечивать возможность ведения и хранения журнала событий. В журнале должны регистрироваться с фиксацией времени факты включения модуля, ввода ключевой информации, проверок целостности ПО, аварийных событий, смены паролей, установления соединений, разрывы соединений и их причины, результаты контроля.

Ориентировочный формат строки журнала приведен в таблице 7.

Таблица 7 – Ориентировочный формат строки журнала

Назначение информации	Начало строки	Дата, (ДД. ММ.ГГ)	Время, (ЧЧ. ММ.СС)	Код события	Крипто-номер абонента	Конец строки
Длина, байт	1	3	3	1	3	1

Как видно из таблицы 7, ориентировочная длина строки журнала составляет 12 байт. Предполагаемая частота возникновения событий – не более 100 в сутки (по опыту работы разрабатывающего подразделения), максимальный срок хранения журнала – 3 месяца (ориентировочный срок проведения регламентных работ). Следовательно, объем журнала – $12 * 100 * 3 * 30 = 108\,000$ байт = 864 кбит.

Для уменьшения габаритов модуля и, учитывая отсутствие необходимости высокой скорости записи и считывания информации журнала событий, целесообразно применение ППЗУ с последовательным интерфейсом. Объем ППЗУ следует рассчитывать исходя из необходимого объема журнала – 864 кбит, плюс 136 кбит на случай необходимости увеличения длины строки журнала. Итого – необходимый объем ППЗУ – 1 Мбит.

Для хранения загрузочной информации процессора «MultiClet R1» и двух ПЛИС необходимо - 256 кбайт + 2*100 кбайт = 456 кбайт = 4 Мбит.

Для уменьшения габаритов модуля и, учитывая отсутствие необходимости высокой скорости загрузки программного обеспечения процессора и прошивок ПЛИС, целесообразно применение ППЗУ с последовательным интерфейсом объемом не менее 4 Мбит.

Согласно ТЗ модуль должен иметь возможность ввода ключевой информации и вывод журнала событий по интерфейсу USB 2.0.

В процессоре «MultiClet R1» имеется в наличии встроенный контроллер USB 2.0, который поддерживает только функцию «Device» и реализует обмен в режиме Full Speed (12 Мбит/с) и High Speed (480 Мбит/с) с использованием внешнего приёмопередатчика, подключаемого по интерфейсу ULPI.

Для связи с внешней шиной USB необходимо использование дополнительного приёмопередатчика, подключаемого к процессору по интерфейсу ULPI.

Для выполнения требования ТЗ по подключению к модулю СКЗИ микротелефона, помимо микрофонного и телефонного усилителя, необходима установка фильтров низких частот и стабилизатора тока, позволяющего снизить влияние речевых сигналов на цепи питания модуля, тем самым обеспечивая дополнительную защиту информации. Необходимо предусмотреть возможность непрерывного (динамического) контроля стабилизатора тока с блокированием работы модуля при его неисправности.

При включении аппаратуры импульс тока по цепям электропитания не должен превышать 1,2 А, при длительности импульса не более 20 мс. Это требование накладывает необходимость применения в модуле ограничителя потребляемого тока при включении питания. Но, учитывая высокие требования к габаритам модуля и то, что модуль не имеет собственного тумблера включения питания и при эксплуатации не извлекается и не устанавливается в аппаратуру при включенном питании, целесообразно ограничитель тока установить в самой аппаратуре.

3.1.2 Описание структурной схемы модуля криптографической защиты информации

Исходя из требований ТЗ, разработанная схема, приведённая в приложении А, соответствует всем вышеперечисленным требованиям.

Рассмотрим назначение составных частей структурной схемы:

- центральный процессор предназначен для реализации алгоритмов речепреобразующего устройства (РПУ), криптографической синхронизации,

шифрования (расшифрования) информации, обеспечивает алгоритмическое управление узлами модуля по сообщениям от устройства индикации и клавиатуры, загрузку абонентской и канальной ПЛИС, обеспечивает контроль всех узлов модуля и проверку целостности программного обеспечения модуля и аппаратуры криптографическим методом с использованием ключа НСД;

- ПЛИС абонентская выполняет функции преобразователя информации, поступающей от абонентского модуля аппаратуры по стыкам UART (возможна организация нескольких каналов) в SPI интерфейс, для передачи на процессор речевой информации и команд управления, и обратно, обеспечивает зашумление принимаемой информации от абонентского модуля и от приемопередатчика USB, формирует частоту контроля стабилизатора тока;

- ПЛИС канальная выполняет следующие функции:

- 1) реализует функции портов ввода/вывода шифрованной информации для процессоров шифратора;

- 2) обеспечивает аппаратное сравнение информации основного и дублирующего тракта передачи;

- 3) обеспечивает зашумление приемных цепей для исключения возможности внешнего воздействия на элементы модуля с целью активации возможных программных и аппаратных закладок.

- ввиду отсутствия в процессоре «MultiClet R1» USB-приемопередатчика, обеспечивающего поддержку физической части USB интерфейса, для взаимодействия модуля с ПЭВМ и УВК необходимо применение отдельного USB-приемопередатчика, обеспечивающего преобразование USB интерфейса в ULPI-интерфейс, поддерживаемый процессором;

- контроллер НСД обеспечивает хранение во внутреннем ОЗУ ключа защиты от НСД. Контроллер имеет резервное питание от встроенной батареи. При срабатывании кнопок НСД (при попытке разборки модуля или при попытке доступа к аккумуляторной батарее) контроллер обеспечивает стирание

всей хранящейся в нем информации, включая ключ НСД, тем самым обеспечивая фиксацию факта НСД, который может быть обнаружен при проверке целостности ПО. Также контроллер обеспечивает контроль напряжений, питающих процессор, ПЛИС, контроллер и напряжения батареи;

- ППЗУ программ реализуется на микросхеме РПЗУ с последовательным интерфейсом и обеспечивает хранения кодов программного обеспечения процессора и прошивок всех ПЛИС;

- ППЗУ ключевое реализуется на микросхеме РПЗУ с параллельным интерфейсом и обеспечивает хранение ключевой информации;

- ППЗУ журнала реализуется на микросхеме РПЗУ с последовательным интерфейсом и обеспечивает хранение журнала событий;

- генератор формирует частоту необходимую для функционирования процессора и узлов, реализованных в ПЛИС;

- узлы гальванической развязки реализованы с использованием проходных фильтров низких частот и быстродействующих оптоэлектронных инверторов и предназначены для исключения проникновения побочной информации модуля в сопрягаемые узлы аппаратуры;

- стабилизаторы напряжения обеспечивают формирование гальванически развязанных напряжений постоянного тока для питания узлов модуля;

- фильтры вторичного электропитания обеспечивают защиту от проникновения высокочастотных сигналов в цепи питания модуля;

- усилитель МКФ обеспечивает усиление речевого сигнала в микрофонных цепях микротелефонной трубки (МТТ) до уровня минус 8 дБ. Усилитель ТЛФ обеспечивает усиление до заданного уровня речевого сигнала в телефонных цепях МТТ;

- стабилизатор тока предназначен для стабилизации тока потребления от сети питания. Напряжением с выхода стабилизатора тока обеспечивается питание аналоговых узлов, осуществляющих усиление речевых

сигналов. Стабилизатор тока содержит в своем составе схему контроля, обеспечивающую динамический контроль параметров стабилизатора тока и формирующую, при его неисправности, сигнал аварии в виде прекращения выдачи контрольной частоты со своего выхода.

3.2 Разработка принципиальной схемы

3.2.1 Выбор элементной базы для построения модуля СКЗИ

Основываясь на разработанной структурной схеме модуля, приведённой в приложении А, разработан перечень элементной базы для построения принципиальной схемы модуля СКЗИ. Перечень приведён в приложении Б.

В соответствии с ТЗ модуль СКЗИ должен быть разработан с использованием преимущественно отечественной элементной базы. Также все выбранные элементы должны выпускаться производителями с приемкой «5» или «ВП» связи с этим, а также учитывая требования к габаритным размерам элементов, был проведен выбор элементной базы для построения принципиальной схемы модуля.

В качестве центрального процессора в соответствии с ТЗ используется процессор «MultiClet R1» (МСр042R100102-LQ256). Для обеспечения процессора рабочей частотой допускается использование кварцевого резонатора, в качестве которого был выбран резонатор РК463-6ДС-8000 К (размеры 6х3,5х1,2 мм). Процессор имеет в своем составе умножитель частоты и может сгенерировать выходную частоту до 300 МГц. Учитывая это, было принято решение отказаться от внешнего генератора и использовать для тактового питания элементов схемы выходную частоту процессора, поделенную до необходимых значений в абонентской ПЛИС.

Для обеспечения работы часов реального времени процессора был выбран кварцевый резонатор РК463-6ДТ-32.768К (размеры 6х3,5х1,2 мм).

В качестве ПЛИС была выбрана микросхема 5576XC1Т, как наименьшая по содержанию логических элементов и габаритам из всех выпускаемых на данный момент отечественной промышленностью.

Для загрузки программного обеспечения процессора и прошивок ПЛИС планировалось использовать ППЗУ с последовательным интерфейсом. Проведенный анализ отечественной элементной базы показал, что на данный момент выпускаются следующие ППЗУ с последовательным интерфейсом и напряжением питания 3,3 В:

- а) 1636PP52У – ППЗУ с последовательным интерфейсом, емкостью 1Мбит, $U_{пит} = 3,3 \text{ В}$;
- б) 1636PP2АУ – ППЗУ с последовательным и параллельным интерфейсами, емкостью 16 Мбит, $U_{пит} = 3,3 \text{ В}$;
- в) 1640PC3У – ППЗУ с последовательным интерфейсом, емкостью 64 кбит, $U_{пит} = 3 - 5,5 \text{ В}$;
- г) 1640PC4У – ППЗУ с последовательным интерфейсом, емкостью 128 кбит, $U_{пит} = 3 - 5,5 \text{ В}$;

Загрузочный файл ПЛИС 5576XC1Т составляет порядка 800 кбит, программное обеспечение процессора может составлять до 256 кбайт, следовательно, для хранения программного обеспечения процессора и прошивок ПЛИС необходимо: $800 + 800 + 256 \cdot 8 = 3648 \text{ кбит}$.

Из выше представленных ППЗУ, для данного объема подходит только микросхема б). Однако данная микросхема имеет также и параллельный интерфейс, а последовательный интерфейс, используемый в ней имеет один двунаправленный вход/выход. Для увеличения скорости считывания информации было принято решение об использовании ИМС 1636PP2У в параллельном режиме.

Для хранения журнала событий необходимо ППЗУ объемом 864 кбит. Для этих целей подходит ИМС 1636PP52У – ППЗУ с последовательным интерфейсом, емкостью 1Мбит.

Как было отмечено в подразделе 3.1.1, в процессоре «MultiClet R1» имеется в наличии встроенный контроллер USB 2.0, который поддерживает только функцию «Device». Для связи с внешней шиной USB необходимо

использование дополнительного приёмопередатчика, подключаемого к процессору по интерфейсу ULPI (UTMI+ Low Pin Interface).

К сожалению, отечественной промышленностью данные приёмопередатчики не выпускаются, поэтому была выбрана ИМС USB3320C-EZK фирмы Microchip Technology Inc, в полной мере отвечающая всем требованиям.

Проанализировав отечественные микроконтроллеры, было определено, что среди них отсутствуют изделия, потребляющие в режиме хранения информации менее 10 мА. То есть ни один из отечественных микроконтроллеров не подходит для использования в качестве контроллера НСД модуля СКЗИ.

Для использования в качестве контроллера НСД был выбран микроконтроллер MSP430F2618TPM фирмы Texas Instruments с объемом ОЗУ 10 кбайт, достаточным для хранения ключа НСД и другой необходимой информации и потреблением в режиме хранения 1 мкА.

В качестве супервизора батарейного питания была выбрана ИМС MAX6362LUT29, также не имеющая отечественных аналогов. ИМС вырабатывает сигнал сброса и переключается на батарейное питание при снижении напряжения 3,3 В до 2,9 В и потребляет ток менее 1 мкА.

Для обеспечения модуля батарейным питанием выбрана литиевая батарея CR 1/2 AA CD фирмы Varta, емкостью 900 мА/час и имеющая конструкцию с выводами для распайки батареи. Единственный российский аналог – литиевая батарея «Космос» не имеет выводов для распайки, не имеет приемки «5» и изготавливается в Китае. Емкость 900 мА/час при потреблении модуля порядка 3 мкА и, учитывая потерю емкости батареи через 10 лет на 65 %, достаточна для хранения информации в модуле в течение (при остаточной емкости – $900/100 \cdot 35 = 315$ мА/час): $315/0,003 = 195000$ часов = 12 лет.

Средний срок службы аппаратуры как правило составляет 10 лет. Следовательно, примененной батареи хватает на весь срок службы аппаратуры.

Для обеспечения оптической развязки применены микросхемы 249ЛП14Р – сдвоенный логический оптрон с временем задержки распространения сигнала 80 нс и корпусе DIP-8.

В качестве фильтров опторазвязки применены фильтры ФПП-Ф-Б-0,5-10000пФ с полосой пропускания 0,3 – 0.6 МГц и затуханием в соответствии с таблицей 8.

