

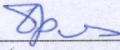
МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
Федеральное государственное бюджетное образовательное учреждение высшего
образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

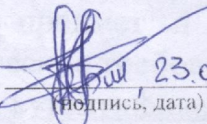
Факультет «Вычислительная техника» (наименование)	Кафедра «Системы автоматизированного проектирования» (наименование)
Направление подготовки	09.03.01 «Информатика и вычислительная техника» (код и наименование)
Профиль	«Системы автоматизированного проектирования» (наименование)

БАКАЛАВРСКАЯ РАБОТА
на тему

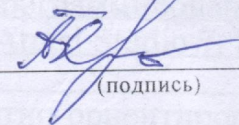
АППАРАТНЫЕ СРЕДСТВА АУТЕНТИФИКАЦИИ
ДЛЯ ИНФОРМАЦИОННЫХ СИСТЕМ

Студент	 (подпись, дата)	ЯУЖЕВ А.А. (фамилия, инициалы)
---------	--	-----------------------------------

Руководитель	 (подпись, дата)	Брякин Л.А. (фамилия, инициалы)
--------------	--	------------------------------------

Нормоконтролёр	проф. каф. САПР ПГУ (должность, место работы)	 (подпись, дата)	Тондаи Л.С. (фамилия, инициалы)
----------------	--	---	------------------------------------

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

Заведующий кафедрой	 (подпись)	Бегларов Х.М. (фамилия, инициалы)
---------------------	--	--------------------------------------

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

Секретарь ГЭК	_____ (подпись)	_____ (фамилия, инициалы)
---------------	--------------------	------------------------------

Пенза 2017

РЕФЕРАТ

Пояснительная записка содержит 89 страниц, 14 рисунков, 2 таблицы, 19 библиографических ссылок, 8 приложений.

АППАРАТНЫЕ СРЕДСТВА, АУТЕНТИФИКАЦИЯ, МИКРОКОНТРОЛЛЕР, ПАРОЛЬ, ПРОГРАММА, ФУНКЦИЯ ГУБКИ, AVR.

Объектом разработки являются аппаратные средства в виде USB-совместимого устройства, предназначенного для автоматизации процесса аутентификации пользователя в информационных системах.

Цель работы – создание аппаратно-программного комплекса для ЭВМ, обеспечивающего упрощение процесса аутентификации пользователя путем хранения и вывода списка обслуживаемых систем и данных, необходимых для аутентификации в них (логин, пароль или ПИН-код). В процессе проектирования разработаны электрическая принципиальная схема устройства, структура программного обеспечения, алгоритмы обработки данных, составлена и отлажена программа для микроконтроллера на языке C++, а также прототип устройства.

Эффективность внедрения аппаратных средств аутентификации обеспечивается сокращением времени на авторизацию в информационной системе, а также упрощает процесс смены текущих данных аутентификации путем снижения объема запоминаемой пользователем информации. Перед промышленным внедрением следует провести опытную эксплуатацию устройства, а также убедиться в полной совместимости устройства с конкретной информационной системой.

Основные характеристики устройства: 2,5 КБ ОЗУ, 32 КБ ПЗУ, 32 КБ EEPROM, 128x32 дисплей. Совместимо с ОС Windows и Linux.

					ВКР 090301.20 17 81 01			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.	Яужев А.А.				Аппаратные средства ... (см. титульный лист) Пояснительная записка	Литера	Лист	Листов
Провер.	Брякин Л.А.						1	1
Н.контр.	Бождай А.С.					ПГУ, гр. 13 ВВ2		
Т. контр.								
Утв.	Бершадский А.М.							

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	6
1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ.....	8
1.1. Аутентификация и авторизация.....	8
1.2. Политики безопасности	10
1.3. Обзор аналогов	12
1.4. Предложения по проекту.....	14
2. ПРОЕКТИРОВАНИЕ	16
2.1. Функциональные требования.....	16
2.2. Структурная схема устройства	17
2.3. Интерфейсы взаимодействия микросхем	18
2.4. Основного алгоритм программы	22
2.5. Элементная база устройства.....	28
2.5.1. Микроконтроллер.....	28
2.5.2. Микросхема постоянной памяти	30
2.5.3. Модуль дисплея.....	31
3. РЕАЛИЗАЦИЯ.....	32
3.1. Разработка схемы электрической принципиальной	32
3.2. Разработка программы для микроконтроллера.....	34
3.2.1. Класс криптографии.....	35
3.2.2. Функции взаимодействия с пользователем	36
3.2.3. Класс взаимодействия с графическим дисплеем	39
3.2.4. Класс взаимодействия с EEPROM	41
3.2.5. Класс хранилища данных	42
3.2.6. Класс выработки секрета.....	44
4. ОТЛАДКА И ТЕСТИРОВАНИЕ	45
4.1. Моделирование электрической схемы	45
4.2. Создание прототипа устройства	46

4.3. Отладка и тестирование программы	47
ЗАКЛЮЧЕНИЕ	48
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ.....	49
Приложение А. Структурная схема устройства	51
Приложение Б. Принципиальная схема устройства	52
Приложение В. Схема взаимодействия классов.....	53
Приложение Г. Исходный код программы	54
1. Класс криптографии	54
2. Функции взаимодействия с пользователем	57
3. Класс взаимодействия с графическим дисплеем	57
4. Класс взаимодействия с EEPROM.....	71
5. Класс хранилища данных	73
6. Класс выработки секрета	78
Приложение Д. Граф диалога с пользователем	81
Приложение Е. Блок-схема алгоритма генерации секрета.....	82
Приложение Ж. Прототип устройства.....	83
Приложение З. Набор функциональных тестов	84

ВВЕДЕНИЕ

Системы аутентификации и авторизации являются неотъемлемой частью современных информационных систем. Операционные системы, базы данных и веб сервисы используют пары логин-пароль для авторизации пользователя в системе и наделения его определенными правами, поэтому содержание этих данных в тайне является одной из важнейших задач пользователя.

Требования к логинам и паролям в различных системах зависят, в общем случае, от политики безопасности, установленной администратором системы. Это может быть ограничение на длину пароля, ограничение или обязательность наличия специальных символов, проверка «надежности» пароля по словарям или шаблонам; в более редких случаях — требование смены пароля после некоторого периода его использования (например, каждый месяц), в том числе с проверкой уникальности нового пароля (для предыдущих N паролей пользователя).

Независимо от политики безопасности, избранной администратором информационной системы, пользователю необходимо запомнить пару логин-пароль, чтобы получать доступ к системе. Комбинация различных требований политики безопасности может привести к проблеме хранения данных для аутентификации, поскольку один и тот же человек может являться пользователем нескольких информационных систем или может иметь несколько профилей в одной и той же системе. В худшем случае многочисленные пары логин-пароль могут быть записаны на физический носитель и храниться в открытом виде, в лучшем случае для этой цели будет использован один из менеджеров паролей.

Менеджеры паролей в общем случае предоставляют защищенное хранилище для данных аутентификации, доступ к которому осуществляется с помощью специализированного программного обеспечения или сочетания программных и аппаратных средств. В таком случае возникает необходимость

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		6

устанавливать или переносить программу для доступа к защищенному хранилищу вместе с самим хранилищем, но такое решение, в большинстве случаев, не является кроссплатформенным.

В данной работе будет рассмотрено решение на основе аппаратных средств автоматизации аутентификации, обеспечивающих хранение данных для выработки аутентификационных данных для информационных систем и обеспечивающих быструю смену любого фрагмента этих данных (логина, пароля или кода доступа) по запросу пользователя с возможностью быстрого возврата к предыдущему состоянию.

Целью работы является разработка устройства, упрощающего процедуру аутентификации для пользователя нескольких информационных систем, минимально зависящего от типа информационной системы и ее операционной системы.

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		7

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Аутентификация и авторизация

Основным понятием в данной работе является аутентификация. Аутентификация предполагает процесс сравнения данных, предоставляемых пользователем (фактор) с данными, хранящимися в информационной системе. В процессе сравнения осуществляется идентификация пользователя на основе предоставленных данных и при обнаружении соответствия пользователь авторизуется — наделяется соответствующим набором прав и ограничений.

Различные источники аутентификационной информации, а также отличительные черты и характеристики, идентифицирующие пользователя информационной системы, являются факторами аутентификации. Набор используемых факторов определяется требованиями к безопасности информационной системы.

В общем случае выделяют 3 фактора аутентификации:

1) То, что известно пользователю — пароль.

Пароль — это секрет, которым должен обладать только авторизованный субъект. Паролем может выступать слово, сочетание слов, персональный идентификационный номер (ПИН) или сочетание из произвольных символов. Данный фактор легко реализуется, имеет низкую стоимость внедрения, но имеет существенные недостатки: сохранить пароль в тайне является сложной задачей, поскольку способы его похищения или подбора постоянно развиваются (фишинг, троянские программы, специальные словари и т.д.).

2) То, что имеет пользователь — устройство аутентификации.

Для этого фактора важен тот факт, что пользователь обладает неповторимым, уникальным предметом, содержащим некоторую характеристику (например, ключ для замка, личная печать, специальный файл данных). Применительно к информационным системам в качестве хранилища

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		8

характеристики могут выступать смарт-карты, а также специальные токены. Для злоумышленника получение физического устройства является гораздо более сложной задачей, а в случае кражи пользователь может сообщить об этом администратору. Данный фактор обеспечивает более высокий уровень безопасности, однако является дорогостоящим: в информационной системе должны иметься модули для программной поддержки фактора, а также физические устройства-считыватели для получения характеристики с устройства.

3) То, что является частью пользователя — биометрика.

Для этого фактора характеристикой является физическая особенность пользователя. Это может быть отпечаток пальца или ладони, голос, особенности глаза, лица и т.п. Для пользователя данный фактор является наиболее простым, поскольку не нужно запоминать пароли и переносить с собой устройство аутентификации, однако требования к самой системе аутентификации весьма высоки: необходима исключительно высокая чувствительность биометрической системы, чтобы аутентифицировать пользователя, но исключить аутентификацию злоумышленника со схожими биометрическими параметрами или использующего специальные приемы обхода биометрической защиты (например, подделка отпечатков пальцев при помощи муляжей).

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		9

1.2. Политики безопасности

Наиболее распространенным фактором аутентификации в современных информационных системах является пара логин–пароль. Администратором информационной системы могут использоваться различные компоненты политик безопасности паролей, накладывающие ограничения на те или иные их показатели.

Типичными компонентами политик безопасности являются:

1) Требования к длине и структуре пароля

Множество политик безопасности имеют ограничение на минимальную длину пароля. В большинстве случаев пароль должен содержать 8 и более символов, однако в некоторых случаях этого также недостаточно. Более длинные пароли в целом могут быть более безопасными, но некоторые системы могут иметь ограничение и на максимальную длину пароля для обеспечения обратной совместимости.

Некоторые политики безопасности предлагают или требуют от пользователя определенный тип пароля:

- использовать символы в верхнем и нижнем регистрах (зависимость от регистра);
- использовать одну или несколько цифр;
- использовать специальные символы, такие как \$, #, @ и т.п.;
- запрещают использовать паролей из черного списка;
- запрещают использовать информацию из личных данных пользователя;
- запрещают использовать название компании и ее аббревиатуру;
- запрещают использовать пароли, имеющие формат даты, номеров автомобилей, телефонов и т.п.

Некоторые системы устанавливают пароли пользователям самостоятельно или предлагают выбрать один из предварительно сформированного списка.

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		10

2) Черные списки паролей

Черные списки паролей содержат сочетания символов, запрещенные к использованию. Они также могут содержать пароли, удовлетворяющие требованиям политики безопасности, но использование которых небезопасно по тем или иным причинам. Типичными примерами являются «Password0», «Qwerty123», «P@ssw0rd».

3) Период действия пароля

Некоторые политики требуют у пользователя периодичной смены пароля, зачастую — один раз в 90 или 180 дней. В общем случае, системы проверяют несколько последних паролей пользователя на предмет их соответствия.

Также необходимо учитывать человеческий фактор: человек, в отличие от компьютера, не может удалить ранее запомненную информацию и заменить ее чем-то другим. Пользователи часто отдают предпочтение легко запоминаемым, но менее стойким паролям, или используют мнемонические техники для построения сложных паролей на основе шаблонов и словаря. Однако даже такой подход не срабатывает при частой смене паролей: высока вероятность того, что пользователь будет устанавливать пароли вида «password1», «password2», и т.п.

В некоторых информационных системах могут существовать ограничения на пароли, накладываемые поддержкой обратной совместимости программного или аппаратного обеспечения. Например, на длину или на алфавит пароля.

Игнорирование политик безопасности при использовании парольной защиты может привести к увеличению вероятности неавторизованного доступа к информационной системе, особенно в случае с использованием долговременных паролей.

Принудительное использование сложных паролей увеличивает вероятность того, что пользователь просто запишет пароль на какой-либо носитель в открытом виде (например, на лист бумаги или в текстовый файл), что существенно увеличивает риск обнаружения пароля злоумышленниками.

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		11

1.3. Обзор аналогов

Существующие средства хранения аутентификационной информации (менеджеры паролей) могут быть классифицированы по способу хранения и защиты данных, а также по возможности работы под управлением различных операционных систем. Наиболее распространенные из них рассмотрены ниже.

1) KeePassX

KeePassX [1] является одним из наиболее популярных приложений для хранения паролей с использованием мастер-ключа для шифрования базы данных. Имеет реализации для всех основных платформ (включая мобильные).

Его основными достоинствами является использование алгоритмов шифрования AES и Salsa, а также механизма выработки паролей Argon2 [2]. Способен генерировать надежные случайные пароли с заданными длиной и алфавитом. Хранит не только пароли, но и вспомогательные данные (в том числе файлы, закодированные в base64). Использует API безопасности операционной системы (Windows, Linux). Имеет открытый исходный код.

В качестве основных недостатков стоит выделить необходимость переносить базу данных паролей между устройствами, на которых она будет использоваться (в том числе, необходима синхронизация правок в базе данных); этот недостаток может быть решен использованием облачных сервисов хранения данных (например, DropBox, Google Drive, Yandex Disk).

2) LastPass

LastPass [3] позиционирует себя как менеджер паролей с облачным хранилищем. Основным достоинством является доступность на любом устройстве, имеющем подключение к Интернету и соответствующее приложение или веб-браузер. Синхронизация изменений базы паролей автоматическая.

Недостатки: закрытый исходный код, необходимость доверия третьей стороне, необходимость иметь соединение с Интернетом.

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		12

Также в качестве недостатка такого подхода к хранению паролей можно выделить события февраля 2017 года, именуемые Cloudbleed [4]. Ошибка в системе безопасности сервиса Cloudflare привела к утечке данных из памяти сервера, включая персональные данные (HTTP куки, авторизационные токены, POST запросы и другую закрытую информацию), часть которых была закеширована поисковыми системами. Общая схема уязвимости показана на рисунке 1.

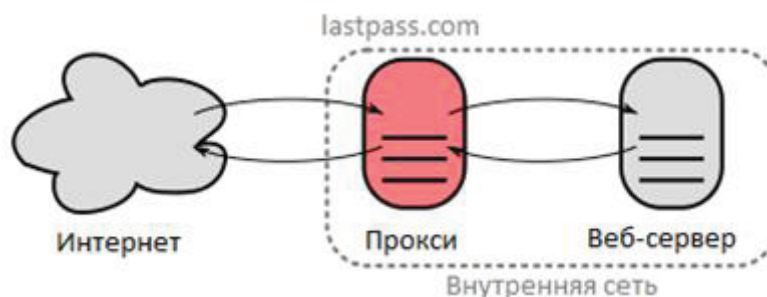


Рисунок 1 — Схема уязвимости Cloudbleed

3) RoboForm2Go

RoboForm2Go [5] — это программно-аппаратная реализация менеджера паролей, специализирующаяся на интеграции в браузеры. В качестве хранилища используется подготовленный USB накопитель, доступ к которому осуществляется посредством специализированной программы-клиента; такой подход позволяет «создавать» переносимые токены с данными для аутентификации, однако для их использования необходимо иметь предустановленное клиентское ПО и один из поддерживаемых браузеров.

1.4. Предложения по проекту

В данной работе под аппаратными средствами аутентификации подразумевается одно или несколько устройств, обеспечивающих автоматизацию одного из этапов аутентификации (далее — устройство). Разрабатываемое устройство позволит устранить некоторые недостатки вышеописанных сервисов:

1) Необходимость наличия хранилища паролей — отсутствует, т.к. генерация выполняется на основе некоторого набора данных, введенных пользователем для идентификации информационной системы, при этом сохранение самих паролей на устройстве не требуется.

2) Независимость от платформы — использование спецификации USB-HID (Universal serial bus — human interface device) позволяет подключаться к любому устройству, имеющему шину USB и соответствующий системный драйвер (если политика безопасности системы это позволяет) и осуществлять вывод информации с помощью эмуляции клавиатуры; альтернативно — вывод информации может быть осуществлен другим способом (например, через терминал или на экран).

3) Аппаратная реализация — операционная система и ПО, работающее в ее среде, не имеет доступа к ПО устройства и к данным, являющимися исходными для генерации паролей; дополнительно, исходные данные могут храниться на устройстве в зашифрованном или замаскированном виде.

4) Не требуется подключение к Интернету, а также установка сторонних приложений (за исключением драйвера USB-HID во встраиваемых системах и клиента telnet для инициализации устройства).

