МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра математического моделирования и анализ данных

ШИЛЯЕВ
Иван Владимирович

КРИПТОГРАФИЧЕСКАЯ ЗАЩИТА ДАННЫХ В IoT СИСТЕМАХ

Дипломная работа

Научные руководители:
ассистент
М.А. Казловский,
кандидат физ.-мат. наук,
доцент И.А. Бодягин

Допущена к защите

«___» _____ 2022 г.

Зав. кафедрой ММАД _____
кандидат физ.-мат. наук, доцент И.А. Бодягин

Минск, 2022

Оглавление

Реферат

Дипломная работа: 49 страниц, 4 главы, 10 рисунков, 5 таблиц, 14 использованных источников, 2 приложения.

Ключевые слова: ИНТЕРНЕТ ВЕЩЕЙ, КРИПТОГРАФИЧЕСКАЯ ЗАЩИТА ДАННЫХ, АУТЕНТИФИЦИРОВАННОЕ ШИФРОВАНИЕ.

Объект исследования: криптографическая защита данных в протоколах, применяемых в сфере интернета вещей.

Цель работы: изучение сетевых протоколов, применяемых в сфере интернета вещей, изучение и сравнение безопасности этих протоколов, анализ уязвимостей и угроз, разработка прототипа умного устройства и протокола взаимодействия с применением белорусских криптографических стандартов.

Методы исследования: а) теоритические: изучение источников, посвящённых протоколам, применяемым в сфере интернета вещей; изучение характеристик этих протоколов, методов, применяемых для защиты данных; б) практические: составление матрицы, сравнивающей устойчивость выбранных технологий к некоторому общему набору угроз в целях выявления наиболее криптостойкого решения; разработка прототипа умного устройства на собственной прошивке, использующей методы защиты данных, описанные в белорусском криптографическом стандарте.

Результат: разработанная прошивка, использующая sponge-функцию и аутентифицированное шифрование, описанные в стандарте СТБ 34.101.77, а также устройство, работающее на этой прошивке.

Область применения: сфера информационной безопасности и интернета вещей.

Abstract

Diploma thesis: 49 pages, 4 chapters, 10 figures, 5 tables, 14 sources, 2 attachments.

Keywords: INTERNET OF THINGS, CRYPTOGRAPHIC DATA PROTECTION, AUTHENTICATED ENCRYPTION.

Object of study: cryptographic data protection in protocols used in the Internet of Things.

Purpose of work: study of networking protocols used in the Internet of Things, study and comparison of these protocols security, analysis of vulnerabilities and threats, development of a smart device prototype and interaction protocol using Belarusian cryptographic standards.

Research methods: a) theoretical: study of the sources devoted to protocols, used in the Internet of Things; study the characteristics of these protocols and data protection methods; b) practical: creation of matrix comparing the robustness of the selected technologies to a common set of threats in order to identify the most crypto-resistant solution; development of a smart device prototype on its own firmware, using data protection methods described in the Belarusian cryptographic standard.

Result: developed firmware that uses sponge function and authenticated encryption described in the standard СТБ 34.101.77, as well as a device running on this firmware.

Scope: the sector of information security and the Internet of Things.

Введение

Термин «Интернет вещей» («Internet of Things») появился более 20 лет назад, а история развития технологии насчитывает почти два столетия. Среди множества определений термина можно выделить следующее: интернет вещей — это глобальная сеть объектов, подключённых к интернету, которые взаимодействуют между собой и обмениваются данными без вмешательства человека.

Основными компонентами IoT систем являются:

- объекты, или «вещи»;
- данные, которыми они обмениваются;
- инфраструктура, с помощью которой осуществляется взаимодействие.

К последнему пункту можно отнести разнообразные виды соединения и каналы связи, программные средства и протоколы. Инфраструктура и её криптографический аспект представляют собой наибольший практический интерес и составляют предметную область данной работы.