Таблица 8 – Затухание фильтра ФПП-Ф-Б-0,5

Частота, МГц	5	10	20	50	100	200	500	1000
Затухание, дБ	10	15	25	30	40	40	55	60

В качестве фильтров питания и фильтров узла микрофонных и телефонных усилителей применены фильтры Ф23Б-6,8мкФ с затуханием в соответствии с таблицей 9.

Таблица 9 – Затухание фильтра Ф23Б

Частота, МГц	0,01	0,1	1,0	10	30	100	300	1000
Затухание, дБ	21	38	57	62	62	62	64	68

В качестве разъема подключения микротелефонной трубки применена вилка СПП-Ф1-7-038-03, представляющая собой разъем СНЦ с фильтрами нижних частот с затуханием в соответствии с таблицей 10.

Таблица 10 – Затухание фильтра СПП-Ф1-7-038-03

Частота, МГц	5	10	20	50	100	200	500	1000
Затухание, дБ	20	35	50	60	60	60	60	60

Для выравнивания фронтов сигналов после опторазвязки и фильтров применены триггеры Шмита 1554ТЛ2 в корпусе SO-14.

Для использования в качестве микрофонного усилителя и стабилизатора тока применена ИМС 544УД18У3 – сдвоенный быстродействующий операционный усилитель с габаритами 7,4x7,8x2,8 мм и напряжением питания 5 В.

Для использования в качестве телефонного усилителя применена ИМС 544УД16У3 – сдвоенный операционный усилитель Rail to Rail с габаритами 7,4x7,8x2,8 мм и напряжением питания 5 В.

Для использования в схеме контроля стабилизатора тока применен компаратор 1481СА1Р с напряжением питания 5 В.

В схеме модуля применены керамические SMD-конденсаторы К10-79, танталовые SMD-конденсаторы К53-68, SMD-резисторы Р1-112.

В стабилизаторе тока применена транзисторная матрица 1НТ251, включающая в себя четыре биполярных п-р-п транзистора.

В схеме контроля напряжения применены транзисторные сборки 2П7241АС9 включающая в себя два полевых транзистора.

Для выбора преобразователя напряжения необходимо подсчитать мощность, потребляемую модулем, которая вычисляется по формуле:

$$P_M = P_{ЦП} + P_{ПЛИС1} + P_{ПЛИС2} + 2P_{ППЗУ1} + P_{ППЗУ2} + P_{УС} + P_{КНСД} + P_{USB}, \quad (1)$$

где $P_{ЦП}$ – мощность, потребляемая центральным процессором;

$P_{\text{ПЛИС1}}$ – мощность, потребляемая абонентской ПЛИС;

$P_{\text{ПЛИС2}}$ – мощность, потребляемая канальной ПЛИС;

$P_{\text{ППЗУ1}}$ – мощность, потребляемая параллельным ППЗУ;

$P_{\text{ППЗУ2}}$ – мощность, потребляемая последовательным ППЗУ;

$P_{\text{УС}}$ – мощность, потребляемая микрофонным и телефонным усилителем;

$P_{\text{КНСД}}$ – мощность, потребляемая контроллером НСД;

P_{USB} – мощность, потребляемая модулем USB.

Подставляя значения потребляемых мощностей, взятых из сопроводительной документации, получаем потребляемую мощность модуля СКЗИ:

$$P_{\text{М}} = 1,08 + 0,066 + 0,066 + 2 * 0,165 + 0,132 + 0,165 + 0,132 + 0,132 = 2,103 \text{ Вт.}$$

С учетом запаса по потребляемой мощности необходимо применение преобразователя, рассчитанного на мощность 3 Вт.

Для наших целей подходит новинка фирмы «Аедон» модуль питания МДМ3-1В05Н, мощностью 3 Вт, гальванически развязанным входом и выходом, малыми габаритами (полный аналог популярных ИМС фирмы «TRACO POWER»), обеспечивающим выработку выходного напряжения 5 В из входного напряжения 18 – 36 В.

Для выработки напряжений 3,3 В и 1,8 В применены микросхемы 1309ЕН3.3Т и 1309ЕН1.8 соответственно. Данные ИМС представляют собой линейные регуляторы напряжения с малым падением напряжения и мощностью до 6 Вт.

3.2.2 Построение принципиальной схемы модуля СКЗИ

Основываясь на анализе технических требований, приведённых в главе 3.1.1, разработанной структурной схеме, приведённой в приложении А и описанной в главе 3.1.2, и на выбранной элементной базе, приведённой в приложении Б и описанной в главе 3.2.1, создана принципиальная схема, приведённая в приложении В.

4 Проверка целостности модуля криптографической защиты информации

4.1 ГОСТ 34.11-2012

После изготовления и отгрузки и до ввода в эксплуатацию аппарата транспортируется до места эксплуатации и может храниться в течение длительного времени (до 5 лет). Возникает вопрос о том, как осуществить безопасность или обнаружить изменения программного обеспечения, хранящегося в модуле, возможных в случае преднамеренных действий злоумышленников. Ответом на него может быть механизм защиты, основанный на сравнение текущего состояния системы с эталонным состоянием системы до ввода в эксплуатацию.

Данный механизм будет представлять из себя программное средство, которое выполняет подсчет контрольного значения, будь то хэш-сумма или имитовставка критичных информационных ресурсов системы таких как исполняемые файлы (программы), конфигурационные файлы, содержащие условно постоянную информацию, не меняющуюся на длительном промежутке функционирования системы.

Хэш-функция – функция, отображающая строки бит в строки бит фиксированной длины и удовлетворяет следующим свойствам:

- по данному значению хэш-функции практически невозможно вычислить исходные данные, отображаемые в это значение;
- для заданных исходных данных практически невозможно вычислить другие исходные данные, отображаемые в то же значение функции;
- практически невозможно вычислить какую-либо пару исходных данных, отображаемых в одно и то же значение. [4]

Хэш-сумма – это фиксированная строка бит, получаемая на выходе хэш-функции, на вход которой передаётся последовательность байт.

Криптографическая хэш-функция – это хэш-функция, которая является криптостойкой, иначе говоря удовлетворяет ряду требований, присущих для

криптографических приложений.

На основе подсчета хэш-суммы используется механизм контроля целостности. Данный механизм может обеспечивать целостность файлов модуля путем подсчета эталонной контрольной суммы данных представленных источников и последующего сравнения эталонных контрольных сумм с контрольными суммами, подсчитанными от данных в процессе эксплуатации системы.

Существует множество алгоритмов для вычисления контрольных сумм. Среди них множество зарубежных, однако существует и отечественный алгоритм вычисления контрольной суммы. Далее приведен список наиболее известных алгоритмов хэш-функций:

- MD5;
- SHA1;
- SHA256;
- SHA384;
- SHA512;
- ГОСТ Р 34.11-91 (длина 256 бит);
- ГОСТ Р 34.11-2012 (длина 256 и 512 бит).

Из наиболее известных алгоритмов вычисления контрольных сумм был выбран для анализа современный ГОСТ Р 34.11-2012. Данный ГОСТ был выбран в силу того, что он обеспечивает повышенную криптографическую стойкость и является действующим российским криптографическим стандартом, который определяет процедуру вычисления хэш-функции.

Стандарт ГОСТ 34.11-2012 пришел на смену ГОСТ 34.11-94, который к настоящему времени уже считается потенциально уязвимым (хотя до 2018 года ГОСТ 1994 года выпуска все же применять не возбраняется). Отечественные стандарты хеширования обязательны к применению в продуктах, которые будут применяться в ответственных и критических сферах и для которых обязательно прохождение сертификации в уполномоченных органах (ФСТЭК,

ФСБ и подобных). ГОСТ 34.11-2012 был разработан Центром защиты информации и специальной связи ФСБ России с участием открытого акционерного общества «Информационные технологии и коммуникационные системы» (ИнфоТеКС). В основу стандарта 2012 года была положена функция хеширования под названием «Стрибог». [5]

ГОСТ 34.11-2012 определяет алгоритм и процедуру вычисления хэш-функции для любой последовательности двоичных символов, которые применяются в криптографических методах обработки и защиты информации, в том числе для реализации процедур обеспечения целостности, аутентичности электронной цифровой подписи при передаче, обработке и хранении информации в автоматизированных системах. [4]

Хеш-функция «Стрибог» может иметь две реализации с результирующим значением длиной 256 или 512 бит. На вход функции подается сообщение, для которого необходимо вычислить хеш-сумму. Если длина сообщения больше 512 бит (или 64 байт), то оно делится на блоки по 512 бит, а оставшийся кусочек дополняется нулями с одной единичкой до 512 бит (или до 64 байт). Если длина сообщения меньше 512 бит, то оно сразу дополняется нулями с единичкой до полных 512 бит.

Основу хеш-функции «Стрибог» составляет функция сжатия (g-функция), построенная на блочном шифре, построенном с помощью конструкции Миягучи – Пренеля, признанной одной из наиболее стойких.

В целом хеширование производится в три этапа. Первый этап – инициализация всех нужных параметров, второй этап представляет собой так называемую итерационную конструкцию Меркла – Дамгорда с процедурой МД-усиления, третий этап – завершающее преобразование: функция сжатия применяется к сумме всех блоков сообщения и дополнительно хешируется длина сообщения и его контрольная сумма. [5]

Однако, данный алгоритм не является всеохватывающим для реализации его при проверке целостности в разрабатываемом модуле СКЗИ, а именно у

него нет возможности реализовать защиту от НСД к блоку, ввиду отсутствия у него определённого параметра для защиты от данной угрозы (ключа НСД). Вследствие этого принято решение по реализации контроля целостности критически важной информации согласно ГОСТ 34.12-2015 в режиме имитовставки по ГОСТ 34.13-2015.

4.2 ГОСТ 34.12-2015

ГОСТ определяет алгоритмы базовых блочных шифров, которые применяются в криптографических методах обработки и защиты информации, в том числе для обеспечения конфиденциальности, аутентичности и целостности информации при ее передаче, обработке и хранении в автоматизированных системах.

Алгоритм зашифрования – алгоритм, реализующий зашифрование, т.е. преобразующий открытый текст в шифртекст.

Базовый блочный шифр – блочный шифр, реализующий при каждом фиксированном значении ключа одно обратимое отображение множества блоков открытого текста фиксированной длины в блоки шифртекста такой же длины.

Блочный шифр – шифр из класса симметричных криптографических методов, в котором алгоритм зашифрования применяется к блокам открытого текста для получения блоков шифртекста. [6]

В данном государственном стандарте определены алгоритмы криптографического преобразования предназначенные для аппаратной или программной реализации, удовлетворяющие современным криптографическим требованиям.

Стандарт применяется при создании, эксплуатации и модернизации автоматизированных систем различного назначения.

В ГОСТ дано описание двух видов алгоритма – это блочный шифр с длиной блока 128 бит и ключом 256 бит, так называемый «Кузнечик», и «Магма», новое название старого ГОСТ 28147-89, с фиксированным набором

перестановок.

За основу в алгоритме «Кузнечик» взята SP-сеть (подстановочно-перестановочная сеть). В алгоритм заложено несколько итераций со стадиями подстановок и перестановок. Каждый раунд включает в себя линейное и нелинейное преобразование и операцию наложения так называемого итерационного ключа. Всего раундов девять и один последний неполный раунд, в котором выполняется только наложение десятого итерационного ключа.

Схематичный алгоритм работы ГОСТ 34.12-2015 «Кузнечик» в прямом и обратном режимах приведён на рисунке 5, где K_i – раундовый ключ, L преобразование – линейное преобразование, S преобразование – нелинейное биективное преобразование.