Несмотря на перечисленные преимущества, устройство вносит функциональное ограничение: невозможность хранить заранее определенные пользователем пароли или коды доступа.

Сравнение базовых характеристик приведено в таблице 1.

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		14

Таблица 1. Функции аналогов в сравнении с разрабатываемым устройством

Критерий	Решения			
	Keepass	LastPass	RoboForm2Go	PWDG
Хранилище данных	Шифрованная БД	БД в облаке	Съемный USB носитель	Съемный USB носитель
Базовый алгоритм	Argon2	неизвестно	неизвестно	KMAC
Дополнительное ПО	специальное приложение	веб-браузер или специальное приложение	специальное приложение, браузер	нет
Использование для авторизации в ОС	нет	нет	нет	да
Быстрая смена и восстановление пароля	да, за счет хранения в БД	да, за счет хранения в БД	нет	да, динамическая генерация

2. ПРОЕКТИРОВАНИЕ

2.1. Функциональные требования

Разрабатываемое устройство предназначено для упрощения и автоматизации процедуры аутентификации пользователя в информационных системах, использующих в качестве фактора аутентификации пару логин–пароль или персональный идентификационный номер (ПИН).

Устройство выполняет следующие базовые функции:

- формирование логина, пароля и ПИН-кода для заданной информационной системы и вывод его на экран устройства или, посредством USB, в компьютер;
- хранение и вывод списка обслуживаемых информационных систем;
- ввод и обработка мастер-ключа пользователя, применяемого при формировании данных для аутентификации;
- смена логина, пароля и ПИН-кода по запросу пользователя.

При реализации указанных функций управление устройством должно осуществляться посредством текстового или графического интерфейса пользователя.

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		16

2.2. Структурная схема устройства

Для реализации вышеописанных требований необходимо использовать микроконтроллер, имеющий аппаратную или программно-совместимую поддержку USB, обладающий достаточным объемом постоянной запоминающей памяти и большим количеством контактов для подключения периферийных устройств (микросхем, дисплея, кнопок и т.п.).

Структурная схема представлена на рисунке 2 и в приложении А.

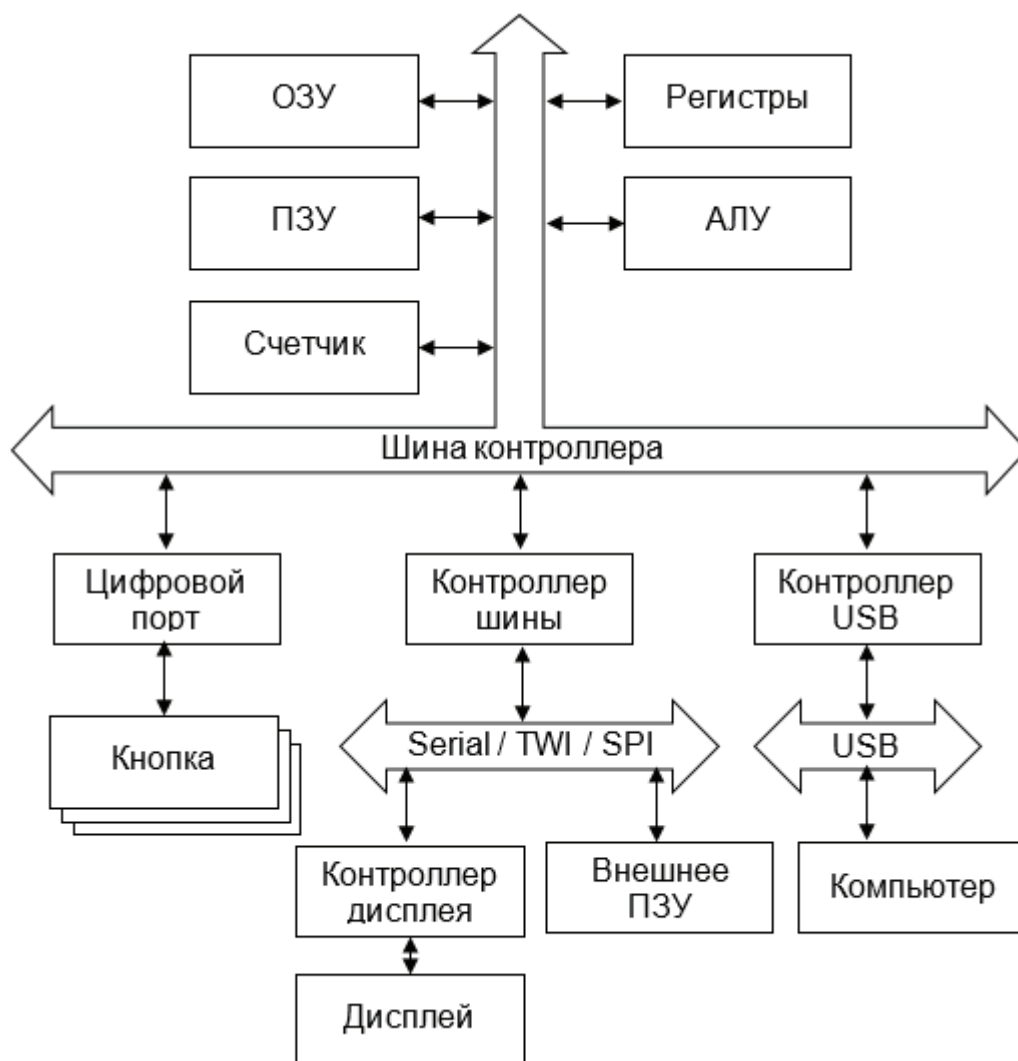


Рисунок 2 — Структурная схема устройства

При недостаточном объеме внутренней памяти микроконтроллера или при её низком ресурсе перезаписи хранилище данных следует перенести на внешнюю микросхему памяти (например, EEPROM или Flash).

2.3. Интерфейсы взаимодействия микросхем

Для обеспечения взаимодействия микроконтроллера с периферийными устройствами необходимо организовать шину обмена данных. Большинство микроконтроллеров имеют аппаратную поддержку или возможность для программной поддержки асинхронного последовательного (Serial Interface), последовательного серийного (Serial Peripheral Interface) и интерфейса TWI/I2C (Two-Wire Interface, Inter-Integrated Circuit). Выбор интерфейса взаимодействия накладывает некоторые ограничения на выбор элементной базы, а также влияет на сложность печатной платы и скорость работы устройства в целом.

1) Асинхронный последовательный интерфейс

Для организации канала передачи с использованием асинхронного последовательного интерфейса [6] необходимо 2 шины данных: RX — приемник и TX — передатчик.

Из-за того, что интерфейс является асинхронным, линия для отсчета тактовой частоты отсутствует, и, следовательно, согласование частоты передачи должно осуществляться еще до начала передачи данных. Сопрягаемые устройства должны иметь тактовые генераторы с одинаковыми частотами для обеспечения корректной обработки данных.

Теоретическое ограничение скорости обмена данными отсутствует, но большинство устройств поддерживают обмен на фиксированном наборе скоростей, максимальная из которых — 230400 бит/с.

Асинхронные последовательные порты кроме данных передают служебную информацию для организации обмена — старт-стопные биты, что может значительно снизить скорость обмена информацией при малых размерах передаваемого пакета.

Также особенностью интерфейса является то, что он предназначен, в основном, для организации обмена между 2 устройствами, поскольку в противном случае возникает проблема разражения коллизий на шинах данных.

К тому же, интерфейс не предоставляет возможности для адресации устройств при таком сценарии работы.

Общая схема организации асинхронного последовательного интерфейса представлена на рисунке 3.

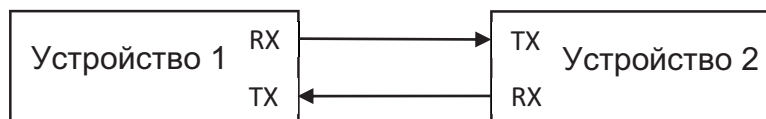


Рисунок 3 — Асинхронный последовательный интерфейс

2) Последовательный периферийный интерфейс

Для организации канала передачи с использованием последовательного периферийного интерфейса [7] необходимо как минимум 4 шины: SCK — тактовая частота, MOSI — выход ведущего, вход ведомого, MISO — выход ведомого, вход ведущего и SS — выбор ведомого.

Последовательный периферийный интерфейс позволяет осуществлять сопряжение нескольких устройств по общим шинам передачи данных и шине тактовой частоты, что избавляет этот интерфейс от недостатков асинхронного интерфейса. Проблема коллизий на шинах данных решается с помощью выделенной для каждого ведомого устройства шины выбора чипа: устройство будет активно только при наличии соответствующего сигнала на его шине SS.

Существенным преимуществом последовательного периферийного интерфейса является простота его аппаратной реализации (требуется несколько сдвиговых регистров), благодаря чему он поддерживается множеством различных микроконтроллеров. Программная поддержка интерфейса также не является сложной задачей: требуется простая проверка значений регистров и установка их состояния для передачи следующего обмена данными.

Главный недостаток — линейный рост количества необходимых шин выбора чипа: для каждого нового ведомого устройства необходимо использовать отдельную шину, что усложняет проектирование печатной платы. Этот недостаток может быть частично устранен за счет использования

мультиплексора, которым ведущее устройство будет активировать ведомые, однако это также усложняет печатную плату и схему устройства в целом.

Общая схема организации последовательного периферийного интерфейса представлена на рисунке 4.

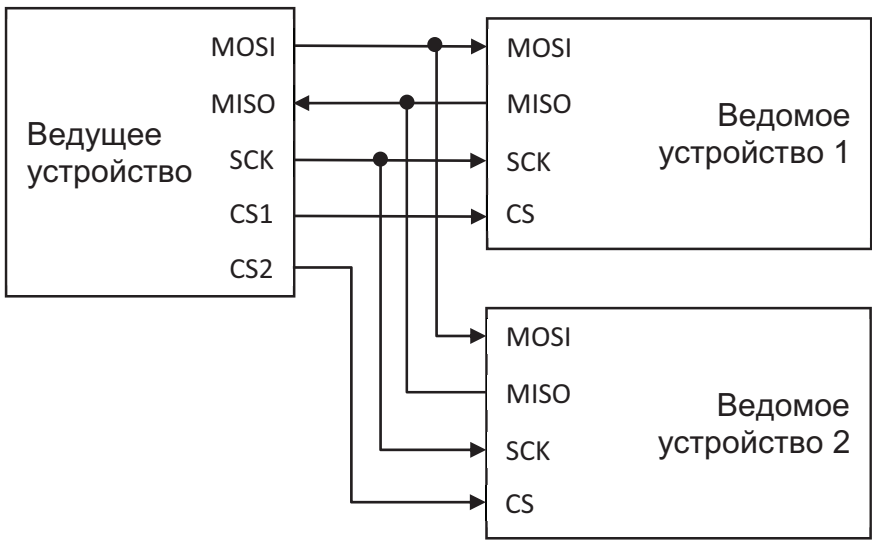


Рисунок 4 — Последовательный периферийный интерфейс

3) Интерфейс TWI

Интерфейс TWI [8] позволяет нескольким ведомым микросхемам в составе одного или различных устройств взаимодействовать с одной или более ведущими микросхемами. Для организации канала передачи необходимо 2 шины: SCL — тактовая частота и SDA — данные. Определение активной ведомой микросхемы осуществляется с помощью адреса, передаваемого по шине данных.

Схема организации интерфейса TWI представлена на рисунке 5.

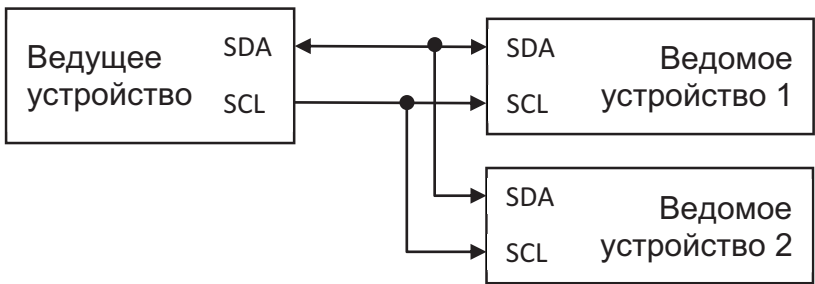


Рисунок 5 — Интерфейс TWI

При передаче данных ведущее устройство формирует пакет, содержащий вид операции (чтение или запись), 7-битный адрес ведомого устройства и собственно передаваемые данные. Каждый байт, передаваемый по шине SDA, подтверждается получающим устройством (ведущим или ведомым) 1 битом подтверждения, что увеличивает надежность передачи, но снижает скорость.

Ведомое устройство принимает данные от ведущего только при обнаружении своего адреса в первом байте переданных данных — это позволяет подключать несколько ведомых устройств на одну пару шин.

Отдельной особенностью TWI является возможность ведомого устройства удлинять такт в случае длительной обработки данных: с аппаратной точки зрения это достигается заземлением шины SCL внутри ведомого устройства.

Несмотря на сложность программной реализации в данном проекте предпочтительно использовать именно этот интерфейс, поскольку в таком случае упрощается разработка принципиальной схемы и печатной платы, а в качестве программной реализации протокола может быть использована одна из ее свободно распространяемых реализаций.

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		21

2.4. Основного алгоритм программы

Для реализации основной функциональности устройства необходимо применить один из алгоритмов генерации псевдослучайной последовательности, формирующий свое внутреннее состояние на основе произвольного массива данных и способного восстановить это же состояние при вводе такой же последовательности входных данных. Для этой цели рассматривались специализированные функции выработки паролей: SHA2-HMAC [9], Scrypt [10], Argon2, а также базовый криптопримитив, лежащий в основе функции хеширования SHA3 — Кессак [11].

Особенности платформы, на которой разрабатывается устройство, накладывают серьезные ограничения на возможность использования некоторых алгоритмов. Так, например, реализовать Scrypt на AVR или STM32 возможно, но только при использовании наименее безопасных модификаторов использования памяти (в таком случае минимальное потребление памяти для хранения внутреннего состояния составляет 1,5 КБ) или при использовании внешней микросхемы SRAM, которая предоставит достаточный объем быстрой памяти для хранения состояния, но существенно усложнит схему устройства и логику работы с памятью. Реализация Argon2 на AVR или STM32 является намного более сложной задачей, чем реализация Scrypt.

Реализация SHA2-HMAC не представляет никакой сложности, поскольку функция хеширования выполняет простые логические операции и поиск по таблицам подстановок, что не требует много памяти или вычислительных ресурсов. Однако для получения HMAC требуется выполнить множество промежуточных операций, увеличивающих стойкость функции. Данная функция может предоставить имитовставку и хеш, но для защиты данных большого размера требуется использовать какую-либо функцию блочного шифрования.

Криптопримитив Кессак — наиболее оптимальный вариант для данной работы, поскольку на его основе с минимальными затратами можно получить большой набор производных криптографических функций.

Кессак — это семейство функций губки [12], являющееся обобщением концепции криптографических функций с бесконечной длиной выходной последовательности.

В качестве основного преобразования в губке используется одно из семи преобразований Кессак- $f[b]$, где $b = 25, 50, 100, 200, 400, 800$ или 1600 (в порядке увеличения вычислительной сложности). Каждое преобразование состоит из простых функций раунда, похожих на преобразование в блочных алгоритмах шифрования, но без использования расписания ключей. Операции, выполняемые в функциях раунда, ограничиваются XOR, AND, NOT и битовым сдвигом. В функциях раунда не требуется поиска значений по таблицам, арифметических операций или сдвигов, зависящих от значения данных. Структура внутреннего состояния кессак представлена на рисунке 6.

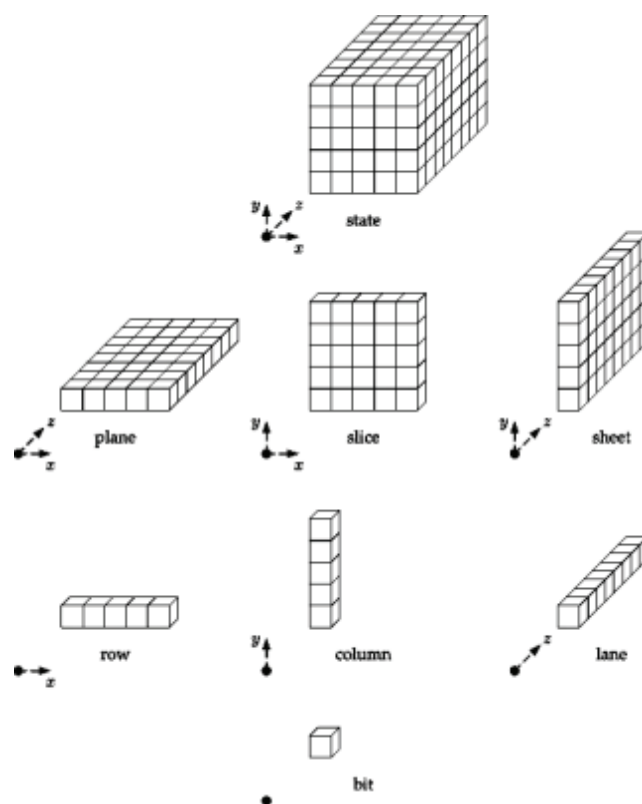


Рисунок 6 — Структура внутреннего состояния губки

Как функция губки (рисунок 7), Кессак имеет переменную длину выходной последовательности, что позволяет упростить способы применения функции для задач, требующих хеш-функции с фиксированной длиной выходной последовательности. Функция в чистом виде может быть использована, например, для стандартного, доменного или случайного хеширования, поточного шифрования, вычисления имитовставки, а также для хеширования деревьев.