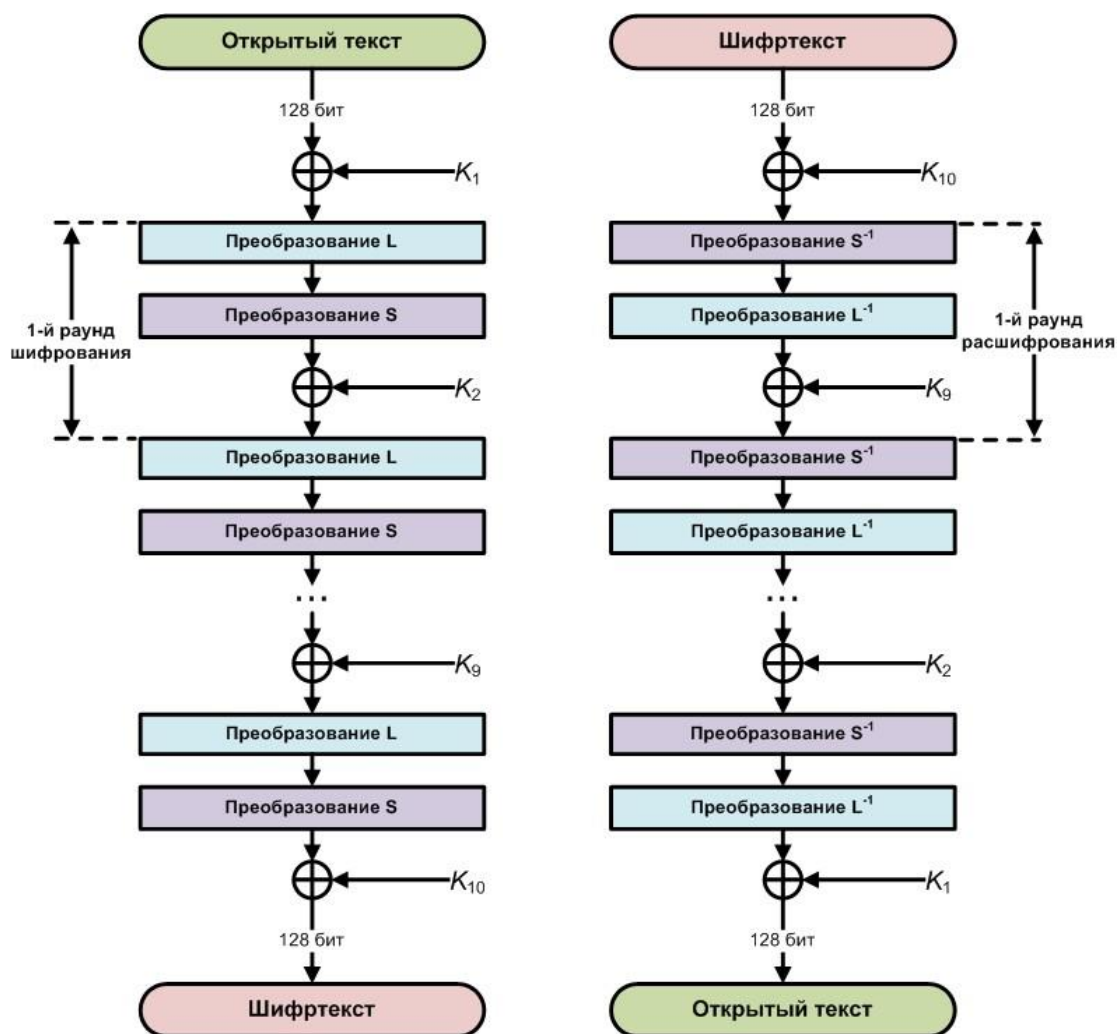


Рисунок 5 – Алгоритм преобразования «Кузнечик»

Раундовые ключи получаются путем преобразований основного ключа

длинной 256 бит. Сначала основной ключ делится пополам, так получается первая пара раундовых ключей. Для генерации каждой последующей пары раундовых ключей применяется 8 итераций сети Фейстеля, в каждой итерации используется константа, которая равна значению номера итерации с применённым к ней линейным преобразованием. Схему получения раундового ключа представлена на рисунке 6, где C – константа, L преобразование – линейное преобразование, S преобразование – нелинейное биективное преобразование.

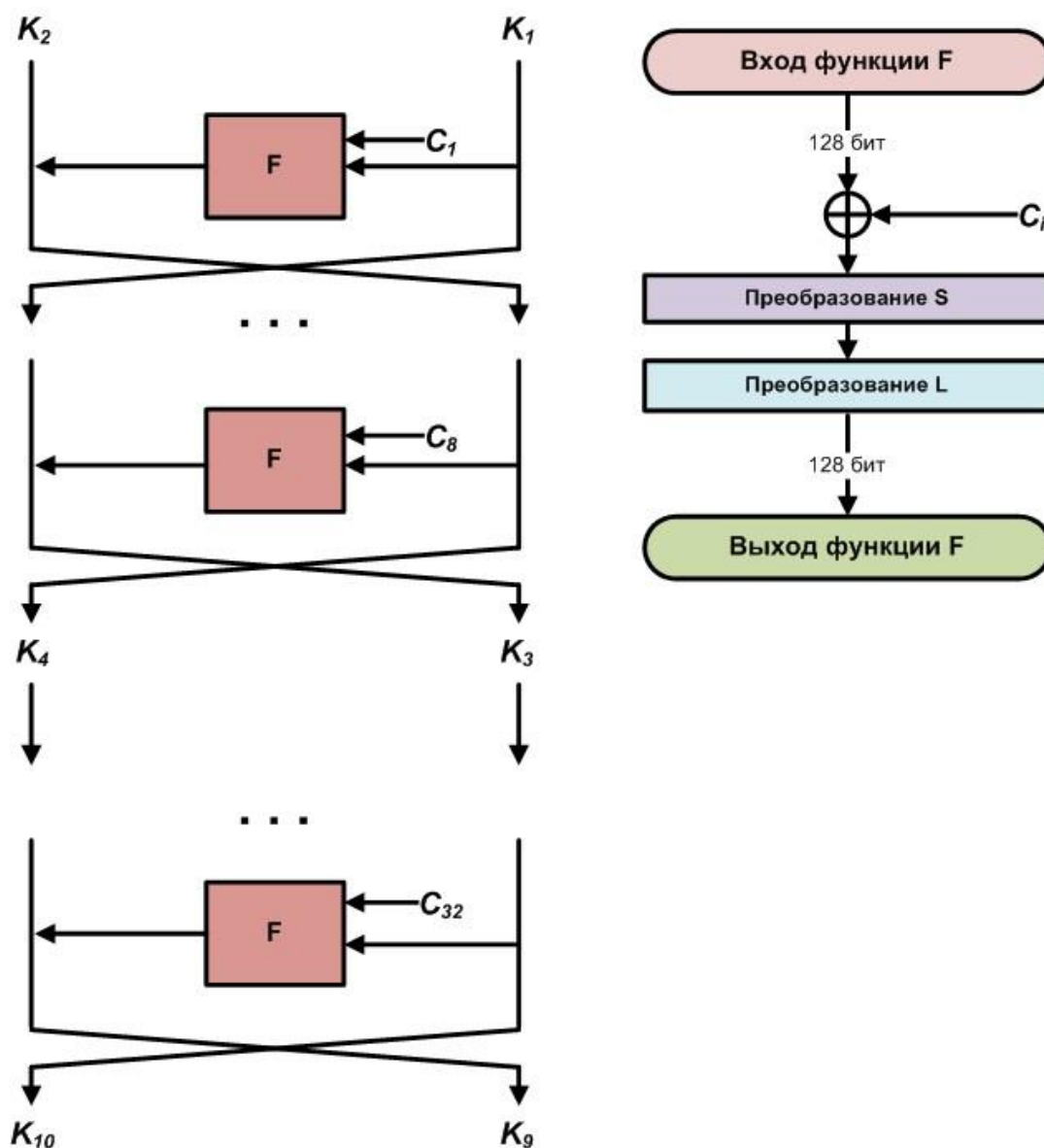


Рисунок 6 – Получение раундового ключа

Шифр «Магма» также известный как ГОСТ 28147-89, достаточно простой

и с лёгкостью может быть реализован на базе любого микроконтроллера. Данная часть ГОСТ 34.12-2015 имеет разрядность блока 64 бита и разрядность ключа равной 256 бит.

Алгоритм работает по следующему принципу. На вход подаётся блок открытого текста равный 64 бит и ключ, после чего осуществляется преобразование, называемое шифром. 64-битный блок разделяется на две равные части, над которыми производятся операции раунда, в которую входят: сложение с ключом раунда, подстановка, циклический сдвиг на 11, сложение с половиной блока. После всего этого раунд заканчивается перестановкой полублоков местами. Всего таких итераций 32. На выходе получается блок шифротекста, той же разрядности. Схематический алгоритм работы «Магмы» представлен на рисунке 7, где N1 и N2 – левая и правая часть 64-битного блока данных, K_x – раундовый ключ.

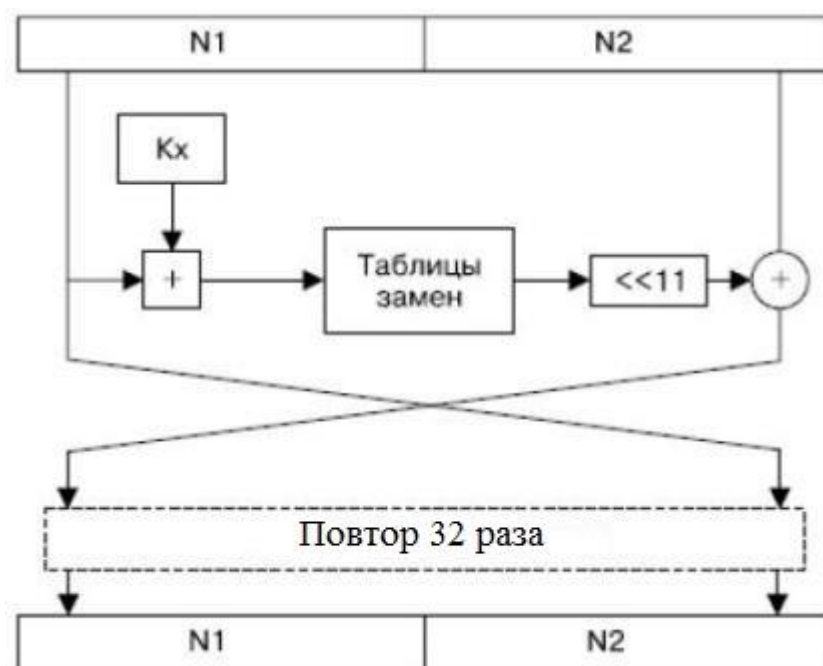


Рисунок 7 – Алгоритм преобразования «Магма»

Раундовых ключей в алгоритме – 32. Каждый имеет разрядность 32 бита. Эти ключи формируются из основного ключа шифрования при помощи алгоритма развёртывания ключа. Исходный ключ содержит 32 байта. 32 байта – это 8 раундовых ключей, каждый по четыре байта; на 32 раунда – ключи

копируются, с той лишь разницей, что последние восемь используются в обратном порядке. Схема формирования ключей представлена на рисунке 8.

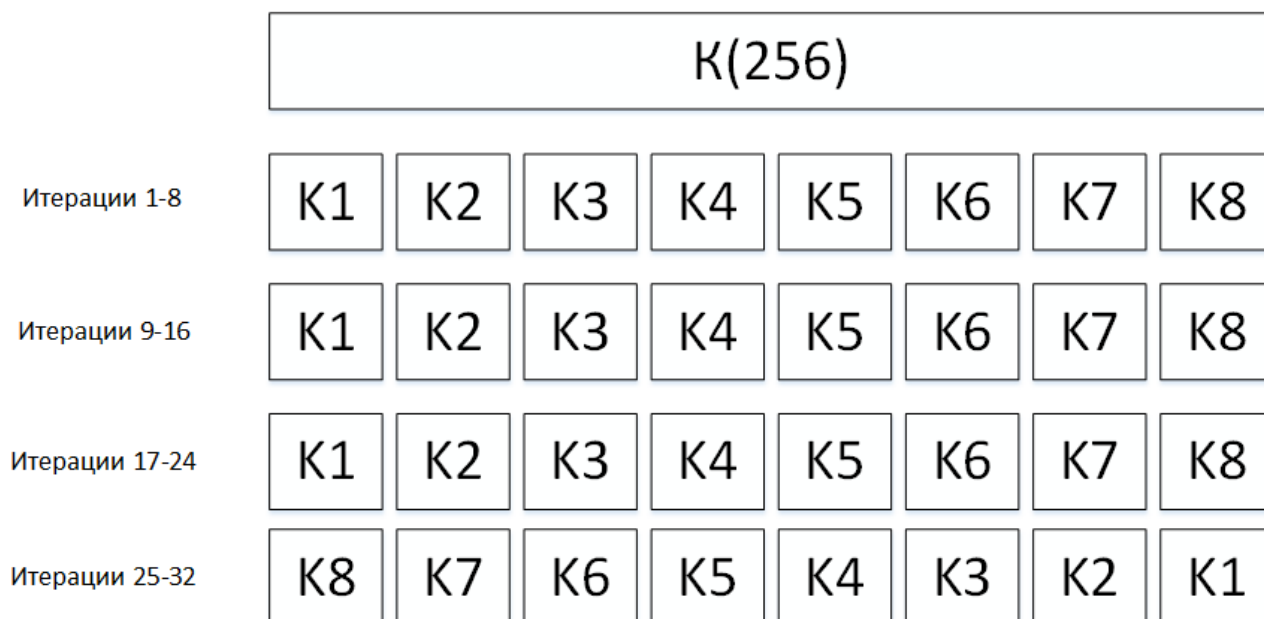


Рисунок 8 – Схема раундовых ключей «Магма»

4.3 ГОСТ 34.13-2015 в режиме имитовставки

Возникает вопрос о том, как обеспечить безопасность информации, хранящейся в системе, от изменений в случае умышленных действий пользователей. Ответом на него может быть механизм защиты, основанный на сравнение текущего состояния системы с эталонным состоянием системы до ввода в эксплуатацию.

Данный механизм будет представлять из себя программное средство, которое выполняет подсчет контрольных имитовставок всех критичных информационных ресурсов системы таких как исполняемые файлы (программы), разделяемые библиотеки, конфигурационные файлы, не меняющуюся на длительном промежутке функционирования системы.

Для решения данной задачи подходит ГОСТ 34.13-2015 в режиме имитовставки для ГОСТ 34.12-2015.

Настоящий стандарт содержит описание режимов работы блочных шифров. Данные режимы работы блочных шифров определяют правила

криптографического преобразования данных и выработки имитовставки для сообщений произвольного размера.

Имитовставка – строка бит фиксированной длины, полученная применением симметричного криптографического метода к сообщению, добавляемая к сообщению для обеспечения его целостности и аутентификации источника данных. [7]

На первом этапе работы алгоритма необходимо выработать вспомогательные ключи, которые вычисляются с использованием основного ключа, отличного от ключа, на котором непосредственно производится шифрование данных.