Как дуплексный объект, Кессак может быть использован для получения псевдослучайных последовательностей и аутентифицированного шифрования.

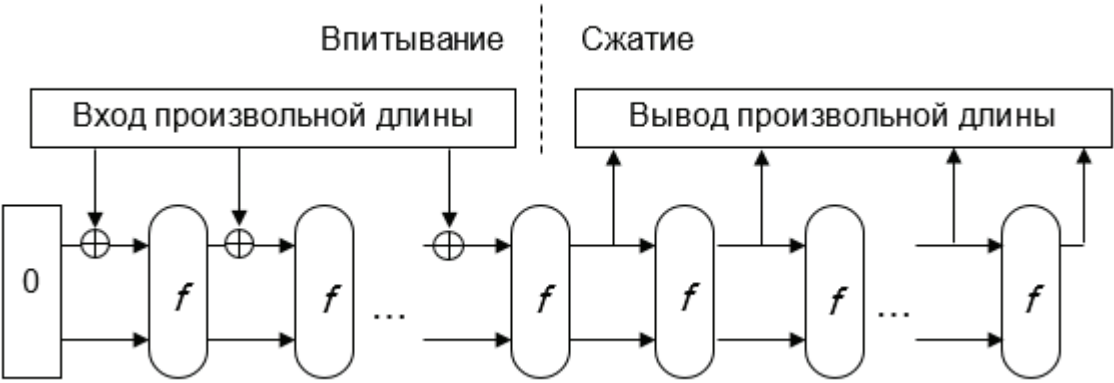


Рисунок 7 — Функция губки

Важной особенностью конструкции губки является легкость персонализации. Например, для получения симметричного ключа, имитовставки и случайной маски возможно использовать одну и ту же входную последовательность данных (например, пароль), но при добавлении на вход функции персонифицирующих данных получаемые выходные последовательности будут различны.

Способы применения функции губки:

1) Имитовставка

Добавление секретного ключа на вход хеш-функции Кессак позволяет получить в имитовставку (код аутентификации сообщений, MAC). Сценарий использования функции губки представлен на рисунке 8.

Для получения имитовставки на основе хеш-функций SHA-1 или SHA-2 требовалось использовать дополнительное преобразование HMAC, поскольку хеш-функции не обеспечивали достаточного уровня криптографической стойкости.

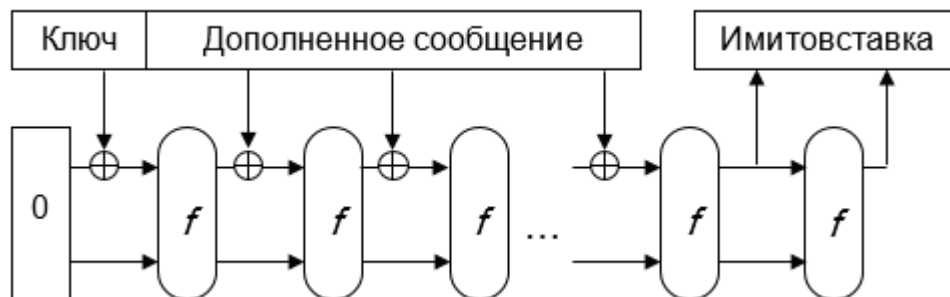


Рисунок 8 — Использование губки для получения имитовставки

2) Поточный шифр

Добавление секретного ключа с вектором инициализации позволяет получить поточный шифр, поскольку функция губки способна генерировать данные произвольной длины. Сценарий использования функции губки представлен на рисунке 9.

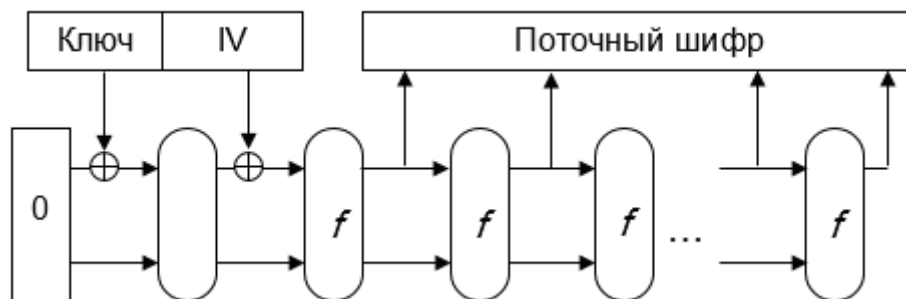


Рисунок 9 — Использование губки для поточного шифра

3) Поточный шифр с произвольным доступом

Добавление на вход ключа, вектора инициализации и номера блока позволяет получить поточный шифр с произвольным доступом — аналог блочного шифра в режиме счётчика (например, AES-CTR). Данная реализация может быть использована в совокупности с имитовставкой для шифрования пакетов данных или сообщений, передаваемых по сети. Сценарий использования функции губки представлен на рисунке 10.

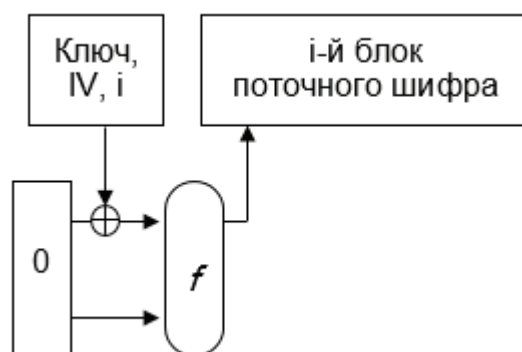


Рисунок 10 — Использование губки для поточного шифра с произвольным доступом

4) Генерация масок и ключей

Использование входа и вывода произвольной длины может применяться для генерации симметричных ключей из паролей, связывания сообщения с электронной подписью, получения случайных данных из шумов. Если периодически сбрасывать состояние конструкции губки, то возможно получить полноценный генератор псевдослучайных чисел. Сценарий использования функции губки представлен на рисунке 11.

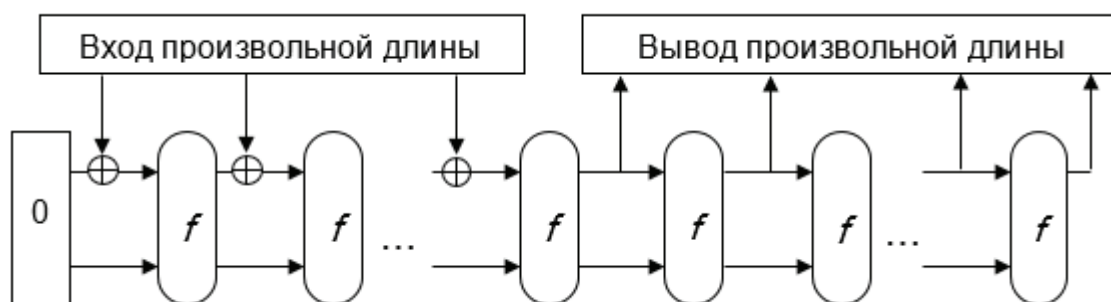


Рисунок 11 — Использование губки для генерации масок и ключей

Производные от Кессак алгоритмы стандартизованы в виде функций с переменной длиной выходной последовательности SHAKE128 и SHAKE256, а также хеш-функций от SHA3-224 до SHA3-512 [13]. Параметры губки, стандартизированные Национальным институтом стандартов и технологий (NIST), представлены в таблице 2.

Таблица 2. Стандартизированные параметры конструкции губки

Алгоритм	r	c	d	Длина вывода (бит)
SHAKE128	1344	256	0x1F	не ограничено
SHAKE256	1088	512	0x1F	не ограничено
SHA3-224	1152	448	0x06	224
SHA3-256	1088	512	0x06	256
SHA3-384	832	768	0x06	384
SHA3-512	576	1024	0x06	512

В качестве базового примитива в работе будет использовано преобразование Кессак-f[1600], а для решения прикладных задач будут использоваться функции на его основе: КМАС (имитовставка), SHA3-256 (хеш) и Кессак-256 (псевдослучайная последовательность).

2.5. Элементная база устройства

2.5.1. Микроконтроллер

Atmega32U4 — восьмибитный AVR микроконтроллер с расширенной RISC архитектурой и большим набором поддерживаемых протоколов взаимодействия микросхем. Имеет 32 регистра, 32 КБ Flash памяти для программы, 2,5 КБ SRAM (ОЗУ), 1КБ EEPROM (ПЗУ).

Основным критерием выбора данного микроконтроллера является аппаратный драйвер USB, позволяющий реализовывать легковесные программы для взаимодействия с компьютером. В отличие от Atmega16U4, у данного контроллера имеется больший объем всех видов памяти (Flash, SRAM и EEPROM).

Кроме аппаратной поддержки USB контроллер предоставляет широкие возможности для подключения периферийных устройств, а также использования внутренних компонентов:

- 8-битный, 16-битные таймеры;
- высокопроизводительный 10-битный таймер;
- watchdog таймер;
- 8-битные каналы с широтно-импульсной модуляцией (ШИМ, PWM);
- 10-битный аналогово-цифровой преобразователь (АЦП, ADC);
- аппаратная поддержка интерфейсов USART, SPI, TWI;
- прерывания.

Для защиты программного обеспечения имеется возможность блокировки загрузчика.

Рабочее напряжение: $V_{CC}=2.7...5.5V$, максимальный потребляемый ток: не более 50 mA; диапазон рабочих температур: $-40...85\text{ }^{\circ}C$.

Для реализации всех функциональных требований данной работы использование всех возможностей контроллера не требуется, поэтому

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		28

некоторые его компоненты (например, АЦП) могут быть отключены. Подобное решение не только уменьшит потребляемую мощность, но и снизит вероятность появления помех внутри контроллера.

Для взаимодействия с периферийными устройствами возможно использование любого сочетания протоколов (SPI/TWI/USART) или только одного из них. Неиспользуемые аппаратные драйверы протоколов также могут быть отключены.

Поскольку устройство будет взаимодействовать с компьютером по USB, важна стабильность тактового генератора микроконтроллера. Несмотря на то, что контроллер имеет внутренний тактовый генератор, использование внешнего источника колебаний является обязательным. Для данного устройства желательно использовать 16 МГц кварцевый резонатор.

Для обеспечения электрической совместимости на шине USB (D+ и D–), а также на линиях питания рекомендуется использовать дополнительные компоненты: резисторы для линий данных и компоненты защиты от статического напряжения для линий питания, однако данных мер будет недостаточно при умышленном создании короткого замыкания на линии питания или подачи высокого напряжения на линии данных.

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		29

2.5.2. Микросхема постоянной памяти

AT24C256 — микросхема электрически стираемой перепрограммируемой постоянной памяти (EEPROM), использующая интерфейс TWI. Она обеспечивает 262144 бит памяти, организованных в 32768 8-битных слов (32 КБ). На одной шине может быть расположено до 4 микросхем памяти с различными адресами устройств (устанавливаются подачей постоянного сигнала на соответствующие контакты микросхем).

Доступ к ячейкам памяти: случайный или постраничный (64 байт). В случае случайного доступа требуется 15-битная адресация (2 байта). Максимальное время записи ячейки — 5 мс.

Рабочее напряжение: $V_{CC}=2.7...5.5V$ (стандартный режим) или $1.8...3.6V$ (низковольтный режим); максимальный потребляемый ток: не более 5 мА. Диапазон рабочих температур: $-55...125\text{ }^{\circ}C$.

Использование внешней микросхемы памяти расширяет возможности хранилища устройства: микроконтроллер имеет только 1 КБ EEPROM памяти, чего будет достаточно только тестовых целей или простейшего сценария работы. Использование внешней памяти значительно расширяет возможности устройства, при этом снижая нагрузку на внутреннюю память контроллера (она может использоваться для других целей), но при этом снижает общую производительность системы за счет времени обработки запросов контроллером памяти, а также затратами на реализацию протокола взаимодействия.

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		30

2.5.3. Модуль дисплея

SSD1306 — микроконтроллер OLED дисплеев с поддержкой протоколов TWI, SPI, а также внутренним преобразователем постоянного напряжения (DC/DC) для питания матрицы дисплея.

На основе данного контроллера выпускается множество дисплеев с различными размерами матрицы: 96x64, 128x32, 128x64 и другие, при этом основной способ взаимодействия с матрицей остается тем же. Аппаратные модули дисплея могут быть сконфигурированы для поддержки разных интерфейсов взаимодействия (SPI/TWI), что определяется комбинацией сигналов на 3 выделенных контактах (если предусмотрено производителем) или могут иметь поддержку только одного фиксированного протокола. Это также справедливо для конфигурации схем питания матрицы дисплея: для питания может использоваться внутренний преобразователь напряжения (требуется использование дополнительных конденсаторов и резисторов для его работы) или внешний источник 12 В.

Одним из существенных недостатков применения модуля дисплея на основе данного контроллера является большое количество вариантов исполнения от различных производителей в сочетании с отсутствием полной документации на конкретный модуль в целом, исключая документацию на контроллер. Этот недостаток преодолевается поиском аналогичных решений от различных производителей, а также использованием методов обратного проектирования готовых схем.

Несмотря на отсутствие полноценной документации, модули доступны с различными вариантами протоколов (SPI/TWI), логических уровней (3.3/5В) и схем питания матрицы. Одним из существенным преимуществом данного модуля является доступность на рынке плат разработчика для всех вышеперечисленных вариантов исполнения.

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		31

3. РЕАЛИЗАЦИЯ

3.1. Разработка схемы электрической принципиальной

Для разработки принципиальной схемы и печатной платы использовался пакет приложений DipTrace [14]. Для поддержки используемых компонентов и микросхем была создана библиотека элементов.

Принципиальная схема составлена в соответствии с рекомендациями по электрической совместимости компонентов, описанными в документации к микроконтроллерам Atmega32U4 и SSD1306.

В принципиальной схеме используется минимальное количество вспомогательных компонентов, однако при этом обеспечивается стабильное взаимодействие компонентов устройства между собой и микроконтроллера с компьютером. Для дополнительной защиты от статического напряжения, являющегося одной из причин выхода из строя USB устройств, используются компоненты защиты от статического напряжения PGB1010603.

Для согласования логических уровней напряжения контроллеров Atmega32U4 и SSD1306 используются схемы с транзисторами. Для некоторых исполнений модуля дисплея использование данного подхода не требуется, поскольку схема преобразования логических уровней может быть реализована внутри модуля (уточняется по документации).

Каждой микросхеме в устройстве в соответствие поставлен конденсатор для устранения помех по линии питания.

Принципиальная схема представлена в приложении Б.

При подаче напряжения микроконтроллер Atmega32U4 и контроллер дисплея SSD1306 выполняют сброс и начальную инициализацию независимо друг от друга. С небольшой задержкой Atmega32 начинает выполнение программы, размещенной в Flash памяти: на этом этапе микроконтроллер устанавливает режим работы контактов, настраивает аппаратные драйверы

интерфейсов (активирует TWI) и отправляет команды на инициализацию специальных протоколов обмена на основе TWI контроллеру дисплея и микросхеме памяти. Для SSD1306 необходима специальная последовательность команд, устанавливающих режим питания и начальное состояние матрицы (в противном случае матрица будет отображать случайные данные, шум). Для микросхемы памяти особой процедуры инициализации не требуется.

После инициализации периферийных устройств микроконтроллер ожидает ввода от пользователя. При определенной последовательности управляющих команд, введенных кнопками устройства, контроллер сформирует и отправит данные по протоколу USB-HID на компьютер, к которому он подключен.

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		33

3.2. Разработка программы для микроконтроллера

Программа для микроконтроллера содержит несколько модулей, оформленных в виде классов. Модули являются взаимозависимыми и взаимодействуют посредством вызова методов или прямого доступа к членам класса.

Упрощенная схема взаимодействия классов и базовых функций представлена на рисунке 12. Полная диаграмма классов представлена в приложении В.