Для выработки вспомогательных ключей необходимо сначала зашифровать блок, состоящий из нулей, необходимой размерности, чаще всего это 64 и 128 бит, в стандарте это число называется R .

Затем вырабатывается первый вспомогательный ключ K_1 , который равен побитовому сдвигу влево на единицу числа R , если первый бит слева полученного на предыдущем этапе числа R равен единице, в противном случае, если первый бит равен 1, то ключ так же равен циклически сдвинутому на 1 числу R , но ещё сложенному по модулю 2 с числом B , которое равно для 64 битного блока $0^{59}||11011$, а для 128 битного блока $0^{120}||10000111$, где степень обозначает количество нулей, а $||$ - операция конкатенации.

После этого вычисляется вспомогательный ключ K_2 , который равен ключу K_1 циклически сдвинутому на единицу влево, если первый бит K_1 равен нулю, в противном случае ключ также равен K_1 сдвинутому на 1 влево, но ещё сложенному по модулю 2 с числом B , которое равно для 64 битного блока $0^{59}||11011$, а для 128 битного блока $0^{120}||10000111$, где степень обозначает количество нулей, а $||$ - операция конкатенации.

Процесс выработки имитовставки заключается в следующем. Исходное сообщение разбивается на равные блоки, размерность которых равна размерности блоков алгоритма шифрования, причём если последнему блоку не хватает бит,

то он заполняется до нужного размера одной единицей и нулями. На вход алгоритму шифрования подается результат сложения по модулю очередного блока текста и результата зашифрования на предыдущем шаге. На первом шаге блок просто шифруется без сложения. При обработке последнего блока на вход базовому алгоритму блочного шифрования подается результат сложения по модулю последнего блока, результата зашифрования на предыдущем шаге и одного из вспомогательных ключей. Вспомогательный ключ выбирается в зависимости от того, является ли последний блок исходного сообщения полным, в этом случае используется K_1 , или нет, в этом случае используется K_2 . Значение имитовставки это результат применения процедуры усечения к выходу алгоритма шифрования при обработке последнего блока. Алгоритм работы имитовставки приведён на рисунке 9.

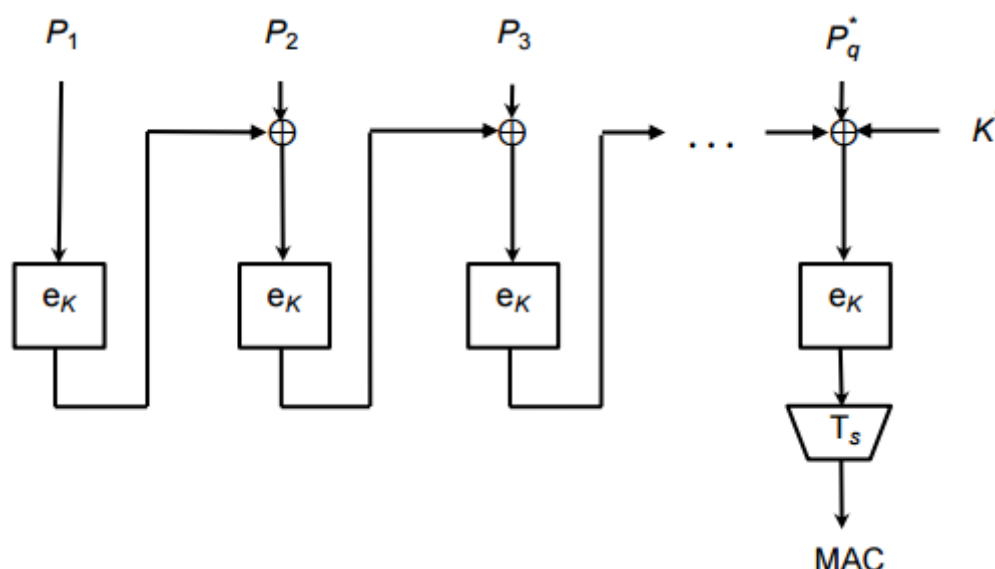


Рисунок 9 – Алгоритм работы ГОСТ 34.13-2015 в режиме имитовставки

4.4 Реализация ГОСТ 34.12-2015 в режиме имитовставки ГОСТ 34.13-2015

Для достижения цели проверки целостности критически важных программ модуля криптографической защиты реализована программа проверки целостности вышеописанного алгоритма ГОСТ 34.13-2015 в режиме

имитовставки на базе ГОСТ 34.12-2015 «Магма». Реализация ГОСТ произведена на языке С.

Язык Си — компилируемый язык. Это означает, что текст программы, написанный на Си с помощью любого текстового редактора, но чаще всего используются встроенные в интегрированные среды разработки, для получения выполняемой программы обрабатывается компилятором, причем компиляция выполняется двумя повторениями. При первом проходе компиляции обрабатываются директивы программы, начинающиеся со знака #, при втором — транслируется текст программы и создается непосредственно машинный код. Для получения исполняемого модуля к нему подсоединяются соответствующие библиотеки.

Программа была разработана согласно алгоритму, приведённому в приложении Г, с помощью компилятора Geany IDE. Код программы приведён в приложении Д. Далее описаны основные функции программы.

Функция getKey имеет размерность void и получает из заданной ячейки памяти ключ для шифрования с алгоритмом ГОСТ 34.12-2015. Он реализован путём получения из шины данных за 1 итерацию 16 бит данных и постепенного заполнения всех 256 бит. Код функции представлен на рисунке 10.

```
void getKey ()
{
    block32 midleKey;
    int i,j;
    block32 *blockOfKey;

    for (i=0; i<=7; i++)
    {
        for (j=0; j<=1; j++)
        {
            blockOfKey = (&keyNSD + ( j * 16 ) + ( i * 32 ));
            midleKey = *blockOfKey;
            midleKey <<= 16;
        }
        keyNSD[i] = midleKey;
    }
}
```

Рисунок 10 – Код функции getKey

Функция `magma` имеет размерность 64 бит, что соответствует 2 без знаковым длинному целочисленному блоку. Функция принимает на вход также блоки размерностью 32 бит и производит на нём операции описанные в ГОСТ 34.12-2015 «Магма» на ключе НСД, который задаётся вызовом функции `getKey`. На выходе функция возвращает зашифрованный блок, подаваемый на вход. Для реализации функции необходимо значение подстановок, определённые в ГОСТ и приведённые на рисунке 11.

```
const byte tau[8][16] =
{
    0xC, 0x4, 0x6, 0x2, 0xA, 0x5, 0xB, 0x9, 0xE, 0x8, 0xD, 0x7,
    0x0, 0x3, 0xF, 0x1,
    0x6, 0x8, 0x2, 0x3, 0x9, 0xA, 0x5, 0xC, 0x1, 0xE, 0x4, 0x7,
    0xB, 0xD, 0x0, 0xF,
    0xB, 0x3, 0x5, 0x8, 0x2, 0xF, 0xA, 0xD, 0xE, 0x1, 0x7, 0x4,
    0xC, 0x9, 0x6, 0x0,
    0xD, 0x8, 0x2, 0x1, 0xD, 0x4, 0xF, 0x6, 0x7, 0x0, 0xA, 0x5,
    0x3, 0xE, 0x9, 0xB,
    0x7, 0xF, 0x5, 0xA, 0x8, 0x1, 0x6, 0xD, 0x0, 0x9, 0x3, 0xE,
    0xB, 0x4, 0x2, 0xC,
    0x5, 0xD, 0xF, 0x6, 0x9, 0x2, 0xC, 0xA, 0xB, 0x7, 0x8, 0x1,
    0x4, 0x3, 0xE, 0x0,
    0x8, 0xE, 0x2, 0x5, 0x6, 0x9, 0x1, 0xC, 0xF, 0x4, 0xB, 0x0,
    0xD, 0xA, 0x3, 0x7,
    0x1, 0x7, 0xE, 0xD, 0x0, 0x5, 0x8, 0x3, 0x4, 0xF, 0xA, 0x6,
    0x9, 0xC, 0xB, 0x2
};
```

Рисунок 11 – Значение перестановок

Функция содержит 32 итерации для двух поочерёдно чередующихся частей входного блока, начиная с левой части производятся преобразования, в которые входят сложение с раундовым ключом, затем идёт перестановка каждого байта согласно заданной таблице подстановок, представленной на рисунке 11. Реализация перестановки представлена на рисунке 12.

```

for (int q = 3; q >= 0; q--)
{
    character = *((byte *)&RES + q);
    first_byte = (character & 0xF0) >> 4;
    second_byte = (character & 0x0F);
    first_byte = tau[n][first_byte];
    n--;
    second_byte = tau[n][second_byte];
    n--;
    character = (first_byte << 4) | second_byte;
    *((byte *)&RES + q) = character;
}

```

Рисунок 12 – Фрагмента кода реализации перестановок

После всех произведённых операций происходит циклический сдвиг на 11 бит влево, после чего полученное число складывается по модулю 2 со второй частью входного блока, и в конце блоки меняются.

Функция `getImitoForUART` целочисленная. Она принимает на вход номер порта UART по которому будут передаваться и приниматься данные для расчёта имитовставки по алгоритму ГОСТ 34.13-2015, и адрес эталонного значения, с которым и будет сравниваться рассчитанное значение. На выходе функции будет 1, если результат сравнения эталонного значения и рассчитанной имитовставки не совпали, в противном случае функция вернёт 0.

Для подсчёта имитовставки необходимо рассчитать вспомогательные ключи, алгоритм которого описан в разделе ГОСТ 34.13-2015, для получения вспомогательных ключей необходимо преобразовать последовательность нулей с помощью вышеописанной функции `gamma`.

Далее происходит инициализация необходимого порта UART со всеми необходимыми параметрами, такими как частота работы, режим. После на инициализированный порт приходят данные по 8 бит информации, которые постепенно заполняют 2 блока размерностью 32 бит, над которым и происходит преобразование зашифрования. Первая итерация заключается только в зашифровании блока, последующие итерации происходят с предварительным сложением сообщения по модулю 2 с результатом предшествующей итерации,

причём если один из блоков или оба остаются незаполненными, они автоматически дополняется справа одной 1 и остальным количеством нулей.

После вычисления имитовставки, она сравнивается с эталонным значением, адрес которого подаётся на вход функции `getImitoForUART` и возвращает 1, что соответствует ошибке либо ноль, что соответствует корректной работе блока. Фрагмент кода реализующий данную проверку приведён на рисунке 13.

```
if ( ( *imito != *addressForImito ) || ( *((byte *)&imito + 4 ) != *((byte *)&addressForImito + 4)))  
{  
    return 1;  
}  
else  
{  
    return 0;  
}
```

Рисунок 13 – Фрагмент кода реализующий проверку имитовставки эталонному значению

Функция `getImitoForSPI` целочисленная. Она принимает на вход номер порта SPI по которому будут передаваться и приниматься данные для расчёта имитовставки по алгоритму ГОСТ 34.13-2015, и адрес эталонного значения, с которым и будет сравниваться рассчитанное значение. На выходе функции будет 1, если результат сравнения эталонного значения и рассчитанной имитовставки не совпали, в противном случае функция вернёт 0.

Для подсчёта имитовставки необходимо также, как и в функции `getImitoForUART`, рассчитать вспомогательные ключи, алгоритм которого описан в разделе ГОСТ 34.13-2015, для получения вспомогательных ключей необходимо преобразовать последовательность нулей с помощью вышеописанной функции `магма`.

Далее происходит инициализация необходимого порта SPI со всеми необходимыми параметрами, такими как длина передаваемого сообщения за 1 такт, режим, разрешение работы порта. После на инициализированный порт приходят данные по 32 бита информации (снвное отличие порта SPI от UART, у UART размерность составляет 8 бит), которые постепенно заполняют 2 блока

размерностью 32 бит каждый, над которым и происходит преобразование зашифрования. Первая итерация заключается только в зашифровании блока, последующие итерации происходят с предварительным сложением сообщения по модулю 2 с результатом предшествующей итерации, причём если один из блоков не заполнен, он автоматически дополняется справа одной 1 и нулями до необходимого количества.

После вычисления имитовставки, она сравнивается с эталонным значением, адрес которого подаётся на вход функции `getImitoForUART` и возвращает 1, что соответствует ошибке либо ноль, что соответствует корректной работе блока. Фрагмент кода реализующий данную проверку приведён на рисунке 13.

Функция `getImitoForDATA` целочисленная. Она принимает на вход адреса данных начала и конца, по которым необходимо произвести проверку целостности данных по алгоритму ГОСТ 34.13-2015, адрес эталонного значения, с которым и будет сравниваться рассчитанное значение, и номера шины данных по которым будет осуществляться чтение данных. На выходе функции будет 1, если результат сравнения эталонного значения и рассчитанной имитовставки не совпали, в противном случае функция вернёт 0.

Для подсчёта имитовставки необходимо также, как и в функции `getImitoForUART`, рассчитать вспомогательные ключи, алгоритм которого описан в разделе ГОСТ 34.13-2015, для получения вспомогательных ключей необходимо преобразовать последовательность нулей с помощью вышеописанной функции `магма`.