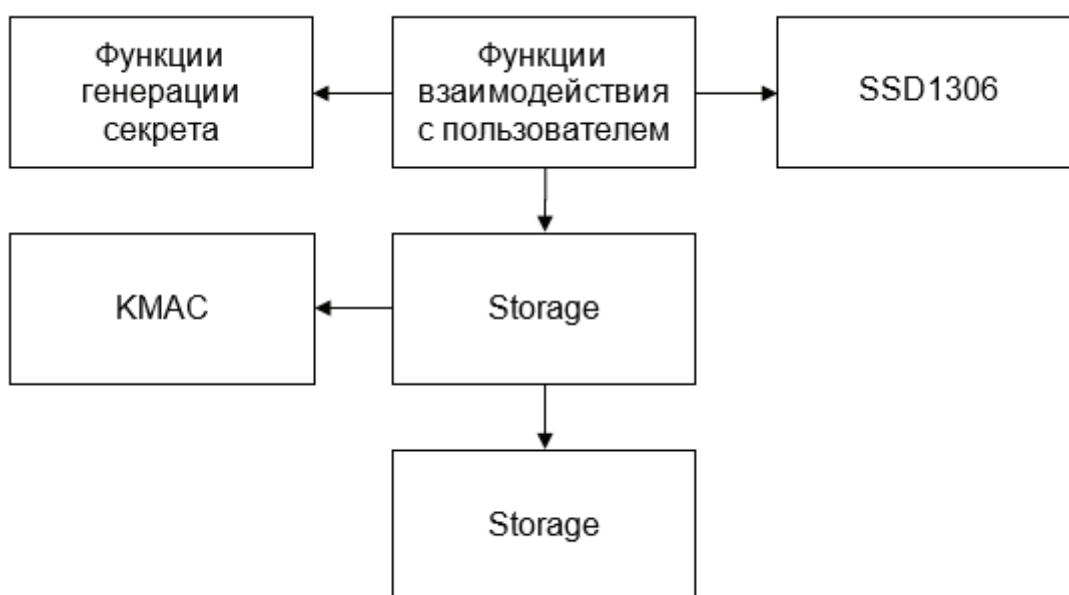


Рисунок 12 — Схема взаимодействия классов

При подаче питания на контроллер выполнение начинается не собственно программой, а загрузчиком LUFA. Библиотека загрузчика обеспечивает поддержку USB для микроконтроллеров AVR с аппаратной реализацией этого протокола. По завершению инициализации загрузчик передает управление основной программе микроконтроллера. Исходный код программы представлен в приложении Г.

3.2.1. Класс криптографии

Класс криптографии реализует, собственно, функцию губки, а также ее вариант для подсчета имитовставки с заданным ключом.

В качестве основы класса взята открытая опорная реализация функции преобразования Кессак (кессакр) для платформы AVR на языке Си [15]. Существует относительно оптимальная реализация на AVR-ASM, однако для простоты поддержки кода была взята указанная выше реализация.

Кроме базового преобразования в класс добавлены методы формирования и предварительной обработки внутреннего состояния, вызываемые до основного преобразования (поглощение, выравнивание блока, сжатие). Кроме того, класс включает функцию маскирования с использованием состояния губки, на основе которого строится защита хранимых в микросхеме EEPROM данных.

Объект класса может быть единственным в программе, поскольку предусмотрена функция гарантированного сброса внутреннего состояния губки.

На основе базового класса Кессак в соответствии с расширением к стандарту [16] построен класс подсчета имитовставки КМАС. Он предоставляет доступ ко всем методам базового класса и дополняет их функциями для подсчета имитовставки. Внутреннее состояние губки для этих классов общее, поэтому при использовании методов не следует смешивать вызовы Кессак и КМАС, т.к. это приведет к получению некорректного результата. Разнородные вызовы методов следует разделять методом сброса внутреннего состояния.

Функции, производящие выходные данные, способны производить последовательности переменной длины — эту особенность можно использовать для формирования частичных контрольных сумм для проверки целостности данных.

3.2.2. Функции взаимодействия с пользователем

Класс взаимодействия с пользователем обеспечивает логику вывода и обработки иерархического меню, с помощью которого пользователь получает доступ к базовым функциям устройства и получает ответ в виде сообщений, выводимых на дисплей. Также класс осуществляет маршрутизацию команд между классами.

Иерархическое меню представляет собой простое, но очень гибкое решение на основе структур элементов с указанием функций-обработчиков событий. Элементы меню хранятся в фиксированном массиве, определенном для текущей процедуры или метода, а каждый элемент содержит указатели на отображаемый при его активности текст, атрибут скрытности и указатель на функцию-обработчик.

Текст, отображаемый пользователю при активировании элемента меню, хранится в программной памяти (flash) и не переносится в стек, позволяя использовать его в криптографических операциях и при формировании входных данных для них. Данные в программной памяти имеют особый режим доступа к ним и, практически, недоступны для записи (крайне низкий ресурс перезаписи памяти).

Все функции-обработчики имеют универсальный интерфейс, обеспечивающий прием одного аргумента произвольного типа (void*) и возврат статуса операции. С помощью упаковки параметров в структуры данных возможна передача и получение любого количества входных и выходных параметров различного типа, в том числе и специализированных статусов операций.

Меню имеет иерархическую структуру, глубина которой ограничена только размерами стека. Переход на уровень выше в иерархии осуществляется с помощью особого возвращаемого статуса функции.

Структура меню в виде графа взаимодействия с пользователем представлена на рисунке 13 и в приложении Д.

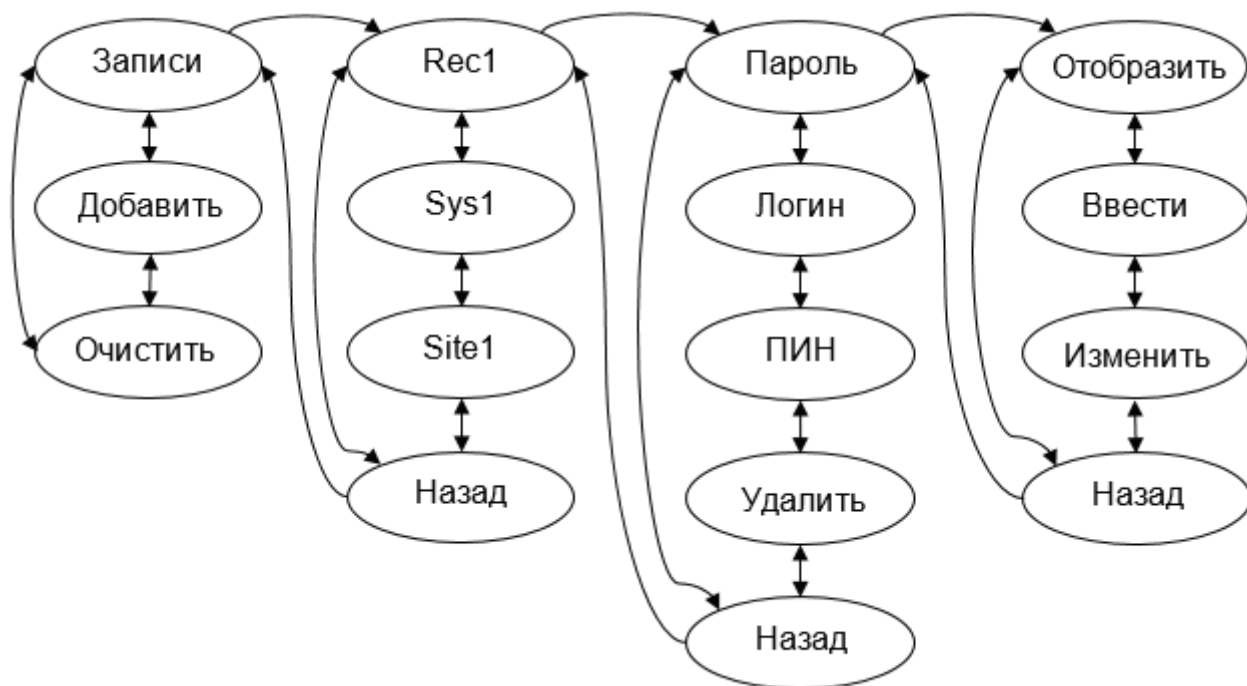


Рисунок 13 — Граф диалога с пользователем

Навигация по меню осуществляется с использованием аппаратных управляющих кнопок: «вверх», «вниз» и «выбор». Малое число кнопок упрощает конструкцию устройства и его печатную плату, но требует введения дополнительных элементов в меню для обеспечения возврата в родительские узлы. Переходы между элементами на текущем уровне вложенности осуществляется с помощью кнопок «вверх» и «вниз», выбор элемента (активация) — кнопкой «выбор».

Для выполнения некоторых действий пользователю необходимо осуществить ввод данных в специальном формате или текстовых данных.

Поскольку с использованием всего 3-х кнопок ввод текста представляет сложность, такой тип данных вводится с использованием терминала компьютера. Для этой цели могут использоваться программы для ОС Windows HyperTerminal или putty. Процесс ввода данных инициируется устройством, при этом при начале обмена на экран и в терминал будет направлено одинаковое

текстовое сообщение с указанием необходимой информации. Данные вводятся ASCII-строками, признаком окончания передачи является символ конца строки.

Под данными в специальном формате подразумеваются числа, вводимые на устройстве, а также последовательность разблокировки. Числа вводятся при помощи диалога-итератора, в котором кнопками «вверх» и «вниз» осуществляется увеличение или уменьшение числа, а кнопкой «выбор» осуществляется ввод текущего отображенного значения. Последовательность разблокировки вводится также кнопками «вверх» и «вниз», но при этом на экране отображается соответствующий символ; завершение ввода происходит или при нажатии кнопки «выбор» или при достижении предела длины последовательности.

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		38

3.2.3. Класс взаимодействия с графическим дисплеем

Взаимодействие с аппаратным модулем дисплея осуществляется по протоколу, определенному спецификацией контроллера SSD1306 [17], и реализуемому посредством шины TWI.

Графические данные, прежде чем поступить в контроллер дисплея, подготавливаются в специальном видеобуфере, реализуемом программным способом. Логика работы с контроллером дисплея реализована по аналогии с графическим драйвером с открытым исходным кодом, предоставляемого компанией Adafruit [18].

Поскольку экран является графическим, а не символьным, текстовые данные необходимо предварительно преобразовать в растровую форму с использованием шрифта (например, CP437) и поместить в видеобuffer. Вывод графической информации (точек, линий, двумерных фигур) осуществляется тем же способом. При вызове метода синхронизации данные из видеобuffer передаются контроллеру дисплея и отображаются на матрице. Поскольку содержимое видеобuffer не синхронизируется с фактическим буфером данных контроллера дисплея в реальном времени, возможна предварительная подготовка и преобразование данных в памяти контроллера, что снижает нагрузку на шину TWI и обеспечивает практически незаметную смену изображения на дисплее.

Видеобuffer представляет собой непрерывный массив байт размерности, соответствующей размеру матрицы дисплея (128x32 пикселя), доступ к которому осуществляется по битовым смещениям. Для ускорения доступа к данным массива некоторым переменным установлены директивы принудительного использования байтовых регистров контроллера (переменные-итераторы). Доступ к каждому отдельному пикселю не влияет на состояние соседних пикселей.

Необходимость использования видеобуфера обусловлена тем, что несмотря на наличие аппаратного видеобуфера в контроллере SSD1306 доступ к его данным посредством протокола управления дисплеем невозможен (это не зависит от используемой шины обмена данными). Этот факт накладывает ограничения на программу микроконтроллера ATmega, поскольку видеобуфер использует 512 байт памяти независимо от необходимости изменять содержимое дисплея.

Для сокращения объема кода часть наиболее часто используемых команд была сгруппирована в функции-помощники, обеспечивающие вывод базовых примитивов — строк пользовательского меню — с фиксированными смещениями. На этапе отладки устройства эти функции могут быть переопределены для вывода данных посредством подключенного через usb2com терминала компьютера.

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		40

3.2.4. Класс взаимодействия с EEPROM

Класс взаимодействия с памятью осуществляет доступ к данным, хранящимся на внешней памяти, посредством шины TWI. Данный класс реализует побайтовое чтение и обновление данных, а также постраничный доступ.

Необходимость использования обновления вместо записи обусловлена необходимостью ограничить количество перезаписи ячеек памяти с целью увеличить срок ее службы (особенно актуально, т.к. записываются зашифрованные данные): прежде чем записать ячейку памяти выполняется сравнение текущего и нового значений ячейки, результат которой определяет необходимость перезаписи ячейки.

Постраничный доступ включает операции чтения и записи групп байтов, использующие аппаратные возможности микросхемы памяти: последовательное чтение и запись. Данная микросхема позволяет последовательно читать данные произвольной длины, используя одну команду, передаваемую через TWI, и внутренний счетчик, а также позволяет выполнять постраничную запись до 64 байт. Функция постраничной записи не реализуется в полной мере из-за использования логики обновления данных ячеек, но последовательное чтение значительно увеличивает скорость обмена данными при использовании внешней памяти. Поскольку большинство операций с памятью микросхемы EEPROM — это чтение, введение буфера для постраничной записи существенно увеличивает производительность устройства в целом.

Полезность функций оптимизации работы с памятью сильно зависит от уровня использования криптографии: при использовании маскирования данные становятся уязвимы к некоторым видам атак, но при изменении нескольких байт данных структура данных не будет перезаписываться полностью, что увеличит ресурс памяти.

3.2.5. Класс хранилища данных

Данные, обрабатываемые устройством, представлены в виде специальных структур и хранятся в зашифрованном виде во внешней памяти. Вместе с непосредственно данными хранится служебная информация и мастер-ключ пользователя, с помощью которого осуществляется доступ к записям.

В состав структуры записи входят следующие поля:

1. Контрольная сумма — часть хеша, вычисленного с помощью алгоритма Кессак, обеспечивающая контроль целостности и корректности данных после их расшифровки ключом пользователя.
2. Строка наименования информационной системы или ресурса (ограничена 255 байтами).
3. Счетчики текущих пароля, логина и PIN кода.

Все поля, за исключением контрольной суммы, маскируются с помощью псевдослучайной последовательности, производимой базовой функцией губки Кессак, состояние которой определяется ключом пользователя. Для усиления защиты против некоторых атак необходимо ввести зависимость формируемой маски от содержимого данных (или их хеша), однако это приведет к увеличению износа внутренней памяти EEPROM из-за частой перезаписи ячеек.

Доступ к структурам записей осуществляется с помощью служебной информации, записываемой в первые 512 байт памяти. Служебная информация включает в себя следующие поля:

1. Количество записей;
2. Маски доступа к записям;
3. Мастер-ключ пользователя и его контрольная сумма.

Доступ к записям осуществляется посредством индексов, преобразуемых в пары маска-подмаска (mask-submask). Общий принцип доступа к записям представлен на рисунке 14.

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		42

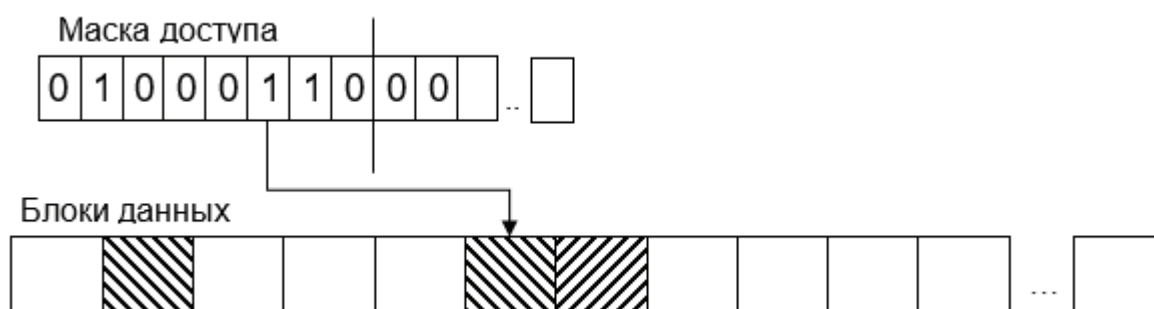


Рисунок 14 — Схема доступа к данным

При получении данных их памяти по рассчитанному смещению выполняется их декодирование с использованием мастер-ключа и проверка контрольной суммы. Такая схема позволяет исключить доступ к открытым данным с помощью посторонних устройств, а также проверяет их целостность при хранении.

Добавление записей осуществляется по первому доступному индексу. Удаление включает изменение поля масок доступа и перезапись соответствующих записи ячеек памяти.

3.2.6. Функции выработки секрета

Для выработки секрета (логина, пароля или ПИН-кода) используется заправка, формируемая с помощью мастер-ключа пользователя, а также контекста записи, для которой формируется секрет. Заправка формируется функцией КМАС и представляет собой байтовую последовательность фиксированного размера.

На основе заправки и типа секрета определяется используемая для генерации секрета символьная маска. Символьные маски определяют допустимые алфавиты символов, используемых при генерации секрета. Примером маски является строка «dddd», которая определяет правило формирования секрета: «Четыре символа из алфавита «цифры», или строка «vvvddd», определяющая правило «2 согласные буквы в малом регистре, 1 гласная в малом регистре и 2 цифры».

Алфавиты представляют собой подмножества печатных символов набора ASCII, разбитые по следующим признакам: гласные, согласные, буквы, цифры, буквы и цифры, специальные символы, все печатные символы ASCII; дополнительно эти подмножества разделяются по признаку регистра: верхний, нижний и смешанный.

В качестве результата функция формирует текстовую строку на основе заправки и символов соответствующего алфавита, которая используется в качестве секрета.

Реализация этих функций имеет несколько особенностей: ввиду ограниченности ресурсов памяти микроконтроллера, при формировании паролей используется набор фиксированных масок, составленных из подмножества всех доступных алфавитов (некоторые алфавиты объединены: например, буквенные разных регистров).

Общая блок-схема алгоритма генерации секрета представлена в приложении Е.

4. ОТЛАДКА И ТЕСТИРОВАНИЕ

4.1. Моделирование электрической схемы

Особенностью разрабатываемого устройства является то, что его составные части не являются простейшими электрическими компонентами, а представляют собой микросхемы различной степени интеграции, взаимодействующие с помощью программно-аппаратного протокола взаимодействия TWI. Эти факты серьезно осложняют процесс моделирования устройства с использованием электронно-вычислительных машин ввиду отсутствия соответствующего программного обеспечения.

Учитывая низкую рыночную стоимость составных компонентов, а также широкую доступность плат разработчика для микроконтроллера Atmega32U4 (например, платы Arduino, DfRobot, Adafruit и др.), моделирование работы устройства осуществлялось сначала на плате разработчика, а затем на прототипе устройства. Данный подход несет в себе риск повреждения составных компонентов устройства из-за неправильного монтажа схемы, а также риск сброса состояния микросхем (памяти и микроконтроллера) при ошибках согласования взаимодействия.

Проверка корректности конечной принципиальной схемы устройства осуществлялась посредством средств контроля целостности схем, встроенного в пакет DipTrace. В ходе проверки схемы ошибок не выявлено.

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		45

4.2. Создание прототипа устройства

Прототипирование выполнялось с использованием платы разработчика на основе микроконтроллера Atmega32U4, совместимой с платформой Arduino Leonardo. Совместимость с данной платформой позволяет использовать средства разработки и отладки, предоставляемые Arduino Studio, а также предоставляют доступ к библиотеке исходных кодов для программной реализации протоколов взаимодействия микросхем, а также шаблонов для проверки корректности подключения периферийных устройств (например, программа для отладки протокола TWI «i2c_scanner»).

Основным преимуществом среды Arduino Studio, в сравнении с Atmel Studio, является простота сборки и загрузки проекта в микроконтроллер, поскольку фазы компиляции, встраивания загрузчика и прошивки объединены в одну операцию, но при этом контролировать или изменять процесс нельзя.

Главным недостатком данного подхода является невозможность полноценной пошаговой отладки программы, поскольку среда разработки не умеет взаимодействовать с JTAG отладчиком микроконтроллера (этот недостаток сглаживается добавлением отладочной печати в серийный порт компьютера в контрольных точках программы).

Еще одним недостатком является увеличение объема кода за счет включения вспомогательных библиотек, обеспечивающих поддержку среды разработки.

Физический прототип построен на основе нескольких плат разработчика, размещенных на отладочной плате и соединенных между собой медными проводами. Для реализации интерфейса USB используется соответствующий кабель, подключенный к плате разработчика микроконтроллера. Внешний вид отладочной платы показан в приложении Ж.

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		46

4.3. Отладка и тестирование программы

Предварительное тестирование переходов состояний программы и криптографических алгоритмов производилось с помощью консольного Win32 приложения, созданного в среде Microsoft Visual Studio. Поскольку исходный код программы написан на языке C++, привязка к аппаратным возможностям платформы у большинства классов программы отсутствует (или может быть заменена альтернативным кодом, подключаемым с помощью директив компилятора).

Отладка основной функциональности программы на микроконтроллере осуществлялась посредством текстовых сообщений, передаваемых в терминал компьютера, а после подключения модуля дисплея — в виде коротких текстовых строк, характеризующих текущее состояние программы микроконтроллера.

Для разработанного устройства было проведено функциональное тестирование. Набор используемых функциональных тестов представлен в приложении 3.

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		47

ЗАКЛЮЧЕНИЕ

В ходе выполнения бакалаврской работы требования технического задания реализованы в полном объеме: средства автоматизации аутентификации в виде прототипа USB-HID совместимого устройства обеспечивает хранение списка информационных систем, генерацию и вывод соответствующих данных для аутентификации посредством графического дисплея или с помощью эмуляции клавиатуры.

Перспективой дальнейшей разработки является улучшение удобства существующих и добавление новых функций. В частности, функции инициализации устройства и управления списком хранимых информационных систем могут быть реализованы с помощью специального приложения с графическим интерфейсом в среде операционной системы (Windows, Linux и/или Android), что существенно упростит начальный этап использования устройства. Создание клиентского приложения потребует разработки дополнительных протоколов обмена данными между компьютером и устройством аутентификации (в лучшем случае — защищенного ассиметричной криптографией), а также использования библиотек для разработки кроссплатформенных приложений с графическим интерфейсом (QT, FLTK и т.п.).

Набор функций устройства может быть расширен за счет добавления опций для ограничения используемых алфавитов символов, динамического формирования масок, а также хранения пользовательских паролей — эти изменения требуют дополнительной оптимизации кода (в частности, динамического формирования видеобуфера), а также реализации более сложного диалога с пользователем при вводе данных информационной системы.

Возможности по дальнейшей оптимизации кода не ограничены.

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		48

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. KeePass Password Safe. – URL: <http://keepass.info> (дата обращения: 20.05.2017).
2. Argon2: the memory-hard function for password hashing and other applications / A. Biryukov, D. Dinu, D. Khovratovich – Luxembourg: University of Luxembourg, 2016. – URL: <https://password-hashing.net/argon2-specs.pdf> (дата обращения: 20.05.2017).
3. LastPass. – URL: <https://lastpass.com> (дата обращения: 20.05.2017).
4. Incident report on memory leak caused by Cloudflare parser bug. URL: <https://blog.cloudflare.com/incident-report-on-memory-leak-caused-by-cloudflare-parser-bug> (дата обращения: 21.05.2017).
5. RoboForm: World's Best Password Manager. – URL: <https://www.roboform.com> (дата обращения: 21.05.2017).
6. А.В. Белов. Микроконтроллеры AVR. От азов программирования до создания практических устройств. – М.: НиТ, 2012 – 544 с.
7. А.А. Лапин. Интерфейсы. Выбор и реализация – М.: Додэка-XXI, 2007. – 360с.
8. I2C-bus specification and user manual. Rev. 6. NXP Semiconductors. URL: www.nxp.com/documents/user_manual/UM10204.pdf (дата обращения: 21.05.2017).
9. RFC 6234 – US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF). – URL: <https://tools.ietf.org/html/rfc6234> (дата обращения: 19.05.2017).
10. RFC 7914 – The scrypt Password-Based Key Derivation Function. – URL: <https://tools.ietf.org/html/rfc7914> (дата обращения: 19.05.2017).
11. The Keccak sponge function family. URL: <http://keccak.noekeon.org> (дата обращения: 19.05.2017).

12. Sponge Functions / G. Bertoni, J. Daemen, M. Peeters, G. Van Assche. URL: <http://sponge.noekeon.org/SpongeFunctions.pdf> (дата обращения: 19.05.2017).

13. FIPS 202 – SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. – URL: <http://dx.doi.org/10.6028/NIST.FIPS.202> (дата обращения: 20.05.2017).

14. DipTrace – Schematic and PCB Design Software. URL: <http://diptrace.com> (дата обращения: 21.05.2017)

15. Keccak code package. URL: <https://github.com/gvanas/KeccakCodePackage> (дата обращения: 20.05.2017).

16. NIST SP 800-185 – SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash / J. Kelsey, S. Chang, R. Perlner. – URL: <https://doi.org/10.6028/NIST.SP.800-185> (дата обращения: 21.05.2017).

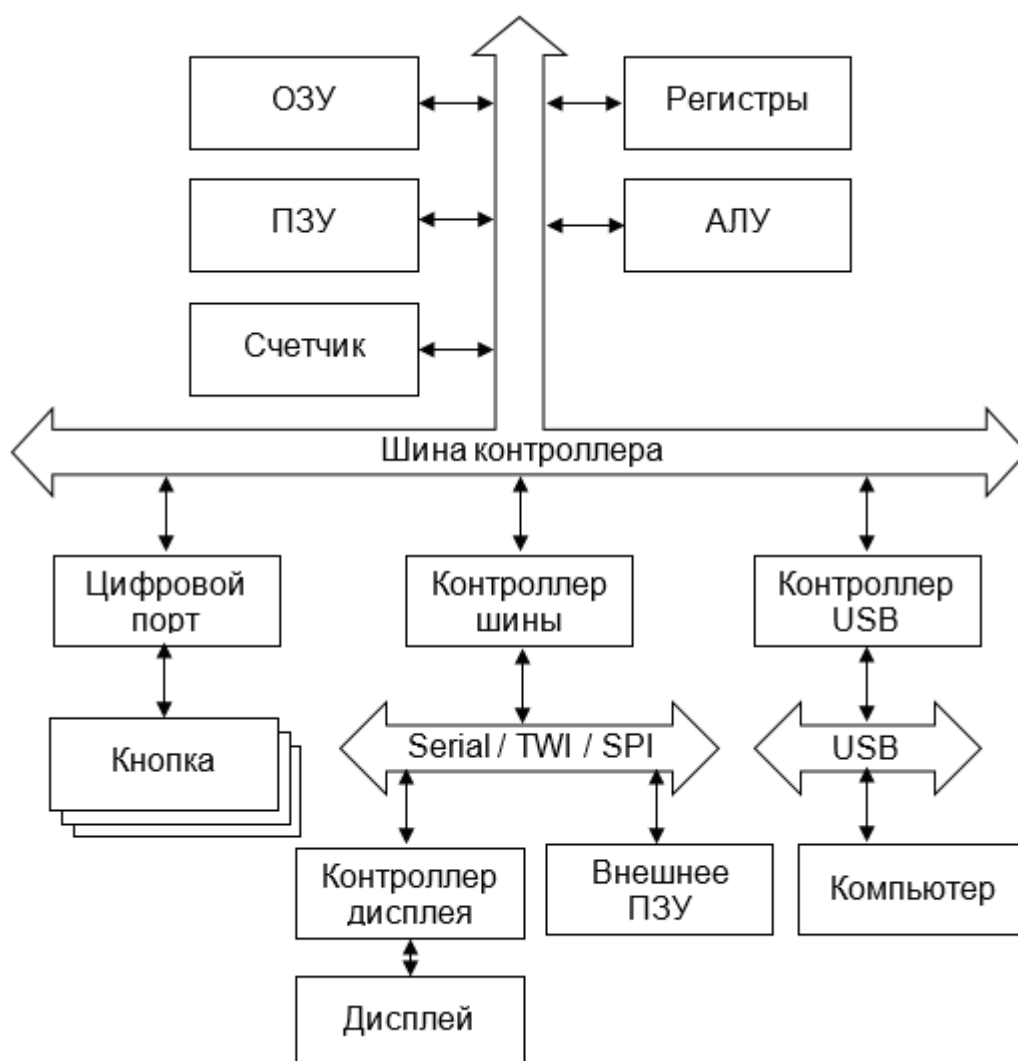
17. SSD1306 Advance Information. 128x64 Dot Matrix OLED/PLED Segment/Common Driver with Controller. URL: <http://www.solomon-systech.com/> (дата обращения: 21.05.2017).

18. Adafruit GFX. URL: <https://cdn-learn.adafruit.com/downloads/pdf/adafruit-gfx-graphics-library.pdf> (дата обращения: 21.05.2017).

19. ATmega16U4/ATmega32U4 Datasheet. – URL: http://www.atmel.com/Images/Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf (дата обращения: 20.05.2017).

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		50

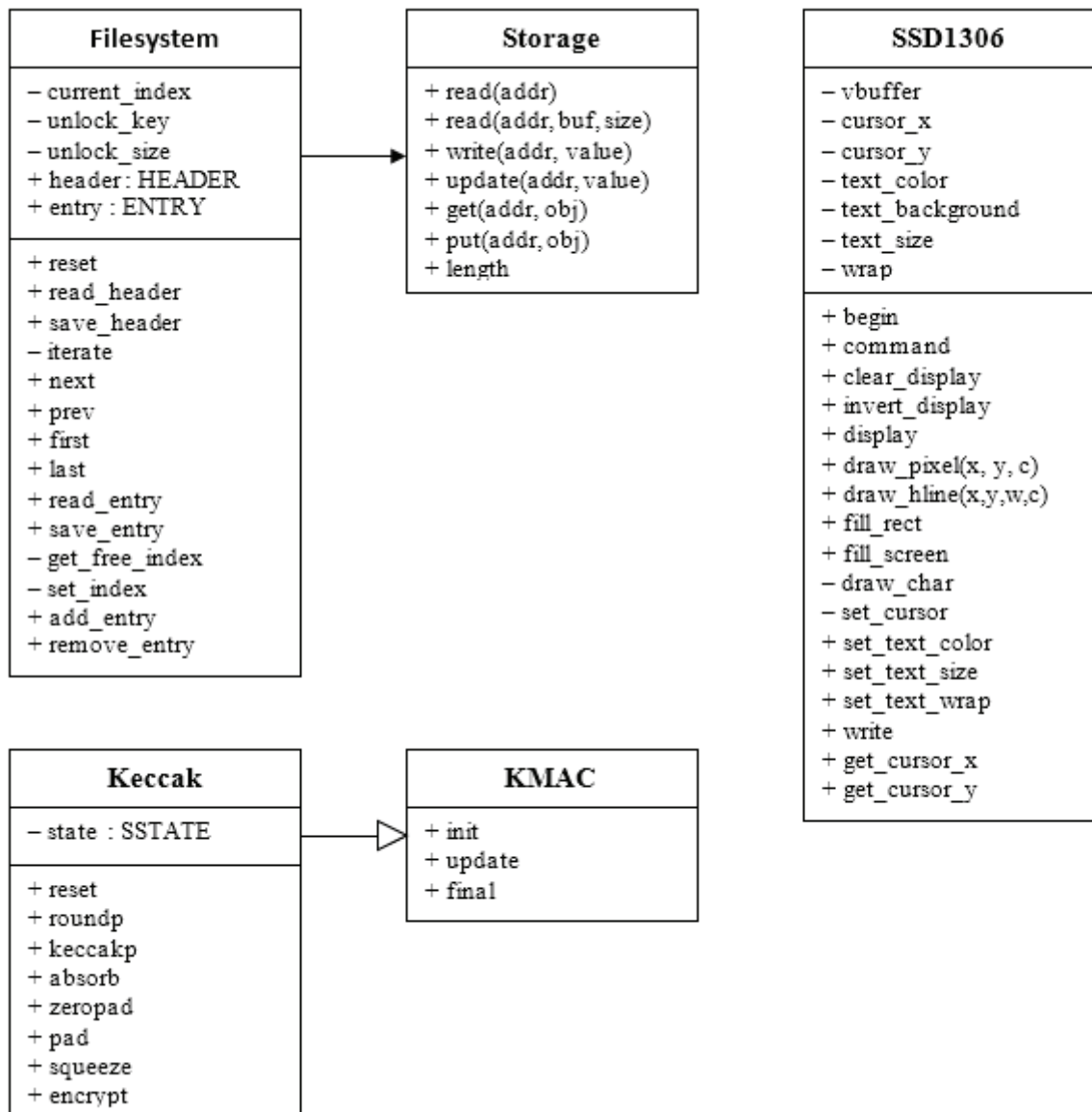
Приложение А. Структурная схема устройства



Приложение Б. Принципиальная схема устройства

					ВКР 090301.20 17 81 01	Лист
						52
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение В. Диаграмма классов



Приложение Г. Исходный код программы

1. Класс криптографии

```
#ifndef KECCAK_H
#define KECCAK_H

#include "utility.h"

#define KMAC

#define kKeccakBlockSize 136

class Keccak
{
    struct
    {
        uint8_t a[200];
        uint8_t input_size;
        uint8_t output_size;
    } state;

public:
    Keccak() { reset(); }
    ~Keccak() { reset(); }

    // Сброс внутреннего состояния.
    void reset()
    {
        ZeroMemory(state.a, sizeof(state.a));
        state.input_size = 0;
        state.output_size = 0;
    }

    // Раунд Кеccak.
    void roundp(uint8_t rci);

    void кеccakp(){
        uint8_t i = 0;
        do {
            roundp(i);
        } while (++i < 24);
    }

    // Поглощение данных губкой.
    void absorb(const void* data, uint16_t size);

    // Дополнение последнего блока нулями.
    void zeropad()
    {
        uint8_t encbuf = 0;
        if (state.input_size != 0)
        {
            state.input_size = (kKeccakBlockSize - 1);
            absorb(&encbuf, 1);
        }
    }
}
```

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		54

```

// Дополнение последнего блока.
void pad(uint8_t tag);

// Сжатие губки.
void squeeze(void* data, uint16_t size);

// Маскирование данных с использованием губки.
void encrypt(void *output, const void *input, size_t size)
{
    state.input_size = 0;

    uint8_t *out = (uint8_t *)output;
    const uint8_t *in = (const uint8_t *)input;
    uint8_t bytes_available;
    while (size > 0)
    {
        if (state.output_size >= kKeccakBlockSize)
        {
            keccakp();
            state.output_size = 0;
        }

        bytes_available = kKeccakBlockSize - state.output_size;
        if (bytes_available > size)
            bytes_available = size;

        const uint8_t *d = ((const uint8_t *) (state.a)) + state.output_size;
        for (uint8_t index = 0; index < bytes_available; ++index)
            out[index] = in[index] ^ d[index];

        state.output_size += bytes_available;
        size -= bytes_available;
        out += bytes_available;
        in += bytes_available;
    }
}

};

#ifdef KMAC

// Функции преобразования массивов байт.
static uint8_t left_encode(uint8_t* encbuf, uint16_t value);
static uint8_t right_encode(uint8_t* encbuf, uint16_t value);

class Kmac : public Keccak
{
public:

    void update(const void* data, uint16_t len)
    {
        absorb(data, len);    // "data"
    }

    void final(uint8_t* out, uint16_t outlen)
    {
        uint8_t encbuf[sizeof(uint16_t) + 1];

        absorb(encbuf, right_encode(encbuf, outlen * 8)); // "L"
        pad(0x04);
        squeeze(out, outlen);
    }
}