Далее происходит вычисление имитовставки в заданных значениях, для этого заполняются рабочие блоки по 32 бита информации, над которым и происходит преобразование зашифрования. Первая итерация заключается только в зашифровании блока, последующие итерации происходят с предварительным сложением сообщения по модулю 2 с результатом предшествующей итерации, причём если один из блоков не заполнен, он

автоматически дополняется справа одной 1 и нулями до необходимого количества.

После вычисления имитовставки, она сравнивается с эталонным значением, адрес которого подаётся на вход функции `getImitoForUART` и возвращает 1, что соответствует ошибке либо ноль, что соответствует корректной работе блока. Фрагмент кода реализующий данную проверку приведён на рисунке 13.

Основная функция содержит проверку на ошибки, всех критических данных и в случае несоответствия рассчитанных имитовставок с эталонными значениями на лицевую панель и системный журнал посылается код ошибки «47».

Работоспособность программы проверена на нулевых значениях входного сообщения размерностью 256 нулей и ключе, состоящем из восьми цифр от 1 до 8 в шестнадцатеричном представлении: `c0fcba4d`. Значения имитовставок были записаны по адресу `0x00040000-0x00040020`. Запуск с этими же параметрами и с указанными параметрами адреса эталонного значения выполнялся без ошибок, о чём соответствует результат отладочной переменной `status`, результат которой приведён на рисунке 14.

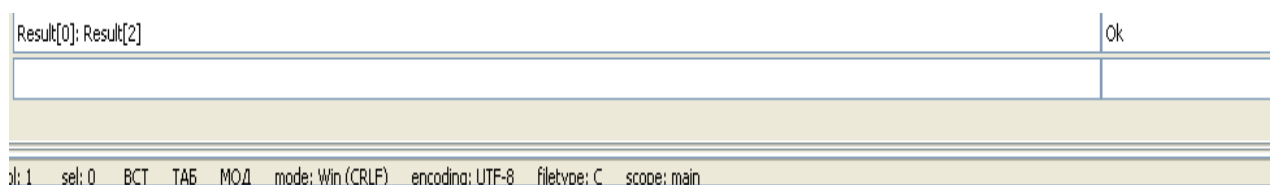


Рисунок 14 – Результат работы программы при успешной проверке

При изменении первоначального ключа в последнем шестнадцатеричном числе на единицу, что отличалось от эталонного значения и, вследствие чего, программа выдала код ошибки «47» в заданный порт и отладочная переменная изменила своё значение, результат которой представлен на рисунке 15.

Result[0]: Result[2]	Err
col: 1 sel: 0 BCT TAB MOD mode: Win (CRLF) encoding: UTF-8 filetype: C scope: main	

Рисунок 15 – Результат работы программы при возникновении ошибки

ЗАКЛЮЧЕНИЕ

В рамках данной выпускной квалификационной работы были рассмотрены существующие стандарты по обеспечению целостности, которые можно применить в специализированной аппаратуре, изучены функция хэширования ГОСТ Р 34.11-2012, алгоритмы шифрования ГОСТ 34.12-2015 и принцип работы данного алгоритма в режиме имитовставки по ГОСТ 34.13-2015, изучены возможности процессора MultiClet R1 и его отладчика на базе среды разработки Geany IDE.

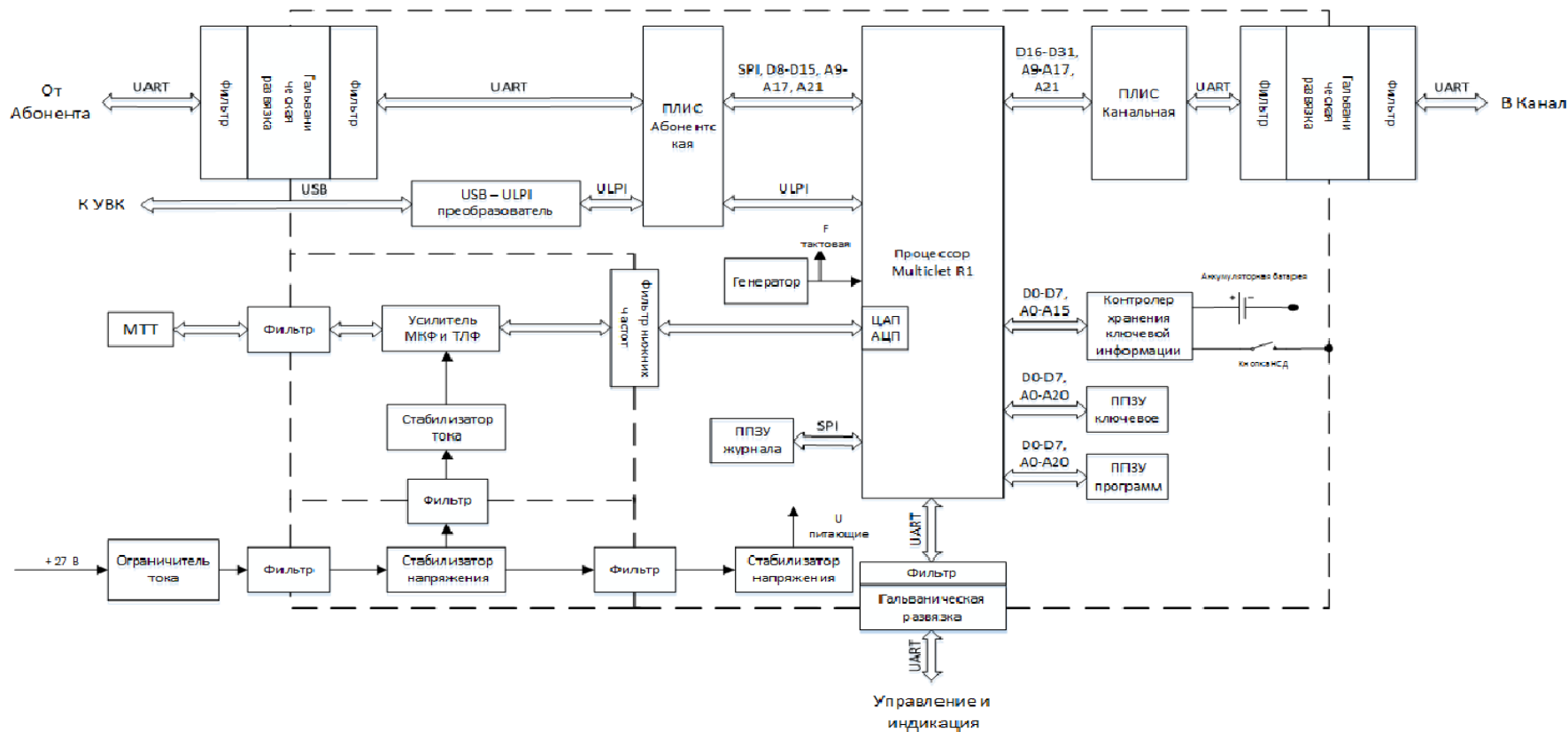
В результате выполнения работы были разработаны структурная и принципиальная схемы модуля криптографической защиты информации на базе процессора MultiClet R1 и реализовано программное средство для контроля целостности программного обеспечения, хранящегося в ППЗУ программ, контроллере хранения ключевой информации, абонентском модуле и контроллере управления и индикации, с использованием алгоритма ГОСТ 34.12-2015 «Магма» в режиме имитовставки по ГОСТ 34.13-2015. Проверка целостности ПО производится на ключе НСД, отличным от основной ключевой информации модуля.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 ГОСТ Р 50922-2006. Защита информации. Основные термины и определения. – Введ. 01.02.2008. – М. : Федеральное агентство по техническому регулированию и метрологии : Изд-во стандартов, 2008, 12 с.
- 2 ГОСТ 28147-89 «Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования» [Электронный ресурс]. – Режим доступа: <http://gostrf.com/normadata/1/4294826/4294826631.pdf>.
- 3 Официальный сайт компании MultiClet. Введение в мультиклеточные процессоры. [Электронный ресурс]. – Режим доступа: <http://multiclet.com/index.php/ru/support/technical-documents/112-introduction-in-multicellular-processors>.
- 4 ГОСТ Р 34.11-2012. Информационная технология. Криптографическая защита информации. Функция хэширования. – Введ. 01.01.2013. – М. : Федеральное агентство по техническому регулированию и метрологии : Изд-во стандартов, 2013 – V, 24 с.
- 5 Добротун Е. Импортзамещенное шифрование глазами программиста. Хэшируем по ГОСТ Р 34.11-2012// Хакер [Электронный ресурс]. – Электрон. журн. – 2016. – 20 июля – Режим доступа: <https://xakep.ru/2016/07/20/hash-gost-34-11-2012/>
- 6 ГОСТ Р 34.12–2015 «Информационная технология. Криптографическая защита информации. Блочные шифры» [Электронный ресурс]. - Режим доступа: http://www.tc26.ru/standard/gost/GOST_R_3412-2015.pdf.
- 7 ГОСТ Р 34.13–2015 «Информационная технология. Криптографическая защита информации. Режимы работы блочных шифров» [Электронный ресурс]. – Режим доступа: http://www.tc26.ru/standard/gost/GOST_R_3413-2015.pdf.

ПРИЛОЖЕНИЕ А
(обязательное)
Структурная схема блока СКЗИ.

ЛУ.100503.С3.2.ДП.12ПК.119.01ПЗ



				ЛУ.100503.С3.2.ДП.12ПК.119.01ПЗ		
Изм. Лист	№ док.м.	Подп.	Дата	<div>Модуль СКЗИ</div> <div>Структурная схема</div>	Лит.	Масштаб
Разработ.	Улитин В.В.					1:1
Проб.	Варламов В.В.			<div>Лист</div> <div>Листов 1</div>		
Т.контр.						
И.контр.	Иванов А.П.			<div>Капуравал</div> <div>Формат А3</div>		
Утв.	Варламов В.В.					

ПРИЛОЖЕНИЕ Б

(обязательное)

Перечень элементов, предназначенный для создания модуля СКЗИ.

Поз. обозначение		Наименование			Кол.	Примечание
C72		Конденсатор K10-79-10 В-1 мкФ+80-20%-H90				
		АЖЯР.673511.004 ТУ			1	
C73		Конденсатор K53-68 "В"-25 В-10 мкФ+-20%				
		АЖЯР.673546.007 ТУ			1	
C74		Конденсатор K53-68 "В"-25 В-10 мкФ+-20%				
		АЖЯР.673546.007 ТУ			1	
C75		Конденсатор K10-79-10 В-1 мкФ+80-20%-H90				
		АЖЯР.673511.004 ТУ			1	
C76		Конденсатор K53-68 "В"-25 В-10 мкФ+-20%				
		АЖЯР.673546.007 ТУ			1	
C77		Конденсатор K10-79-10 В-1 мкФ+80-20%-H90				
		АЖЯР.673511.004 ТУ			1	
C78...		Конденсатор K10-79-10 В-0,1 мкФ+80-20%-H90				
C98		АЖЯР.673511.004 ТУ			21	
C99,		Конденсатор K10-79-10 В-10 мкФ+80-20%-H90				
C100		АЖЯР.673511.004 ТУ			2	
C101		Конденсатор K53-68 "Е"-50 В-15 мкФ+-20%				
		АЖЯР.673546.007 ТУ			1	
C102,		Конденсатор K10-79-10 В-1 мкФ+80-20%-H90				
C103		АЖЯР.673511.004 ТУ			2	
C104		Конденсатор K10-79-50 В-4.7 мкФ+80-20%-H90				
		АЖЯР.673511.004 ТУ			1	
C105,		Конденсатор K10-79-10 В-1 мкФ+80-20%-H90				
C106		АЖЯР.673511.004 ТУ			2	
C107		Конденсатор K10-79-50 В-4.7 мкФ+80-20%-H90				
		АЖЯР.673511.004 ТУ			1	
C108...		Конденсатор K10-79-10 В-0,1 мкФ+80-20%-H90				
C114		АЖЯР.673511.004 ТУ			6	
C115		Конденсатор K53-68 "D"-10 В-100 мкФ+-20%				
		АЖЯР.673546.007 ТУ			1	

Инв. № подл.					ПГУ.100503.С3.2.ДП.12ПК119.02.ПЗ	Лист 2
	Изм	Лист	№ докум.	Подп.		