```

```

void init(const void* key, uint16_t keylen)
{
    uint8_t encbuf[sizeof(uint16_t) + 1];

    reset();

    absorb(encbuf, left_encode(encbuf, kKeccakBlockSize));
    absorb(encbuf, left_encode(encbuf, 4 * 8));    // "function"
    absorb("KMAC", 4);
    absorb(encbuf, left_encode(encbuf, 0));    // "customization string"
    absorb("", 0);
    zeropad();

    absorb(encbuf, left_encode(encbuf, kKeccakBlockSize));
    absorb(encbuf, left_encode(encbuf, keylen * 8));    // "key"
    absorb(key, keylen);
    zeropad();
}

} kmac;

////////////////////////////////////

void KeccakHash(const void* in, uint16_t in_size, uint8_t* out, uint16_t
out_size)
{
    kmac.reset();
    kmac.absorb(in, in_size);
    kmac.pad(0x01);
    kmac.squeeze(out, out_size);
}

void KeccakMask(const void* key, uint16_t key_size, const void* iv,
uint16_t iv_size, const void* in, void* out, uint16_t size)
{
    kmac.reset();
    kmac.absorb(key, key_size);

    if (iv && iv_size > 0)
        kmac.absorb(iv, iv_size);

    kmac.pad(0x01);
    kmac.encrypt(out, in, size);
}

#endif
#endif

```

					БКР 090301.20 17 81 01	Лист
						56
Изм.	Лист	№ докум.	Подпись	Дата		

2. Функции взаимодействия с пользователем

```
#include <stdint.h>

enum STATUS
{
    S_OK,
    S_CANCEL,
    S_MENU_GO_BACK,
    S_FAIL,
    S_CRC_CORRUPT,
    S_NO_ITEMS,
    S_OUT_OF_BOUNDS,
    S_TOO_MANY_ITEMS
};

// Криптография и дисплей.
#include "кеccak.h"
#include "GFX.h"
SSD1306 display;

// Набор функций-примитивов для построчечного вывода.
inline void DisplayReset(uint8_t start_line)
{
    if (!start_line)
        display.clear_display();
    else
        display.fill_rect(0, start_line * FONT_HEIGHT,
                          SSD1306_LCDWIDTH, SSD1306_LCDHEIGHT, BLACK);
}
inline void DisplayUpdate()
{
    display.display();
}

// Вывод строки, строки из памяти или числа
void DisplayLine(uint8_t line, prog_char* s)
{
    display.fill_rect(0, line * FONT_HEIGHT, SSD1306_LCDWIDTH, FONT_HEIGHT,
                     BLACK);
    display.set_cursor(0, line * FONT_HEIGHT);
    display.println(s);
}
void DisplayLine(uint8_t line, char* s)
{
    display.fill_rect(0, line * FONT_HEIGHT, SSD1306_LCDWIDTH, FONT_HEIGHT,
                     BLACK);
    display.set_cursor(0, line * FONT_HEIGHT);
    display.println(s);
}
void DisplayLine(uint8_t line, uint16_t value)
{
    display.fill_rect(0, line * FONT_HEIGHT, SSD1306_LCDWIDTH, FONT_HEIGHT,
                     BLACK);
    display.set_cursor(0, line * FONT_HEIGHT);
    display.println(value, DEC);
}

// генерация секрета и хранилище.
#include "generator.h"
```

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		57

```

#include "filesystem.h"

////////////////////////////////////
// Структура элемента меню пользователя.
typedef struct _menu_t
{
    prog_char* text;           // текст
    bool      visible;         // видимость
    uint8_t   (*action)(void*); // обработчик
} menu_t;

////////////////////////////////////

// Контакты микроконтроллера, к которым подключены кнопки.
#define UI_NONE    0
#define UI_UP      4
#define UI_DOWN    10
#define UI_SELECT  7

uint8_t UserInput()
{
    uint8_t input = UI_NONE;
    digitalWrite(LED_BUILTIN, LOW);

    // Изменено для работы прототипа.
    while (input == UI_NONE)
    {
        if (digitalRead(UI_UP) == digitalRead(6))
            input = UI_UP;
        else if (digitalRead(UI_DOWN) == digitalRead(12))
            input = UI_DOWN;
        else if (digitalRead(UI_SELECT) == digitalRead(9))
            input = UI_SELECT;
    }

    digitalWrite(LED_BUILTIN, HIGH);
    delay(200);
    return input;
}

// Функция ввода данных через терминал.
uint8_t SerialStringRequest(prog_char* message, char* buffer, uint16_t length)
{
    uint8_t in = 0;
    uint16_t i = 0;

    // ожидать подключения по USB
    while (!Serial)
    {
        DisplayReset(0);
        DisplayLine(0, F("Connect via terminal"));
        DisplayLine(1, F("and press SELECT"));
        DisplayUpdate();

        if (UserInput() != UI_SELECT)
            return S_CANCEL;
    }

    // вывести сообщение на экран и продублировать его в терминале
    DisplayReset(0);
    DisplayLine(0, message);

```

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		58

```

DisplayUpdate();
Serial.println(message);

// сбросить буферы
digitalWrite(LED_BUILTIN, HIGH);
Serial.flush();
while (!Serial.available());

// считывать строку до символа \n
do
{
    in = Serial.read();
    buffer[i++] = in;
} while (in != '\n' && i < length);

buffer[i - 1] = 0; // удалить \n

digitalWrite(LED_BUILTIN, LOW);
Serial.flush();

return S_OK;
}

// Изменение числового счетчика с помощью кнопок.
uint16_t UserInputCounter(uint16_t value, uint8_t start_line = 0)
{
    uint8_t ready = false;
    do
    {
        DisplayReset(start_line);
        DisplayLine(start_line, value);
        DisplayUpdate();

        switch (UserInput())
        {
            case UI_SELECT:
                ready = true;
                break;
            case UI_UP:
                ++value;
                break;
            case UI_DOWN:
                --value;
                break;
        }
    } while(!ready);

    return value;
}

////////////////////////////////////

// Функция добавления записи в хранилище.
uint8_t AddRecord(void*)
{
    Filesystem fs;
    char res_name[SITE_NAME_BUFFER] = {0};
    uint8_t status = S_FAIL;

    DisplayReset(1);

```

```

// считать заголовок данных
if (fs.read_header(NULL, 0) != S_OK)
    DisplayLine(1, F("Not initialized"));
else
{
    // получить имя ресурса из терминала
    status = SerialStringRequest(F("Enter new entry"),
                                res_name, sizeof(res_name) - 1);

    // очистить экран после вызова функции
    DisplayReset(1);

    if (status != S_OK)
    {
        if (status == S_CANCEL)
            DisplayLine(1, F("Operation cancelled"));
        else
            DisplayLine(1, F("Operation failed"));
    }
    else
    {
        if (fs.add_entry(res_name) != S_OK)
            DisplayLine(1, F("Too many entries"));
        else
        {
            DisplayLine(0, F("Entity added"));
            DisplayLine(1, fs.entry.name);
            status = S_OK;
        }
    }
}

DisplayUpdate();
delay(2000);
return status;
}

// Функция установки нового мастер-ключа пользователя.
bool InitMasterKey(uint8_t* master_key)
{
    char user_name[USER_NAME_BUFFER] = {0};
    char user_password[MASTER_PASSWORD_BUFFER] = {0};

    // получить имя пользователя из терминала
    if (SerialStringRequest(F("Enter full name"),
                            user_name, sizeof(user_name) - 1) != S_OK)
        return false;

    // получить пароль из терминала
    if (SerialStringRequest(F("Enter master password"),
                            user_password, sizeof(user_password) - 1) != S_OK)
        return false;

    GenerateMasterKey(user_name, user_password, master_key);

    return true;
}

```

```

// Инициализация устройства и добавление базового набора информационных систем.
// (прототип)
uint8_t Initialize(void*)
{
    uint8_t master_key[kMasterKeySize] = {0};

    // подтвердить сброс
    DisplayReset(0);
    DisplayLine(0, F("All saved data will be lost forever"));
    DisplayLine(3, F("Confirm by SELECT"));
    DisplayUpdate();

    if (UserInput() == UI_SELECT)
    {
        if (!InitMasterKey(master_key))
            return S_FAIL;
        else
        {
            Filesystem fs;

            fs.reset(master_key, NULL, 0);

            fs.add_entry("habr.ru");
            fs.add_entry("google.com");
            fs.add_entry("github.com");
            fs.add_entry("meduza.io");

            return S_OK;
        }
    }

    return S_CANCEL;
}

// Обработчик ввода пользователя при нахождении в меню.
uint8_t MenuHandler(void* p_menu, uint8_t size, uint8_t* current, void* param)
{
    menu_t* menu = (menu_t*) p_menu;
    uint8_t status = S_OK;

    // получить команду пользователя
    switch (UserInput())
    {
    {
    case UI_SELECT:
    {
        status = menu[*current].action(param);    break;
    }
    case UI_UP:
        // прокрутка вверх с переполнением
        do { *current = ((*current != 0) ? *current-1 : size-1); }
        while (!menu[*current].visible);
        break;
    case UI_DOWN:
        // прокрутка вниз с переполнением
        do { *current = ((*current != size-1) ? *current+1 : 0); }
        while (!menu[*current].visible);
        break;
    }
    }
    return status;
}

```

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		61

```

// Параметр генерации секрета
typedef struct
{
    void*    fs;    // объект хранилища
    uint8_t type; // тип секрета
} CommonParam;

// Вызов генератора секрета и вывод результата.
uint8_t CallGenerator(void* p, uint32_t type, uint32_t counter,
                      bool keyboard_out = false)
{
    Filesystem* fs = (Filesystem*) p;
    char password[MAX_PASSWORD_LENGTH + 1];

    GeneratePassword(fs->header.master, fs->entry.name,
                    type, counter, NULL, password);

    if (!keyboard_out)
    {
        // отобразить пароль
        DisplayReset(2);
        DisplayLine(2, counter);
        DisplayLine(3, password);
        DisplayUpdate();

        UserInput();
    }
    else
    {
        delay(500);
        Keyboard.print(password);
    }

    ZeroMemory(password, sizeof(password));
    return S_OK;
}

// Возврат на уровень выше в меню.
uint8_t BackFunc(void*)
{
    return S_MENU_GO_BACK;
}

// Функция вызова генератора секрета для отображения на экране.
// (уменьшение объема кода)
uint8_t DisplayCommon(void* p)
{
    CommonParam* generic = (CommonParam*) p;
    Filesystem* fs = (Filesystem*) generic->fs;
    uint16_t counter;

    switch (generic->type)
    {
        case kMaskLong:  counter = fs->entry.counter.password; break;
        case kMaskName:  counter = fs->entry.counter.login;    break;
        case kMaskPin:   counter = fs->entry.counter.pin;      break;
    }

    return CallGenerator(fs, generic->type, counter);
}

```

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		62

```

// Функция вызова генератора секрета для вывода как клавиатура.
uint8_t EnterGeneric(void* p)
{
    CommonParam* generic = (CommonParam*) p;
    Filesystem* fs = (Filesystem*) generic->fs;
    uint16_t counter;

    switch (generic->type)
    {
        case kMaskLong:    counter = fs->entry.counter.password; break;
        case kMaskName:    counter = fs->entry.counter.login;      break;
        case kMaskPin:     counter = fs->entry.counter.pin;        break;
    }

    return CallGenerator(fs, generic->type, counter, true);
}

// Функция изменения счетчика секрета.
uint8_t ChangeGeneric(void* p)
{
    CommonParam* generic = (CommonParam*) p;
    Filesystem* fs = (Filesystem*) generic->fs;
    uint16_t current;
    uint16_t* next;

    switch (generic->type)
    {
        case kMaskLong:    next = &fs->entry.counter.password; break;
        case kMaskName:    next = &fs->entry.counter.login;      break;
        case kMaskPin:     next = &fs->entry.counter.pin;        break;
    }

    current = *next;
    *next = UserInputCounter(*next, 2);
    if (current != *next)
        fs->save_entry();

    return S_OK;
}

// Обработчик меню работы с секретом.
uint8_t HandleCommon(void* p)
{
    menu_t menu[] = {
        { F("Display"),          true,    &DisplayCommon },
        { F("Enter as keyboard"), true,    &EnterGeneric },
        { F("Change"),            true,    &ChangeGeneric },
        { F("\x1b Back"),         true,    &BackFunc },
    };
    uint8_t menu_current = 0;
    do {
        // отобразить текущий элемент меню
        DisplayReset(2);
        DisplayLine(2, menu[menu_current].text);
        DisplayUpdate();

        // ожидать ввода пользователя
    } while (MenuHandler(menu, sizeof(menu) / sizeof(menu[0]), &menu_current, p)
    != S_MENU_GO_BACK);
    return S_OK;
}

```

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		63

```

// Обработчики меню информационной системы.
uint8_t HandlePassword(void* p)
{
    CommonParam g = {p, kMaskLong};
    return HandleCommon(&g);
}
uint8_t HandleLogin(void* p)
{
    CommonParam g = {p, kMaskName};
    return HandleCommon(&g);
}
uint8_t HandlePin(void* p)
{
    CommonParam g = {p, kMaskPin};
    return HandleCommon(&g);
}

uint8_t RemoveRecord(void* p)
{
    Filesystem* fs = (Filesystem*) p;

    DisplayReset(2);
    DisplayLine(2, F("Are you sure?"));
    DisplayLine(3, F("Confirm by SELECT"));
    DisplayUpdate();

    if (UserInput() == UI_SELECT)
    {
        fs->remove_entry();
        return S_MENU_GO_BACK;
    }

    return S_OK;
}

// Функция меню информационной системы.
void HandleSelectedRecord(void* p)
{
    menu_t menu[] = {
        { F("Password"),          true,    &HandlePassword },
        { F("Login"),              true,    &HandleLogin },
        { F("PIN"),                 true,    &HandlePin },
        { F("Remove record"),       true,    &RemoveRecord },
        { F("\x1b Back"),           true,    &BackFunc },
    };
    uint8_t menu_current = 0;

    do
    {
        // отобразить текущий элемент меню
        DisplayReset(1);
        DisplayLine(1, menu[menu_current].text);
        DisplayUpdate();

        // ожидать ввода пользователя
    } while (MenuHandler(menu, sizeof(menu) / sizeof(menu[0]), &menu_current, p)
    != S_MENU_GO_BACK);
}

```

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		64

```

// Функция вывода списка информационных систем.
uint8_t ShowRecordsList(void*)
{
    uint8_t go_back = 0;
    uint8_t hide_back_button = 1;
    Filesystem fs;

    // считать служебную информацию
    if (fs.read_header(NULL, 0) != S_OK)
    {
        DisplayReset(0);
        DisplayLine(0, F("Not initialized"));
        DisplayUpdate();
        delay(2000);
        return S_FAIL;
    }

    // проверить наличие элементов
    if (fs.header.records > 0)
        fs.first();
    else
    {
        DisplayReset(0);
        DisplayLine(0, F("No entries"));
        DisplayUpdate();
        delay(2000);
        return S_OK;
    }

    do
    {
        if (hide_back_button)
        {
            DisplayReset(0);
            switch (fs.read_entry())
            {
                case S_OK:          DisplayLine(0, fs.entry.name);          break;
                case S_NO_ITEMS:    DisplayLine(0, F("No data"));          break;
                case S_CRC_CORRUPT: DisplayLine(0, F("Corrupted data")); break;
            }
            DisplayUpdate();
        }
        else
        {
            DisplayReset(0);
            DisplayLine(0, F("\x1b Back"));
            DisplayUpdate();
        }

        switch (UserInput())
        {
            case UI_SELECT:
            {
                if (hide_back_button)
                    HandleSelectedRecord(&fs);
                else
                    go_back = 1;
                break;
            }
        }
    }
}

```

					БКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		65

```

        case UI_UP:
            if (hide_back_button)
                hide_back_button = fs.prev();
            else
            {
                fs.last();  hide_back_button = 1;
            }
            break;
        case UI_DOWN:
            if (hide_back_button)
                hide_back_button = fs.next();
            else
            {
                fs.first();  hide_back_button = 1;
            }
            break;
    }
    } while (!go_back);
    return S_OK;
}

// Функция проверки внутреннего состояния криптографии, исполняемая при запуске.
uint8_t Vectors(void*);

// Функция проверки SRAM.
uint8_t SramUsage(void*);

// Функция инициализации устройства
void setup();

// Основная функция-обработчик.
void loop()
{
    menu_t menu[] = {
        { F("Records"),      true,    &ShowRecordsList },
        { F("Add entry"),     true,    &AddRecord },
        { F("Initialize"),    true,    &Initialize },
        { F("SRAM status"),   true,    &SramUsage },
    };
    uint8_t menu_current = 0;

    if (Vectors(NULL) != S_OK)
    {
        DisplayReset(0);
        DisplayLine(0, F("Test vectors failed"));
        DisplayUpdate();
        delay(5000);
        return;
    }

    do
    {
        // отобразить текущий элемент меню
        DisplayReset(0);
        DisplayLine(0, menu[menu_current].text);
        DisplayUpdate();

        // обработать меню
        MenuHandler(menu, sizeof(menu) / sizeof(menu[0]), &menu_current, NULL);
    } while (1);
}

```

3. Класс взаимодействия с графическим дисплеем

```
#ifndef _SSD1306_H
#define _SSD1306_H

#include "commands.h"
#include "font.h"

#define BLACK 0
#define WHITE 1

class SSD1306 : public Print
{
public:

    SSD1306()
    {
        cursor_y_ = 0;
        cursor_x_ = 0;
        text_size_ = 1;
        text_color_ = WHITE;
        text_background_ = WHITE;
        wrap_ = true;

        clear_display();
    }

    // Функция инициализации протокола взаимодействия и контроллера дисплея.
    void begin(bool reset = true);

    // Отправка команды дисплею
    void command(uint8_t c);

    // Очистить экран.
    void clear_display()
    {
        memset(vbuffer, 0, (SSD1306_LCDWIDTH*SSD1306_LCDHEIGHT / 8));
    }

    // Инверсия дисплея (аппаратная).
    void invert_display(uint8_t i)
    {
        command((i) ? SSD1306_INVERT_DISPLAY : SSD1306_NORMALDISPLAY);
    }

    // Установка одного пикселя
    void draw_pixel(int16_t x, int16_t y, uint8_t color)
    {
        if ((x < 0) || (x >= SSD1306_LCDWIDTH) || (y < 0) || (y >= SSD1306_LCDHEIGHT))
            return;

        // x - столбец
        switch (color)
        {
            case WHITE: vbuffer[x + (y / 8)*SSD1306_LCDWIDTH] |= (1 << (y & 7)); break;
            case BLACK: vbuffer[x + (y / 8)*SSD1306_LCDWIDTH] &= ~(1 << (y & 7)); break;
        }
    }
}
```

					BKP 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		67

```

// Отправка содержимого видеобуфера на контроллер.
void display()
{
    command(SSD1306_COLUMNADDR);
    command(0); // Адрес начала столбца (0 = сброс)
    command(SSD1306_LCDWIDTH - 1); // Адрес конца столбца (127 = сброс)

    command(SSD1306_PAGEADDR);
    command(0); // Адрес начала страницы (0 = сброс)
    command(3); // Адрес конца страницы
                // (для 128x32 нужно 4 байта для данных столбца: индекс 3 = сброс)

    for (uint16_t i = 0; i < (SSD1306_LCDWIDTH * SSD1306_LCDHEIGHT / 8); i++)
    {
        // передать несколько байт в одной транзакции
        Wire.beginTransaction(SSD1306_I2C_ADDRESS);
        Wire.write(SSD1306_SETSTARTLINE);
        for (uint8_t x = 0; x < SSD1306_LCDWIDTH / 8; x++)
            Wire.write(vbuffer[i++]);

        i--;
        Wire.endTransmission();
    }
}

// Быстрое заполнение столбца (необходимо для прямоугольника).
void draw_hline(int16_t x, int16_t y, int16_t w, uint8_t color)
{
    if (y < 0 || y >= SSD1306_LCDHEIGHT)
        return;

    // Проверки выхода за границу экрана
    if (x < 0)
    {
        w += x;
        x = 0;
    }

    if ((x + w) > SSD1306_LCDWIDTH)
        w = (SSD1306_LCDWIDTH - x);

    if (w <= 0)
        return;

    // Указатель для прохода по бvideобуферу
    register uint8_t* pBuf = vbuffer;
    pBuf += ((y / 8) * SSD1306_LCDWIDTH); // установить текущую строку
    pBuf += x; // установить текущий столбец

    register uint8_t mask = 1 << (y & 7);
    switch (color)
    {
    case WHITE:
        while (w--) *pBuf++ |= mask;
        break;
    case BLACK:
        mask = ~mask;
        while (w--) *pBuf++ &= mask;
        break;
    }
}

```

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		68

```

// Заполнение прямоугольника цветом.
void fill_rect(int16_t x, int16_t y, int16_t w, int16_t h, uint8_t color)
{
    for (int16_t i = y; i < y + h; i++)
        draw_hline(x, i, w, color);
}

// Очистка экрана.
void fill_screen(uint8_t color)
{
    fill_rect(0, 0, SSD1306_LCDWIDTH, SSD1306_LCDHEIGHT, color);
}

// Нарисовать ASCII символ.
void draw_char(int16_t x, int16_t y, unsigned char c,
               uint8_t color, uint8_t bg, uint8_t size)
{
    if ((x >= SSD1306_LCDWIDTH) ||           // правая граница
        (y >= SSD1306_LCDHEIGHT) ||         // нижняя граница
        ((x + FONT_WIDTH * size - 1) < 0) || // левая граница
        ((y + FONT_HEIGHT * size - 1) < 0)) // верхняя граница
        return;

    for (int8_t i = 0; i < FONT_WIDTH; i++)
    {
        uint8_t line;
        if (i < (FONT_WIDTH - 1))
            line = pgm_read_byte(font + (c * (FONT_WIDTH - 1)) + i);
        else
            line = 0x0;

        for (int8_t j = 0; j < FONT_HEIGHT; j++, line >>= 1)
        {
            if (line & 0x1)
            {
                if (size == 1)
                    draw_pixel(x + i, y + j, color);
                else
                    fill_rect(x + (i*size), y + (j*size), size, size, color);
            }
            else if (bg != color)
            {
                // масштабирование = рисуем прямоугольники
                if (size == 1)
                    draw_pixel(x + i, y + j, bg);
                else
                    fill_rect(x + i*size, y + j*size, size, size, bg);
            }
        }
    }
}

// Сброс курсора.
void set_cursor(int16_t x, int16_t y)
{
    cursor_x_ = x;
    cursor_y_ = y;
}

```

```

// Установка цветов символа и фона.
void set_text_color(uint8_t c)
{
    text_color_ = text_background_ = c;
}

void set_text_color(uint8_t c, uint8_t bg)
{
    text_color_ = c;
    text_background_ = bg;
}

// Установка размера текста.
void set_text_size(uint8_t s) { text_size_ = (s > 0) ? s : 1; }

// Установка переноса текста при выводе.
void set_text_wrap(bool w) { wrap_ = w; }

// Вывод символа в видеобуфер (основная функция).
virtual size_t write(uint8_t c)
{
    if (c == '\n')
    {
        cursor_y_ += text_size_ * FONT_HEIGHT;
        cursor_x_ = 0;
    }
    else if (c == '\r')
    {
        // пропускаем
    }
    else {
        if (wrap_ && ((cursor_x_ + text_size_ * FONT_WIDTH) >= SSD1306_LCDWIDTH))
        {
            // добавляем \n
            cursor_y_ += text_size_ * FONT_HEIGHT;
            cursor_x_ = 0;
        }
        draw_char(cursor_x_, cursor_y_, c,
                  text_color_, text_background_, text_size_);

        cursor_x_ += text_size_ * FONT_WIDTH;
    }

    return 1;
}

// Получение курсора.
int16_t get_cursor_x() const { return cursor_x_; }
int16_t get_cursor_y() const { return cursor_y_; }

private:
uint8_t vbuffer[SSD1306_LCDHEIGHT * SSD1306_LCDWIDTH / 8]; // видеобуфер
int16_t cursor_x_, cursor_y_; // курсор
uint8_t text_color_; // цвет текста
uint8_t text_background_; // цвет фона
uint8_t text_size_; // размер текста
bool wrap_; // перенос текста
};

#endif

```

4. Класс взаимодействия с EEPROM

```
#ifndef STORAGE_H
#define STORAGE_H

#define kTwiStorageAddress 0x50    // TWI адрес микросхемы
#define kStorageLimit      32768   // AT24C256
#define kPageSize          24      // размер считываемого блока

class Storage
{
public:

    // Чтение одного байта по указанному адресу
    uint8_t read(uint16_t address)
    {
        uint8_t data;
        read(address, &data, 1);
        return data;
    }

    // Чтение группы байт по указанному адресу
    void read(uint16_t address, uint8_t* buffer, uint16_t size)
    {
        uint8_t i = 0;
        memset(buffer, 0xFF, size);

        TWI_GET(kTwiStorageAddress, address, buffer, size);
    }

    // Запись одного байта по указанному адресу
    void write(uint16_t address, uint8_t value)
    {
        TWI_PUT(kTwiStorageAddress, address, buffer, size);
    }

    // Обновление одного байта по указанному адресу
    void update(uint16_t address, uint8_t value)
    {
        if (read(address) != value)
            write(address, value);
    }

    // Получение объекта из памяти по указанному адресу
    template<typename T> T &get(uint16_t address, T &object)
    {
        uint8_t* ptr = (uint8_t*) &object;
        uint16_t bytes_to_read = sizeof(T);
        uint8_t bytes_on_step;
        while (bytes_to_read)
        {
            bytes_on_step = min(bytes_to_read, kPageSize);
            read(address, ptr, bytes_on_step);
            ptr += kPageSize;
            address += kPageSize;
            bytes_to_read -= bytes_on_step;
        }

        return object;
    }
};
```

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		71

```

    // Запись объекта в память по указанному адресу
template<typename T> const T &put(uint16_t address, const T &object)
{
    const uint8_t* ptr = (const uint8_t*) &object;
    uint16_t bytes_to_write = sizeof(T);
    uint8_t memory_map[kPageSize];
    uint8_t bytes_on_step;

    while (bytes_to_write)
    {
        bytes_on_step = min(bytes_to_write, kPageSize);
        read(address, memory_map, bytes_on_step);

        for (uint8_t i = 0; i < bytes_on_step; ++i)
        {
            if (ptr[i] != memory_map[i])
                write(address + i, ptr[i]);
        }

        ptr      += kPageSize;
        address += kPageSize;
        bytes_to_write -= bytes_on_step;
    }

    return object;
}

uint16_t length() { return kStorageLimit; }
} storage;

#endif

```

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		72

5. Класс хранилища данных

```

#ifndef FILESYSTEM_H
#define FILESYSTEM_H

#include "storage.h"

// Размеры контрольных сумм.
#define kEntryCrcSize 4
#define kMetaCrcSize 2

// Запись об информационной системе.
typedef struct // 128 + 12
{
    uint8_t crc[kEntryCrcSize]; // 4

    union
    {
        struct
        {
            char name[kResourceNameSize]; // 128
            struct
            {
                uint16_t login;
                uint16_t password;
                uint16_t pin;
                uint16_t phrase;
            } counter; // 8
        };

        uint8_t data[kResourceNameSize + 8];
    };
} ENTRY;

#define METADATA_SIZE 256
#define ENTRY_INDEX_LIMIT ((kStorageLimit - METADATA_SIZE) / sizeof(ENTRY))
#define MASK_ARRAY_SIZE ((ENTRY_INDEX_LIMIT/8) + ((ENTRY_INDEX_LIMIT%8) ? 1:0))

typedef struct
{
    uint16_t records; // количество записей
    uint8_t masks[MASK_ARRAY_SIZE]; // маска доступа к записям
    uint8_t crc[kMetaCrcSize]; // контрольная сумма мастер-ключа
    uint8_t master[kMasterKeySize]; // мастер-ключ
} HEADER;

// Класс хранилища данных в микросхеме EEPROM.
class Filesystem
{
    uint16_t current_index; // индекс текущего элемента
    uint8_t unlock_key[UNLOCK_KEY_LENGTH];
    uint16_t unlock_size;

public:
    HEADER header; // служебная информация
    ENTRY entry; // текущая запись

```

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		73

```

Filesystem ()
{
    memset(&header, 0, sizeof(header));
    memset(&entry, 0, sizeof(entry));
    memset(&unlock_key, 0, sizeof(unlock_key));
    unlock_size = 0;
    current_index = 0;
}

// Сброс служебной информации базы данных.
// С утратой мастер-ключа и таблицы элементов содержимое памяти
// становится бесполезным.
void reset(uint8_t* master_key, uint8_t* unlock, uint16_t size)
{
    memset(&header, 0, sizeof(header));
    memcpy(header.master, master_key, kMasterKeySize);

    if (unlock && size > 0)
    {
        memcpy(unlock_key, unlock, size);
        unlock_size = size;
    }

    // контроль целостности мастер-ключа
    KeccakHash(master_key, kMasterKeySize, header.crc, sizeof(header.crc));
    save_header();
}

// Чтение и проверка целостности служебной информации.
uint8_t read_header(uint8_t* unlock, uint16_t size)
{
    if (unlock && size > 0)
    {
        memcpy(unlock_key, unlock, size);
        unlock_size = size;
    }

    // прочитать и расшифровать
    storage.get(0, header);
    KeccakMask(unlock_key, unlock_size, NULL, 0,
               &header, &header, sizeof(header));

    // проверить целостность
    uint8_t crc[kMetaCrcSize];
    KeccakHash(header.master, kMasterKeySize, crc, sizeof(crc));
    return (memcmp(crc, header.crc, sizeof(crc)) == 0) ? S_OK : S_CRC_CORRUPT;
}

// Сохранение служебной информации.
void save_header()
{
    // зашифровать и сохранить
    KeccakMask(unlock_key, unlock_size, NULL, 0,
               &header, &header, sizeof(header));
    storage.put(0, header);

    // расшифровать для продолжения работы
    KeccakMask(unlock_key, unlock_size, NULL, 0,
               &header, &header, sizeof(header));
}

```

```

    // Добавление нового элемента в хранилище.
uint8_t add_entry(const char* res_name)
{
    uint16_t name_length = strlen(res_name);
    uint16_t old_index = current_index;
    if (name_length >= sizeof(entry.name))
        name_length = sizeof(entry.name) - 1;

    // найти первый свободный блок
    current_index = get_free_index();

    memset(&entry, 0, sizeof(entry));
    memcpy(entry.name, res_name, name_length);

    // попытаться записать по найденному индексу
    if (save_entry() == S_OK)
    {
        header.records++;
        set_index(current_index, 1);
        save_header();
        return S_OK;
    }
    else
    {
        // неудача, вернуть последний успешно считанный блок
        current_index = old_index;
        read_entry();

        return S_TOO_MANY_ITEMS;
    }
}

// Удаление текущего элемента из памяти.
bool remove_entry()
{
    uint16_t address = METADATA_SIZE + current_index * sizeof(ENTRY);
    memset(&entry, 0xFF, sizeof(entry));
    storage.put(address, entry);

    // пометить блок как свободный
    set_index(current_index, 0);
    header.records--;
    save_header();

    // сбросить итератор
    first();

    return true;
}

inline bool next() { return iterate(+1); }
inline bool prev() { return iterate(-1); }

// Установка итератора блоков данных в начало.
void first()
{
    current_index = 0;
    if (bitRead(header.masks[0], 0) == 0)
        next();
}

```

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		75

```

// Установка итератора блоков данных в конец.
void last()
{
    current_index = ENTRY_INDEX_LIMIT;
    prev();
}

// Чтение блока данных с текущего индекса.
uint8_t read_entry()
{
    uint16_t address = METADATA_SIZE + current_index * sizeof(ENTRY);
    uint8_t mask = current_index / 8;
    uint8_t submask = current_index % 8;

    // если блок существует
    if (bitRead(header.masks[mask], submask) == 0)
        return S_NO_ITEMS;

    // прочитать и расшифровать
    storage.get(address, entry);
    KeccakMask(header.master, kMasterKeySize, NULL, 0,
                &entry, &entry, sizeof(entry));

    // проверить целостность
    uint8_t crc[kEntryCrcSize];
    KeccakHash(entry.data, sizeof(entry.data), crc, sizeof(crc));
    if (memcmp(crc, entry.crc, sizeof(crc)) != 0)
        return S_CRC_CORRUPT;

    return S_OK;
}

// Запись блока данных по текущему индексу.
uint8_t save_entry()
{
    uint16_t address = METADATA_SIZE + current_index * sizeof(ENTRY);
    if (current_index > ENTRY_INDEX_LIMIT)
        return S_OUT_OF_BOUNDS;

    // посчитать контрольную сумму и зашифровать
    KeccakHash(entry.data, sizeof(entry.data), entry.crc, sizeof(entry.crc));
    KeccakMask(header.master, kMasterKeySize, NULL, 0,
                &entry, &entry, sizeof(entry));

    // записать в память
    storage.put(address, entry);
    KeccakMask(header.master, kMasterKeySize, NULL, 0,
                &entry, &entry, sizeof(entry));

    return S_OK;
}
private:

// Установка битового флага в маске.
void set_index(uint16_t index, uint8_t value)
{
    uint8_t mask = index / 8;
    uint8_t submask = index % 8;
    bitWrite(header.masks[mask], submask, value);
}

```

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		76

```

// Получение первого свободного индекса.
uint16_t get_free_index()
{
    uint16_t free_index = 0xFFFF;
    for (uint8_t mask = 0; mask < MASK_ARRAY_SIZE; ++mask)
    {
        // пропускать полные блоки
        if (header.masks[mask] != 0xFF)
        {
            for (uint8_t submask = 0; submask < 8; ++submask)
            {
                if (bitRead(header.masks[mask], submask) == 0)
                    return 8 * mask + submask;
            }
        }
    }

    return free_index;
}

// Поиск блока данных по маске доступа.
bool iterate(int8_t direction)
{
    // проверка выхода за границы маски
    uint16_t next_index = current_index + direction;
    if (next_index == ENTRY_INDEX_LIMIT || next_index == 0xFFFF)
        return false;

    register uint8_t mask = next_index / 8;
    register uint8_t submask = next_index % 8;
    uint8_t mask_limit = (direction > 0) ? MASK_ARRAY_SIZE : 0xFF;
    uint8_t submask_limit = (direction > 0) ? 8 : 0xFF;
    uint8_t submask_reset = (direction > 0) ? 0 : 7;

    for (; mask < mask_limit; mask += direction)
    {
        // пропускать заведомо пустые блоки
        if (header.masks[mask] != 0x00)
        {
            for (; submask < submask_limit; submask += direction)
            {
                if (bitRead(header.masks[mask], submask) == 1)
                {
                    current_index = 8 * mask + submask;
                    return true;
                }
            }
        }

        submask = submask_reset;
    }

    return false;
}
};

#endif