Копировал

Формат А4

					Поз. обозначение	Наименование	Кол.	Примечание
					C116	Конденсатор K10-79-10 В-0,1 мкФ+80-20%-H90		
						АЖЯР.673511.004 ТУ	1	
					C117,	Конденсатор K10-79-10 В-1 мкФ+80-20%-H90		
					C118	АЖЯР.673511.004 ТУ	2	
					C119...	Конденсатор K10-79-10 В-2.2 мкФ+80-20%-H90		
					C121	АЖЯР.673511.004 ТУ	3	
					C122...	Конденсатор K10-79-100 В-47 пФ+-5%-МПО		
					C124	АЖЯР.673511.004 ТУ	3	
					C125	Конденсатор K53-68 "D"-10 В-100 мкФ+-20%		
						АЖЯР.673546.007 ТУ	1	
					C126	Конденсатор K10-79-10 В-0,1 мкФ+80-20%-H90		
						АЖЯР.673511.004 ТУ	1	
					C127	Конденсатор K10-79-10 В-1 мкФ+80-20%-H90		
						АЖЯР.673511.004 ТУ	1	
					C128,	Конденсатор K10-79-10 В-0,1 мкФ+80-20%-H90		
					C129	АЖЯР.673511.004 ТУ	2	
					C130	Конденсатор K10-79-10 В-2.2 мкФ+80-20%-H90		
						АЖЯР.673511.004 ТУ	1	
					C131,	Конденсатор K10-79-100 В-47 пФ+-5%-МПО		
					C132	АЖЯР.673511.004 ТУ	2	
					C133	Конденсатор K10-79-10 В-0,1 мкФ+80-20%-H90		
						АЖЯР.673511.004 ТУ	1	
					C134	Конденсатор K53-68 "D"-10 В-100 мкФ+-20%		
						АЖЯР.673546.007 ТУ	1	
					C135	Конденсатор K10-79-10 В-0,1 мкФ+80-20%-H90		
						АЖЯР.673511.004 ТУ	1	
								</

Поз. обозначение	Наименование	Кол.	Примечание
D1	Микросхема МСр042R100102-LQ256	1	
D2	Микросхема 1636PP52У АЕНВ.431210.158 ТУ	2	
D3	Микросхема 1636PP2АУ АЕЯР.431210.647 ТУ	1	
D4	Микросхема USB3320C-EZK	1	Microchip
			Technology Inc.
D5	Микросхема 1636PP2АУ АЕЯР.431210.647 ТУ	1	
D6	Микросхема 5576XC1Т АЕЯР.431260.478 ТУ	1	
D7	Микросхема МАХ6362LUT29	1	Maxim/Dallas
D8	Микросхема MSP430F2618TPM	1	Texas Instruments
D9	Микросхема 5576XC1Т АЕЯР.431260.478 ТУ	1	
D10	Микросхема 1309ЕН3.3Т АЕЯР.431420.668 ТУ	1	
D11	Микросхема 1309ЕН1.8Т АЕЯР.431420.668 ТУ	1	
D12...	Микросхема 249ЛП14Р АДКБ.431150.214ТУ		
D17		6	
D18	Микросхема 1554ТЛ2 ТБМ АЕЯР.431200.182-04 ТУ	1	
D19	Микросхема 544УД16У3 АЕЯР.431130.510 ТУ	1	
D20,D21	Микросхема 544УД18У3 АЕЯР.431130.588 ТУ	2	
D22	Микросхема 1481СА1Р	1	
	АЕЯР.431350.431-01 ТУ		
G1	Батарея литиевая CR 1/2 AA CD	1	Varta
S1...	Переключатель ПКн105-5В вариант I	3	
S3	АГО.360.034 ТУ		
R1...	Резистор Р1-112-0,125-10 кОм+-5%-К		
R3	ДАРГ.434110.004 ТУ	3	
R4	Резистор Р1-112-0,125-8.06 кОм+-1%-К		
	ДАРГ.434110.004 ТУ	1	

Инв. № подл.

Подп. и дата

Взам. инв. №

Инв. № дубл.

Подп. и дата

Изм

Лист

№ докум.

Подп.

Дата

ПГУ.100503.С3.2.ДП.12ПК119.02.ПЗ

Лист

4

Копировал
Формат А4

					Поз. обозначение	Наименование	Кол.	Примечание
					R5...	Резистор P1-112-0,125-10 кОм+-5%-K		
					R11	ДАРГ.434110.004 ТУ	7	
					R12...	Резистор P1-112-0,125-50 кОм+-5%-K		
					R14	ДАРГ.434110.004 ТУ	3	
					R15	Резистор P1-112-0,125-10 кОм+-5%-K		
						ДАРГ.434110.004 ТУ	1	
					R16	Резистор P1-112-0,125-50 кОм+-5%-K		
						ДАРГ.434110.004 ТУ	1	
					R17	Резистор P1-112-0,125-150 Ом+-5%-K		
						ДАРГ.434110.004 ТУ	1	
					R18,R19	Резистор P1-112-0,125-50 кОм+-5%-K		
						ДАРГ.434110.004 ТУ	2	
					R20...	Резистор P1-112-0,125-10 кОм+-5%-K		
					R22	ДАРГ.434110.004 ТУ	3	
					R23...	Резистор P1-112-0,125-10 кОм+-1%-K		
					R26	ДАРГ.434110.004 ТУ	4	
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата	R27...	Резистор P1-112-0,125-10 кОм+-5%-K		
					R30	ДАРГ.434110.004 ТУ	4	
					R31...	Резистор P1-112-0,125-150 Ом+-5%-K		
					R36	ДАРГ.434110.004 ТУ	6	
					R37...	Резистор P1-112-0,125-1 кОм+-5%-K		
					R48	ДАРГ.434110.004 ТУ	12	
					R49...	Резистор P1-112-0,125-150 Ом+-5%-K		
					R54	ДАРГ.434110.004 ТУ	6	
					R55,R56	Резистор P1-112-0,125-5.49 кОм+-1%-K		
						ДАРГ.434110.004 ТУ	2	
					R57	Резистор P1-112-0,125-6.49 кОм+-1%-K		
						ДАРГ.434110.004 ТУ	1	
					R58	Резистор переменный СП5-16ВА-0,25-10 кОм+-5%		
						ОЖО.468.519 ТУ	1	
Инв. № подл.					ПГУ.100503.С3.2.ДП.12ПК119.02.ПЗ			Лист
								5
	Изм	Лист	№ докум.	Подп.				Дата

Копировал

Формат А4

Поз. обозначение		Наименование		Кол.	Примечание
R59,R60	Резистор P1-112-0,125-30.1 кОм+-1%-К				
	ДАРГ.434110.004 ТУ		2		
R61	Резистор P1-112-0,125-6.49 кОм+-1%-К				
	ДАРГ.434110.004 ТУ		1		
R62	Резистор P1-112-0,125-9.10 кОм+-1%-К				
	ДАРГ.434110.004 ТУ		1		
R63	Резистор P1-112-0,125-20 кОм+-5%-К				
	ДАРГ.434110.004 ТУ		1		
R64,R65	Резистор				
	ОС С2-33Н-0,25-10,2 Ом+-1%-А-В-В				
	ОЖ0.467.093 ТУ ОЖ0.467.138 ТУ		2		
R66,R67	Резистор P1-112-0,125-3.65 кОм+-1%-К				
	ДАРГ.434110.004 ТУ		2		
R68	Резистор P1-112-0,125-6.49 кОм+-1%-К				
	ДАРГ.434110.004 ТУ		1		
R69	Резистор P1-112-0,125-9.31 кОм+-1%-К				
	ДАРГ.434110.004 ТУ		1		
R70	Резистор P1-112-0,125-2.80 кОм+-1%-К				
	ДАРГ.434110.004 ТУ		1		
R71	Резистор P1-112-0,125-22 кОм+-5%-К				
	ДАРГ.434110.004 ТУ		1		
R72,R73	Резистор P1-112-0,125-30.1 кОм+-1%-К				
	ДАРГ.434110.004 ТУ		2		
R74,R75	Резистор P1-112-0,125-6.49 кОм+-1%-К				
	ДАРГ.434110.004 ТУ		2		
R76	Резистор P1-112-0,125-4.3 кОм+-5%-К				
	ДАРГ.434110.004 ТУ		1		
R77	Резистор P1-112-0,125-19.6 кОм+-1%-К				
	ДАРГ.434110.004 ТУ		1		
R78,R79	Резистор P1-112-0,125-150 Ом+-5%-К				
	ДАРГ.434110.004 ТУ		2		

Поз. обозначение	Наименование	Кол.	Примечание
R80	Резистор P1-112-0,125-487 Ом+-1%-K		
	ДАРГ.434110.004 ТУ	1	
R81...	Резистор P1-112-0,125-10 кОм+-5%-K		
R83	ДАРГ.434110.004 ТУ	3	
R84	Резистор P1-112-0,125-1 кОм+-5%-K		
	ДАРГ.434110.004 ТУ	1	
R85	Резистор P1-112-0,125-4.7 кОм+-5%-K		
	ДАРГ.434110.004 ТУ	1	
R86	Резистор P1-112-0,125-1 кОм+-5%-K		
	ДАРГ.434110.004 ТУ	1	
R87	Резистор ОС С2-33Н-1-560 Ом+-5%-А-Д-В		
	ОЖ0.467.093 ТУ ОЖ0.467.138 ТУ	1	
R88	Резистор P1-112-0,125-10 кОм+-5%-K		
	ДАРГ.434110.004 ТУ	1	
R89	Резистор P1-112-0,125-6.49 кОм+-1%-K		
	ДАРГ.434110.004 ТУ	2	
R90	Резистор ОС С2-33Н-2-56 Ом+-5%-А-Д-В		
	ОЖ0.467.093 ТУ ОЖ0.467.138 ТУ	1	
R91,R92	Резистор ОС С2-33Н-0,25-10,2 Ом+-1%-А-В-В		
	ОЖ0.467.093 ТУ ОЖ0.467.138 ТУ	2	
R93, R94	Резистор P1-112-0,125-10 кОм+-5%-K		
	ДАРГ.434110.004 ТУ	2	
V1,2	Транзисторная сборка 2П7241АС9	2	
	АЕЯР.432140.605 ТУ		
V3	Транзисторная матрица 1НТ251	1	
	И93.456.000 ТУ (по У80.073.038 ГЧ)		

Подп. и дата

Инв. № дубл.

Взам. инв. №

Подп. и дата

Инв. № подл.

ПГУ.100503.СЗ.2.ДП.12ПК119.02.ПЗ

Лист
7

Изм

Лист

№ докум.

Подп.

Дата

Копировал

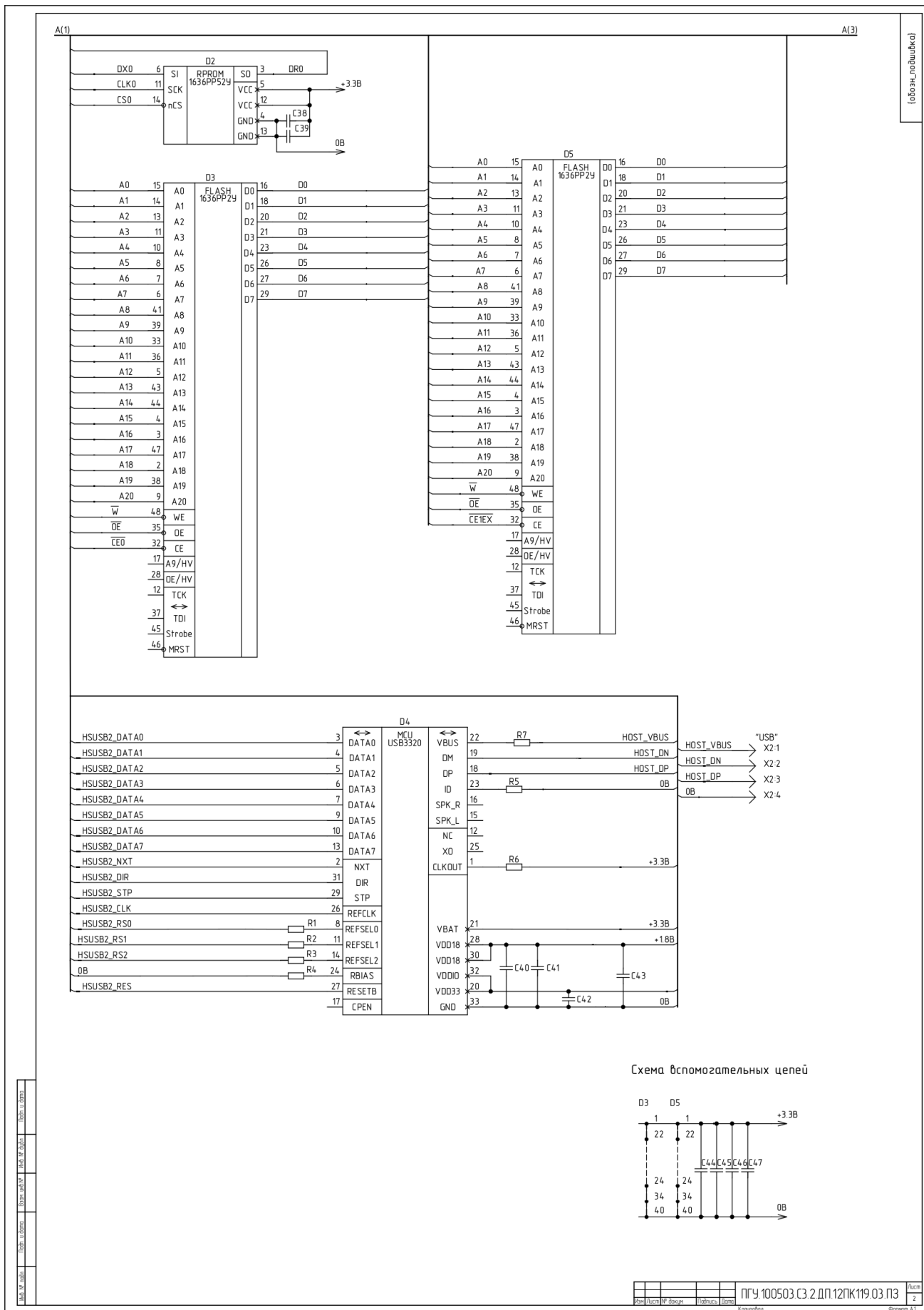
Формат А4

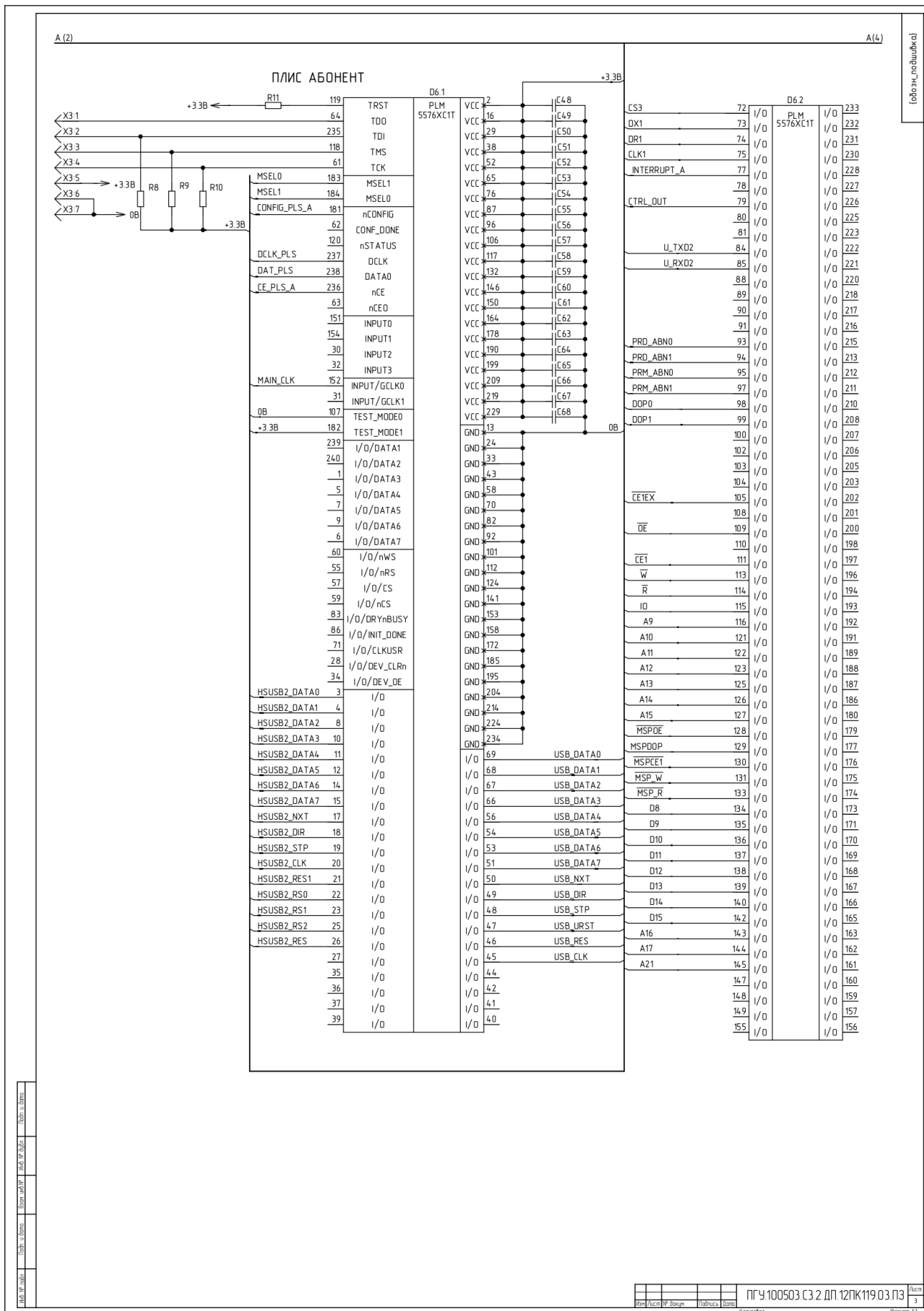
[illegible]

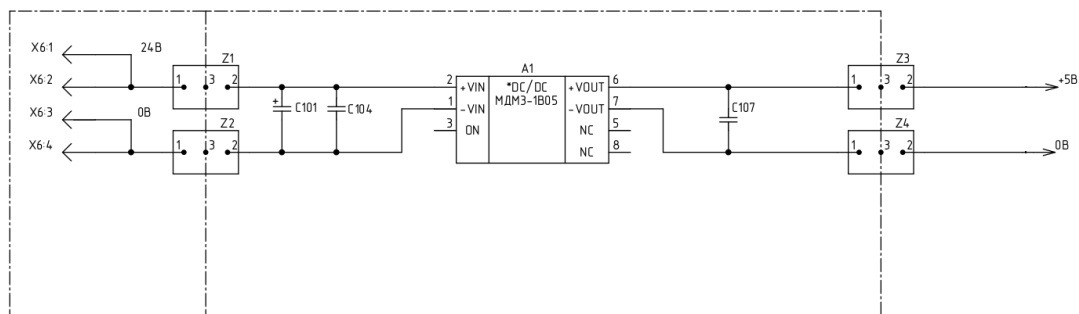
ПРИЛОЖЕНИЕ В

(обязательное)

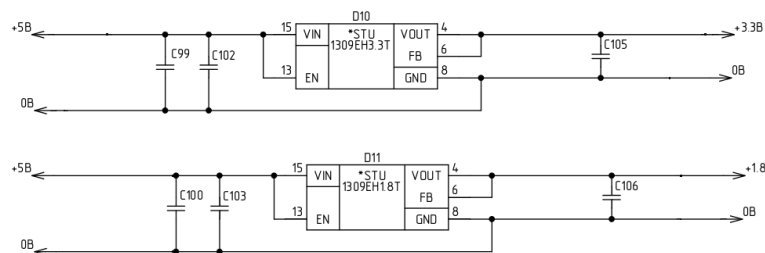
Принципиальная схема модуля СКЗИ.







(обозначения)



Имя файла	Имя файла	Имя файла	Имя файла	Имя файла
Имя файла	Имя файла	Имя файла	Имя файла	Имя файла

Имя файла	Имя файла	Имя файла	Имя файла	Имя файла
Имя файла	Имя файла	Имя файла	Имя файла	Имя файла

ПГУ 100503.СЗ.2.ДП.12ПК119.03.ПЗ

Имя файла

Имя файла

ПРИЛОЖЕНИЕ Г

(обязательное)

Алгоритм ГОСТ 34.12-2015 «Магма», работающий в режиме имитовставки по
ГОСТ 34.13-2015.



ПРИЛОЖЕНИЕ Д

(обязательное)

Код программы проверки целостности основанный на алгоритме ГОСТ 34.12-2015 «Магма», работающий в режиме имитовставки по ГОСТ 34.13-2015.

```

#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <uart.h>
#include <gpio.h>
#include <spi.h>

#define ADDRESSOFSTARTKI 0x60240000
#define ADDRESSOFENDKI 0x6024FFFEF
#define ADDRESSOFSTARTPO 0x40000000
#define ADDRESSOFENDPO 0x403FFFFFF
#define IMITOOFKI 0x00040000
#define IMITOOFPO 0x00040008
#define IMITOOFABONENT 0x00040010
#define IMITOOFMANAGE 0x00040018
#define MASKFORBLOCKS 0xFFFFFFFF
#define MASKFORFIRSBIT 0x7FFFFFFF;

typedef unsigned char byte;
typedef unsigned long block32;

block32 keyNSD[8]={
    0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
};

void getKey ()
{
    block32 midleKey;
    int i,j;
    block32 blockOfKey;

    for (i=0; i<=7; i++)
    {

```

```

        for (j=0; j<=1; j++)
        {
            blockOfKey = *(KEYOFNSD + ( j * 16 ) + ( i * 32 ));
            midleKey = *blockOfKey;
            midleKey <<= 16;
        }
        keyNSD[i] = midleKey;
    }
}

block32* magma(block32 *block)
{
    const byte tau[8][16] =
    {
        0xC, 0x4, 0x6, 0x2, 0xA, 0x5, 0xB, 0x9, 0xE, 0x8, 0xD, 0x7,
        0x0, 0x3, 0xF, 0x1,
        0x6, 0x8, 0x2, 0x3, 0x9, 0xA, 0x5, 0xC, 0x1, 0xE, 0x4, 0x7,
        0xB, 0xD, 0x0, 0xF,
        0xB, 0x3, 0x5, 0x8, 0x2, 0xF, 0xA, 0xD, 0xE, 0x1, 0x7, 0x4,
        0xC, 0x9, 0x6, 0x0,
        0xD, 0x8, 0x2, 0x1, 0xD, 0x4, 0xF, 0x6, 0x7, 0x0, 0xA, 0x5,
        0x3, 0xE, 0x9, 0xB,
        0x7, 0xF, 0x5, 0xA, 0x8, 0x1, 0x6, 0xD, 0x0, 0x9, 0x3, 0xE,
        0xB, 0x4, 0x2, 0xC,
        0x5, 0xD, 0xF, 0x6, 0x9, 0x2, 0xC, 0xA, 0xB, 0x7, 0x8, 0x1,
        0x4, 0x3, 0xE, 0x0,
        0x8, 0xE, 0x2, 0x5, 0x6, 0x9, 0x1, 0xC, 0xF, 0x4, 0xB, 0x0,
        0xD, 0xA, 0x3, 0x7,
        0x1, 0x7, 0xE, 0xD, 0x0, 0x5, 0x8, 0x3, 0x4, 0xF, 0xA, 0x6,
        0x9, 0xC, 0xB, 0x2
    };

    block32 leftPartOfBlock = 0, rightPartOfBlock = 0, RES = 0;
    block32 block_temp[2] = {0};
    block32 modifiedBlock[2] = {0};
    int numberOfBlockKey = 0;
    int k,n,q,i,x;
    byte first_byte, second_byte, character;

```

```

block_temp[0] = block[0];
block_temp[1] = block[1];
modifiedBlock[0] = 0;
modifiedBlock[1] = 0;
i = 0;
x = 1;

rightPartOfBlock = block_temp[1];

leftPartOfBlock = block_temp[0];
numberOfBlockKey = 0;

for (k = 0; k < 32; k++)
{
    if (k == 24) numberOfBlockKey = 7;
    RES = keyNSD[numberOfBlockKey] + leftPartOfBlock;
    first_byte = 0, second_byte = 0, character = 0;
    n = 7;
    for ( q = 3; q >= 0; q-- )
    {
        character = *((byte *)&RES + q);
        first_byte = (character & 0xF0) >> 4;
        second_byte = (character & 0x0F);
        first_byte = tau[n][first_byte];
        n--;
        second_byte = tau[n][second_byte];
        n--;
        character = (first_byte << 4) | second_byte;
        *((byte *)&RES + q) = character;
    }
    RES = (RES << 11) | (RES >> 21);
    RES = rightPartOfBlock^RES;
    if (k < 31)
    {
        rightPartOfBlock = leftPartOfBlock;
        leftPartOfBlock = RES;
    }
}

```

```

    }

    if (k<24)
    {
        numberOfBlockKey++;
        if (numberOfBlockKey>7) numberOfBlockKey = 0;
    }
    else
    {
        numberOfBlockKey--;
        if (numberOfBlockKey < 0) numberOfBlockKey = 7;
    }
}

rightPartOfBlock = RES;
block[0] = leftPartOfBlock;
block[1] = rightPartOfBlock;
return block;
}

int getImitoForUART (UART_TypeDef *UART, block32 *addressForImito)
{
    block32 R[2] = {0};
    block32 B[2] = {0, 27};
    block32 imito[2] = {0};
    block32 keyForImito[2] = {0};
    block32 firstBit=0;
    block32 secondFBit = 0;
    block32 Key1[2] = {0};
    block32 Key2[2] = {0};
    block32 null[2] = {0};
    block32 blockForFilling[2];
    block32 blockPast[2]={0}, blockNow[2] = {0};
    int filedBlocks;
    int filling = 0;
    int i;
    int iteration;
    UART_InitTypeDef UART_InitStructure;

```

```

R[0] = *magma(null);
R[1] = *((byte *)&R + 4);
firstBit = R[0] / MASKFORFIRSBIT;

if (firstBit == 0)
{
    Key1[0] = (R[0] <= 1);
    secondFBit = R[1] / MASKFORFIRSBIT;
    Key1[0] |= secondFBit;
    Key1[1] = (R[1] <= 1);
    Key1[1] |= firstBit;

}
else
{
    Key1[0] = (R[0] <= 1);
    secondFBit = R[1] / MASKFORFIRSBIT;
    Key1[0] |= secondFBit;
    Key1[0] ^= B[0];
    Key1[1] = (R[1] < 1);
    Key1[1] |= firstBit;
    Key1[1] ^= B[1];
}

firstBit = Key1[0] / MASKFORFIRSBIT;

if (firstBit == 0)
{
    Key2[0] = (R[0] <= 1);
    secondFBit = R[1] / MASKFORFIRSBIT;
    Key2[0] |= secondFBit;
    Key2[1] = (R[1] <= 1);
    firstBit = R[0] / MASKFORFIRSBIT;
    Key2[1] |= firstBit;
}