```

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		77

6. Функции выработки секрета

```
#ifndef GENERATOR_H // класс расформирован в набор функций
#define GENERATOR_H

#include "utility.h"

#define PASSWORD_SEED_LENGTH 32
#define MAX_PASSWORD_LENGTH PASSWORD_SEED_LENGTH
#define kMasterKeySize 64
#define UNLOCK_KEY_LENGTH 20

#define MASTER_PASSWORD_BUFFER 128
#define USER_NAME_BUFFER 128
#define SITE_NAME_BUFFER 128

#define kResourceNameSize 128

const char PROGMEM alph_alphanumeric_lower[] =
"abcdefghijklmnopqrstuvwxyz0123456789";
const char PROGMEM alph_alphanumeric_mixed[] =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
const char PROGMEM alph_digit[] = "0123456789";
const char PROGMEM alph_letter_lower[] = "abcdefghijklmnopqrstuvwxyz";
const char PROGMEM alph_letter_mixed[] =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
const char PROGMEM alph_vowel_lower[] = "aeiou";
const char PROGMEM alph_vowel_mixed[] = "AEIOUaeiou";
const char PROGMEM alph_consolant_lower[] = "bcdfghjklmnpqrstvwxyz";
const char PROGMEM alph_consolant_mixed[] =
"BCDFGHJKLMNPQRSTVWXYZbcdfghjklmnpqrstvwxyz";
const char PROGMEM alph_special[] =
"!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~";

// маски должны быть короче, чем PASSWORD_SEED_LENGTH
const char* mask_short[] = {"aAadaAAAd", "aaddAAAd", "AaddAaA"};
const char* mask_pin[] = {"dddd"};
const char* mask_name[] = {"cvllclll", "lvlccvll"};
const char* mask_long[] = {
"CvaAdsCvcvCvcv", "CvcvCvavdsCAcv", "CvcvCvcvCvcAds",
"CvAcdsCaAACvcv", "CvccCvavdsCvcv", "CAAcCvavCvcvds"};

typedef enum { kMaskShort, kMaskLong, kMaskPin, kMaskName } E_MaskType;

// Функция выработки мастер-ключа.
// masterSalt = key_scope . #user_name . user_name
// master_key = kmac(user_password, masterSalt, kMasterKeySize, "")
void GenerateMasterKey(const char* user_name, const char* user_password,
uint8_t* master_key)
{
    uint16_t name_length = strlen(user_name);

    kmac.reset();
    kmac.init(user_password, strlen(user_password));

    kmac.update(&name_length, sizeof(name_length));
    kmac.update(user_name, name_length);

    kmac.final(master_key, kMasterKeySize);
}
```

					ВКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		78

```

// Функция получения символа из алфавита.
const char charFromAlphabet(char char_class, uint8_t seed)
{
    const char* alphabet = NULL;

    switch (char_class)
    {
        case 'a': alphabet = alph_alphanumeric_lower; break;
        case 'd': alphabet = alph_digit; break;
        case 'l': alphabet = alph_letter_lower; break;
        case 'L': alphabet = alph_letter_mixed; break;
        case 'v': alphabet = alph_vowel_lower; break;
        case 'V': alphabet = alph_vowel_mixed; break;
        case 'c': alphabet = alph_consolant_lower; break;
        case 'C': alphabet = alph_consolant_mixed; break;
        case 's': alphabet = alph_special; break;
        default:
        case 'A': alphabet = alph_alphanumeric_mixed; break;
    }

    return pgm_read_byte(&alphabet[seed % strlen_P(alphabet)]);
}

// Функция выработки паролей.
// passwordSalt = siteScope . #res_name . res_name . res_counter .
// . #res_context . res_context
// password_seed = kmac( master_key, passwordSalt, PASSWORD_SEED_LENGTH, "" )
void GeneratePassword(const uint8_t* master_key, const char* res_name,
                     const uint32_t mask_type, const uint32_t res_counter,
                     const char* res_context, char* password)
{
    uint8_t password_seed[PASSWORD_SEED_LENGTH] = {0};
    uint16_t name_length = strlen(res_name);
    memset(password, 0, MAX_PASSWORD_LENGTH);

    kmac.reset();
    kmac.init(master_key, kMasterKeySize);

    kmac.update(&name_length, sizeof(name_length));
    kmac.update(res_name, name_length);

    kmac.update(&res_counter, sizeof(res_counter));

    if (res_context)
    {
        uint16_t context_size = strlen(res_context);
        kmac.update(&context_size, sizeof(context_size));
        kmac.update(res_context, context_size);
    }
    kmac.final(password_seed, PASSWORD_SEED_LENGTH);

    // определить шаблон, по которому будет строиться пароль
    uint16_t masks_in_collection = 0;
    const char** mask_collection = NULL;;

    switch (mask_type)
    {
        case kMaskShort:
            masks_in_collection = sizeof(mask_short);
            mask_collection = mask_short;
            break;
    }

```

					БКР 090301.20 17 81 01	Лист
Изм.	Лист	№ докум.	Подпись	Дата		79

```

    case kMaskLong:
        masks_in_collection = sizeof(mask_long);
        mask_collection = mask_long;
        break;
    case kMaskName:
        masks_in_collection = sizeof(mask_name);
        mask_collection = mask_name;
        break;
    default:
    case kMaskPin:
        masks_in_collection = sizeof(mask_pin);
        mask_collection = mask_pin;        break;
}

// скорректировать количество масок
masks_in_collection /= sizeof(const char*);

// определить номер используемой маски
uint8_t chosen_mask = password_seed[0];
char const* mask = mask_collection[chosen_mask % masks_in_collection];
uint16_t mask_length = strlen(mask);

// составить пароль на основе маски и шума
// пропустить первый байт шума (ранее использован)
for (uint16_t c = 0; c < mask_length; ++c)
    password[c] = charFromAlphabet(mask[c], password_seed[c + 1]);

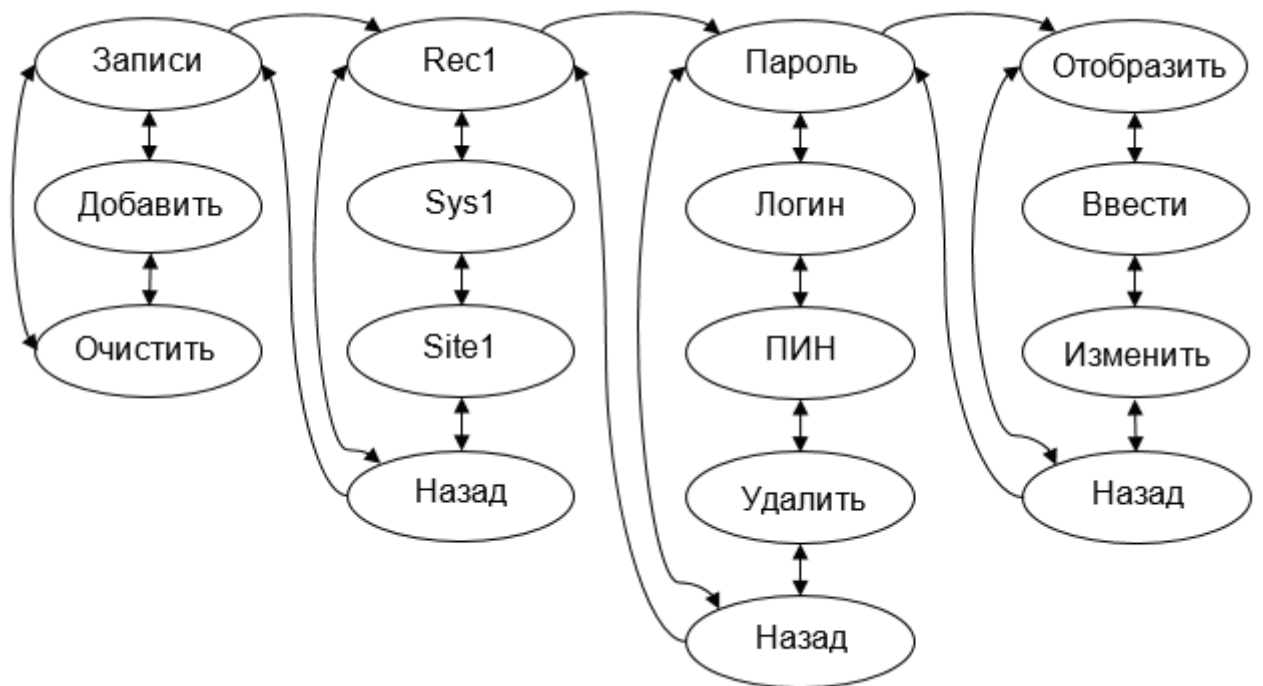
// очистить буфер с шумом
ZeroMemory(password_seed, sizeof(password_seed));
}

#endif

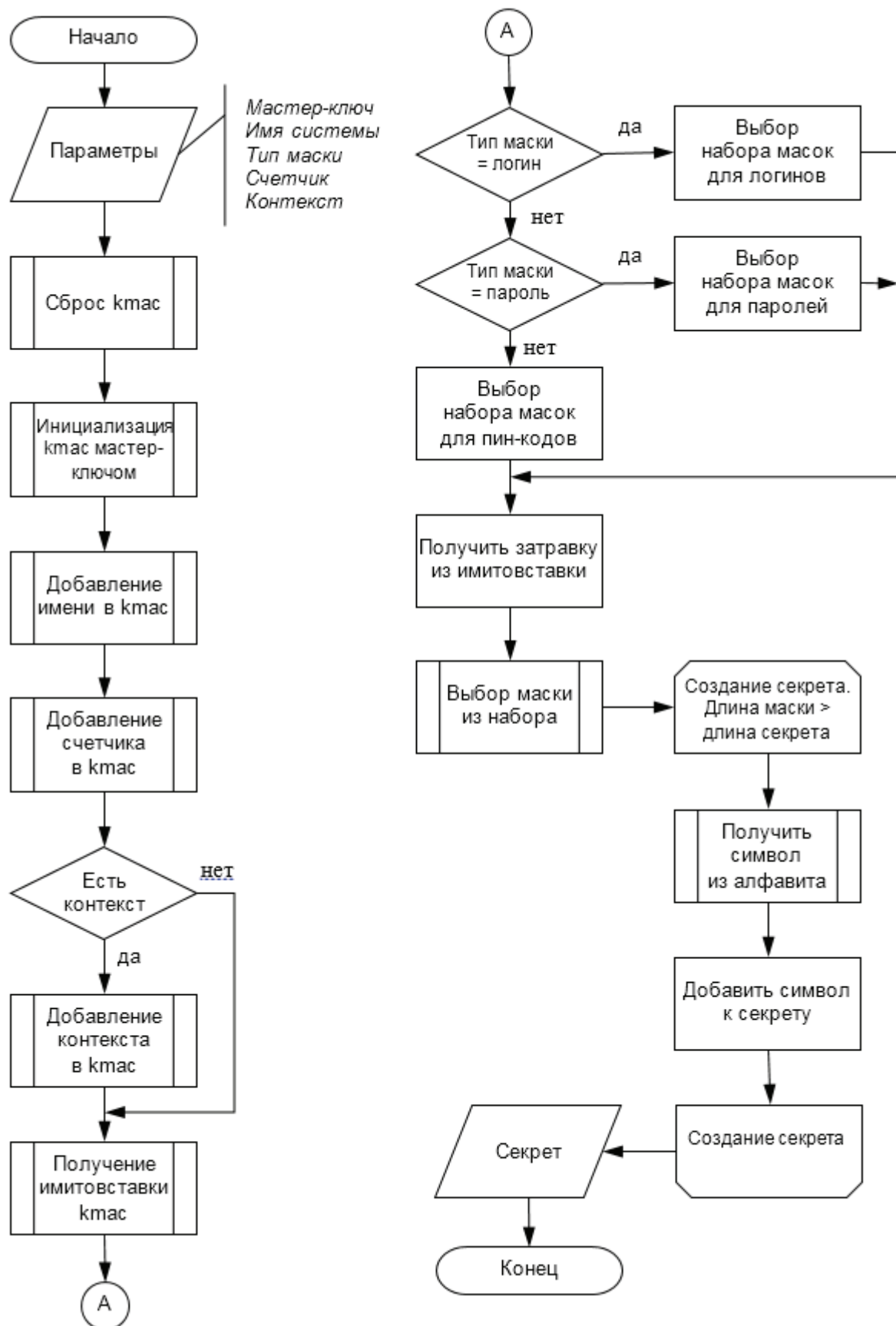
```

					ВКР 090301.20 17 81 01	Лист
						80
Изм.	Лист	№ докум.	Подпись	Дата		

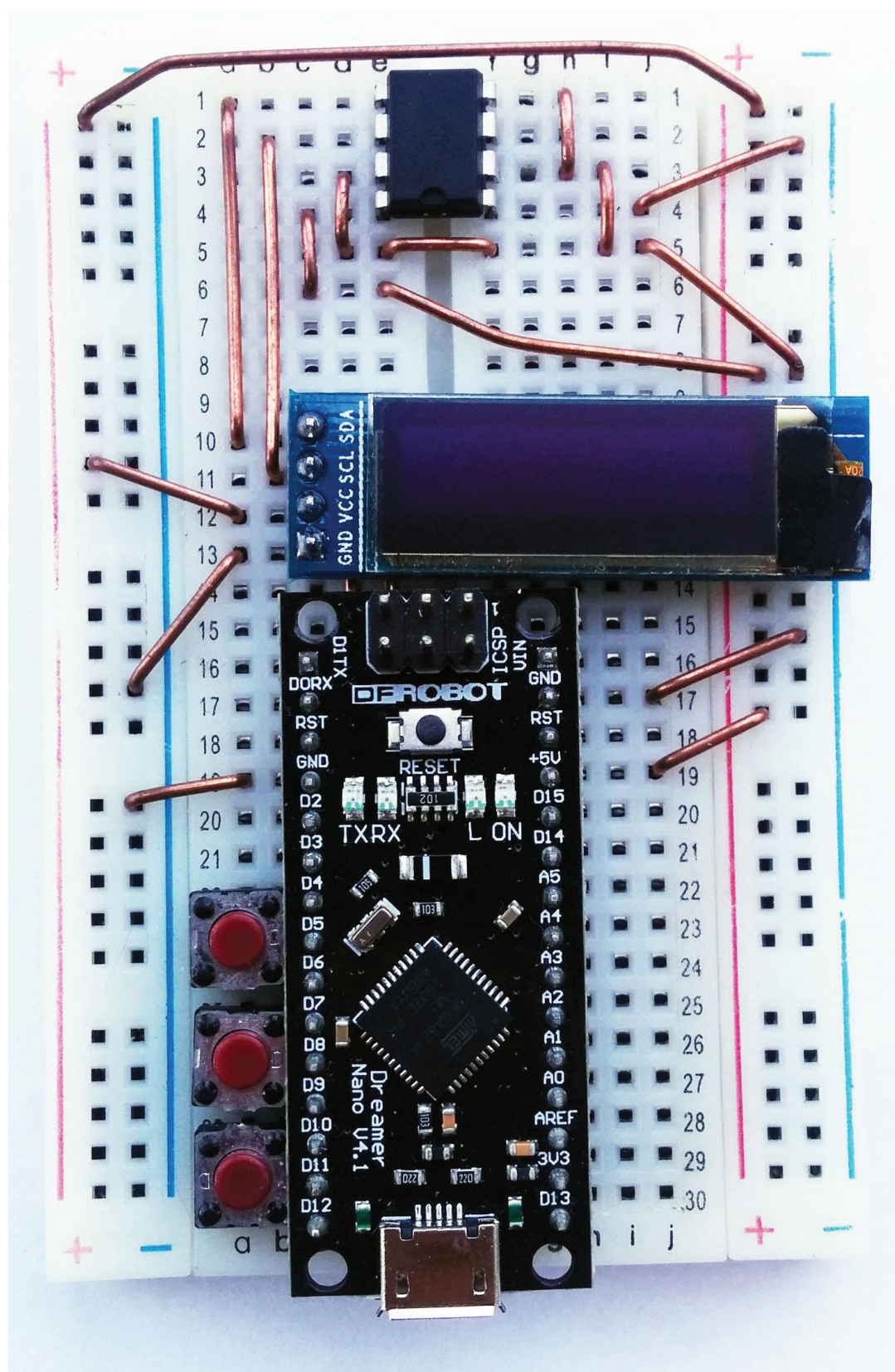
Приложение Д. Граф диалога с пользователем



Приложение Е. Блок-схема алгоритма генерации секрета



Приложение Ж. Прототип устройства



Изм.	Лист	№ докум.	Подпись	Дата

ВКР 090301.20 17 81 01

Лист

83