```

```

    }
else
{
    Key2[0] = (R[0] <= 1);
    secondFBit = R[1] / MASKFORFIRSBIT;
    Key2[0] |= secondFBit;
    Key2[0] ^= B[0];
    Key2[1] = (R[1] <= 1);
    firstBit = R[0] / MASKFORFIRSBIT;
    Key2[1] |= firstBit;
    Key2[1] ^= B[1];
}

UART_InitStructure.System_freq = 8000000;
UART_InitStructure.BaudRate = 38400;
UART_InitStructure.TypeParity = 0x00000000;
UART_InitStructure.Parity = 0x00000000;
UART_InitStructure.FlowControl = 0x00000000;
UART_InitStructure.Mode = 0x00000003;

uart_init(UART, &UART_InitStructure);

while (UART_FIFO_TX_FULL(UART) != 1)
{
};
UART_SEND_BYTE(0xFF, UART);

iteration = 0;

while (UART_BREAK_DETECT(UART) == 0)
{
    for (i = 0; i <= 7; i++)
    {
        if (i <= 3)
        {
            blockForFilling[0] = UART_GET_BYTE(UART);
        }
    }
}

```

```

else
{
    blockForFilling[1] = UART_GET_BYTE(UART);
}

if (UART_BREAK_DETECT(UART) == 1)
{
    filedBlocks = i;
    break;
}

if (i <= 3)
{
    blockForFilling[0] <<= 8;
}

else
{
    blockForFilling[1] <<= 8;
}
}

if (filedBlocks != 7)
{
    if (filedBlocks < 3)
    {
        blockForFilling[0] <<= 1;
        blockForFilling[0] |= 0x1;
        blockForFilling[0] <<= (8*(7-filedBlocks)-1);
    }
    else
    {
        blockForFilling[1] |= 0x1;
        blockForFilling[1] <<= (8*(7-filedBlocks)-1);
    }
}

```

```

        filling = 1;
    }

    if (iteration == 0)
    {
        blockPast[0] = blockForFilling[0];
        blockPast[1] = blockForFilling[1];
        blockPast[0] = *magma(blockPast);
        blockPast[1] = *((byte *)&blockPast + 4);
    }
    else
    {
        blockNow[0] = blockForFilling[0];
        blockPast[0] = blockPast[0]^blockNow[0];
        blockNow[1] = blockForFilling[1];
        blockPast[1] = blockPast[1]^blockNow[1];

        if (UART_BREAK_DETECT(UART) == 1)
        {
            if (filling == 1)
            {
                keyForImito[0] = Key1[0];
                keyForImito[1] = Key1[1];
            }
            else
            {
                keyForImito[0] = Key2[0];
                keyForImito[1] = Key2[1];
            }

            imito[0] = blockPast[0]^keyForImito[0];
            imito[1] = blockPast[1]^keyForImito[1];
            imito[0] = *magma(imito);
            imito[1] = *((byte *)&imito + 4);
        }
    }
}

```

```

        iteration++;
    }

    if ( ( *imito != *addressForImito ) || ( *((byte *)&imito +
4 ) != *((byte *)&addressForImito + 4)) )
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

int getImitoForSPI (SPI_TypeDef *SPI, block32 *addressForImito)
{
    block32 R[2] = {0};
    block32 B[2] = {0, 27};
    block32 imito[2] = {0};
    block32 keyForImito[2] = {0};
    block32 firstBit=0;
    block32 secondFBit = 0;
    block32 Key1[2] = {0};
    block32 Key2[2] = {0};
    block32 null[2] = {0};
    block32 blockForFilling[2];
    block32 blockPast[2] = {0}, blockNow[2] = {0};
    int filedBlocks;
    int filling = 0;
    int i;
    int iteration;
    SPI_InitTypeDef SPI_InitStructure;

    R[0] = *magma(null);
    R[1] = *((byte *)&R + 4);
    firstBit = R[0] / MASKFORFIRSBIT;

```

```

if (firstBit == 0)
{
    Key1[0] = (R[0] <= 1);
    secondFBit = R[1] / MASKFORFIRSBIT;
    Key1[0] |= secondFBit;
    Key1[1] = (R[1] <= 1);
    Key1[1] |= firstBit;
}
else
{
    Key1[0] = (R[0] <= 1);
    secondFBit = R[1] / MASKFORFIRSBIT;
    Key1[0] |= secondFBit;
    Key1[0] ^= B[0];
    Key1[1] = (R[1] < 1);
    Key1[1] |= firstBit;
    Key1[1] ^= B[1];
}

firstBit = Key1[0] / MASKFORFIRSBIT;

if (firstBit == 0)
{
    Key2[0] = (R[0] <= 1);
    secondFBit = R[1] / MASKFORFIRSBIT;
    Key2[0] |= secondFBit;
    Key2[1] = (R[1] <= 1);
    firstBit = R[0] / MASKFORFIRSBIT;
    Key2[1] |= firstBit;
}
else
{
    Key2[0] = (R[0] <= 1);
    secondFBit = R[1] / MASKFORFIRSBIT;

```

```

    Key2[0] |= secondFBit;
    Key2[0] ^= B[0];
    Key2[1] = (R[1] <= 1);
    firstBit = R[0] / MASKFORFIRSBIT;
    Key2[1] |= firstBit;
    Key2[1] ^= B[1];
}

SPI_InitStructure.Word_len = 0x0;
SPI_InitStructure.Divide16 = 0x0;
SPI_InitStructure.SPI_en = 0x1;
SPI_InitStructure.Three_wire_en = 0x0;
SPI_InitStructure.PM = 0x0;
SPI_InitStructure.Reverse_data = 0x1;

spi_init(SPI_TypeDef *SPI, SPI_InitTypeDef &SPI_InitStruct);

while(SPI_ST_NF(SPI) == 1);
SPI_TX_DATA(SPI, MASKFORBLOCKS);

while (SPI_ST_TIP(SPI) == 1)
{
    iteration = 0;
    for (i = 0; i <= 1; i++)
    {
        if ( i = 0 )
        {
            blockForFilling[0] = SPI_RX_DATA(SPI);
        }
        else
        {
            blockForFilling[1] = SPI_RX_DATA(SPI);
        }

        if (SPI_ST_TIP(SPI) == 0)
        {

```

```

        filedBlocks = i;
        break;
    }

}

if (filedBlocks != 1)
{
    blockForFilling[1] <= 1;
    blockForFilling[1] |= 0x1;
    blockForFilling[1] <= (32*(1-filedBlocks)-1);
    filling = 1;
}

if (iteration == 0)
{
    blockPast[0] = blockForFilling[0];
    blockPast[1] = blockForFilling[1];
    blockPast[0] = *magma(blockPast);
    blockPast[1] = *((byte *)blockPast + 4);
}
else
{
    blockNow[0] = blockForFilling[0];
    blockPast[0] = blockPast[0]^blockNow[0];
    blockNow[1] = blockForFilling[1];
    blockPast[1] = blockPast[1]^blockNow[1];

    if (SPI_ST_TIP(SPI) == 0)
    {
        if (filling == 1)
        {
            keyForImito[0] = Key1[0];
            keyForImito[1] = Key1[1];
        }
        else

```

```

        {
            keyForImito[0] = Key2[0];
            keyForImito[1] = Key2[1];
        }

        imito[0] = blockPast[0]^keyForImito[0];
        imito[1] = blockPast[1]^keyForImito[1];
        imito[0] = *magma(imito);
        imito[1] = *((byte *)imito + 4);
    }
}

iteration++;
}

if (( *imito != addressForImito ) || ( *((byte *)&imito +
4 ) != *((byte *)&addressForImito + 4)))
{
    return 1;
}

else
{
    return 0;
}
}

int getImitoForDATA (block32 addressOfStart, block32 addressOfEnd,
block32 enableBit, block32 addressForImito )
{
    block32 R[2] = {0};
    block32 B[2] = {0, 27};
    block32 imito[2] = {0};
    block32 keyForImito[2] = {0};
    block32 firstBit = 0;
    block32 secondFBit = 0;
    block32 Key1[2] = {0};
    block32 Key2[2] = {0};

```

```

block32 null[2] = {0};
block32 blockForFilling[2];
block32 blockPast[2] = {0}, blockNow[2] = {0};
int filedBlocks;
int filling = 0;
int i = 0;
int iteration = 0;
int mode = 0;
int different;

R[0] = *magma(null);
R[1] = *((byte *)R + 4);
firstBit = R[0] / MASKFORFIRSBIT;

if (firstBit == 0)
{
    Key1[0] = (R[0] << 1);
    secondFBit = R[1] / MASKFORFIRSBIT;
    Key1[0] |= secondFBit;
    Key1[1] = (R[1] <=< 1);
    Key1[1] |= firstBit;
}
else
{
    Key1[0] = (R[0] <=< 1);
    secondFBit = R[1] / MASKFORFIRSBIT;
    Key1[0] |= secondFBit;
    Key1[0] ^= B[0];
    Key1[1] = (R[1] <=< 1);
    Key1[1] |= firstBit;
    Key1[1] ^= B[1];
}

firstBit = Key1[0] / MASKFORFIRSBIT;

```

```

if (firstBit == 0)
{
    Key2[0] = (R[0] <= 1);
    secondFBit = R[1] / MASKFORFIRSBIT;
    Key2[0] |= secondFBit;
    Key2[1] = (R[1] <= 1);
    firstBit = R[0] / MASKFORFIRSBIT;
    Key2[1] |= firstBit;
}
else
{
    Key2[0] = (R[0] <= 1);
    secondFBit = R[1] / MASKFORFIRSBIT;
    Key2[0] |= secondFBit;
    Key2[0] ^= B[0];
    Key2[1] = (R[1] <= 1);
    firstBit = R[0] / MASKFORFIRSBIT;
    Key2[1] |= firstBit;
    Key2[1] ^= B[1];
}

if ( enableBit == 0xFF )
{
    mode = 7;
}

else if ( enableBit == 0xFFFF )
{
    mode = 3;
}

else
{
    return 1; //??
}

```

```

    for ( addressOfStart; addressOfStart <= addressOfEnd;
addressOfStart + 64 )
    {

        for (i = 0; i <= mode; i++)
        {
            if ((mode == 3) && (i < 1))
            {
                blockForFilling[0] |= *addressOfStart;
                blockForFilling[0] <<= 16;
            }
            else if ((mode == 3) && (i == 1))
            {
                blockForFilling[1] |= *addressOfStart;
            }
            else if ((mode == 3) && ((i > 1) && (i < mode)))
            {
                blockForFilling[1] |= *addressOfStart;
                blockForFilling[1] <<= 16;
            }
            else if ((mode == 3) && (i == mode))
            {
                blockForFilling[1] |= *addressOfStart;
            }
            else if ((mode == 7) && (i < 3))
            {
                blockForFilling[0] |= *addressOfStart;
                blockForFilling[0] <<= 8;
            }
            else if ((mode == 7) && (i == 3))
            {
                blockForFilling[1] = *addressOfStart;
            }
            else if ((mode == 7) && ((i > 3) && (i < mode)))

```

```

{
    blockForFilling[1] |= *addressOfStart;
    blockForFilling[1] <=& 8;
}
else
{
    blockForFilling[1] |= *addressOfStart;
}
}

if (addressOfStart > addressOfEnd )
{
    different = addressOfStart - addressOfEnd;

    if (different > 32)
    {
        blockForFilling[0] <=& 1;
        blockForFilling[0] |= 0x1;
        blockForFilling[0] <=& ( different - 33 );
        blockForFilling[1] = 0;
    }
    else
    {
        if (different == 0)
        {
            break;
        }
        else
        {
            blockForFilling[1] <=& 1;
            blockForFilling[1] |= 0x1;
            blockForFilling[1] <=& (different-1);
        }
    }
    filling = 1;
}
}

```

```

        if (filling == 1)
        {
            keyForImito[0] = Key1[0];
            keyForImito[1] = Key1[1];
        }
    else
    {
        keyForImito[0] = Key2[0];
        keyForImito[1] = Key2[1];
    }

    imito[0] = blockPast[0]^keyForImito[0];
    imito[1] = blockPast[1]^keyForImito[1];
    imito[0] = *magma(imito);
    imito[1] = *((byte *)&imito + 4);

    if (( *imito != *addressForImito ) || ( *((byte *)&imito +
4 ) != *((byte *)&addressForImito + 4)))
    {
        return 1;
    }
    return 0;
}

int main()
{
    char status[3] = {'O','k',' '};
    getKey();

    if ( getImitoForDATA ((block32 *)ADDRESSOFSTARTKI, (block32
*)ADDRESSOFENDKI, 0xFFFF, (block32 *)IMITOOFKI)  ||
        getImitoForDATA ((block32 *)ADDRESSOFSTARTPO, (block32
*)ADDRESSOFENDPO, 0xFF, (block32 *)IMITOOFPO)  ||
        getImitoForSPI(SPI1, (block32 *)IMITOOFABONENT)          ||
        getImitoForUART(UART0, (block32 *)IMITOOFMANAGE) == 1)

```

```

{
    UART_SEND_BYTE(0x47, UART0);
    SPI_TX_DATA(SPI0, 0x47);
    status[0] = 'E';
    status[1] = 'r';
    status[2] = 'r';
    return 1;
}

else
{
    status[0] = 'O';
    status[1] = 'k';
    status[2] = ' ';
    return 0;
}
}

```