Говоря о практическом применении Интернета вещей, многие отрасли выигрывают при использовании этой технологии. И в каждой из этих отраслей необходимо думать о безопасности и защите данных. В связи с этим возникают задачи актуализации знаний об алгоритмах и протоколах, применяемых в данной сфере, их сравнения и реализации в рамках программного обеспечения, изучения уязвимостей, а также рассмотрения вариантов модификации и улучшения этих протоколов с применением белорусской криптографии. Эти задачи и легли в основу данной работы. В соответствии с задачами были поставлены следующие цели:

1. Изучить сетевые протоколы, применяемые в сфере IoT, и провести их сравнительный анализ;
2. Разобрать криптографический аспект изученных в соответствии с первой целью сетевых протоколов в контексте используемых в них криптографических протоколов и алгоритмов;
3. Описать уязвимости и угрозы используемых решений, а также составить матрицу угроз, демонстрирующую устойчивость выбранных технологий к некоторому общему набору угроз;
4. Разработать собственный прототип, состоящий из управляющего и умного устройств, а также протокол взаимодействия между этими устройствами с применением белорусских криптографических стандартов.

Данная работа состоит из 4 глав, в которых последовательно раскрываются все перечисленные выше вопросы.

Глава 1
Анализ литературы

1.1 Технологии Интернета вещей

IoT включает в себя бесчисленное количество технологий и решений, и чтобы понять их все, необходимо потратить немало времени. Однако в целях упрощения существует возможность разбить весь IoT стек на четыре базовых технологических уровня, которые позволяют функционировать всему Интернету вещей.

Аппаратное обеспечение устройств является первым из этих уровней. Устройства — это те самые «вещи» в аббревиатуре IoT. Выступая в роли интерфейса между реальным и цифровым миром, они могут принимать разные формы и размеры, а также иметь разные уровни технологической оснащённости в зависимости от выполняемой задачи. Практически любой предмет может быть подключен к Интернету и оснащён необходимым инструментарием (сенсорами, датчиками и т.д.) в целях измерения и сбора данных. Единственным существенным ограничением может быть реальный практический сценарий использования.

Программное обеспечение является элементом, который делает девайсы по-настоящему «умными». Программы ответственны за коммуникацию с облаком, сбор данных, взаимодействие между устройствами, а также анализ данных в реальном времени. Более того, программное обеспечение помогает взаимодействовать с IoT системами на уровне приложения конечному пользователю, визуализируя обработанные данные для него.

Уровень коммуникации (уровень сообщения) тесно связан с программным и аппаратным обеспечением, однако важно рассматривать его отдельно. Этот уровень содержит средства для обмена информацией между умными устройствами и остальным IoT миром. Он включает в себя как физическое соединение, так и специальные протоколы, на которых будет сделан акцент в данной работе. Выбор правильного решения для обмена сообщениями является ключевым при построении каждой системы. Технологии отличаются в зависимости от способа передачи данных и управления устройствами.

Благодаря программному и аппаратному обеспечению девайсы могут считывать, что происходит вокруг, и коммуницировать с пользователями по специальным каналам связи. **IoT платформа** — это место, в котором все собранные данные обрабатываются, анализируются и представляются пользователю в удобном виде. Её достоинством является извлечение полезных данных из большого объёма информации, который передаётся от устройств по каналам связи.

1.2 Используемые протоколы

Существует множество разнообразных способов взаимодействия умных устройств между собой. Поэтому при выборе протоколов для Интернета вещей часто возникает вопрос о том, есть ли реальная необходимость разработки новых решений, в то время как хорошо зарекомендовавшие себя протоколы сети Интернет уже используются повсеместно десятилетиями. Причина для этого кроется в том, что существующие протоколы часто оказываются недостаточно эффективными и слишком энергоёмкими для работы с возникающими IoT технологиями. Поэтому речь пойдёт об альтернативных решениях, посвящённых именно IoT системам.

Одна из возможных классификаций разбивает все протоколы на три группы: ближнего, среднего и дальнего действия. Наиболее ярким представителем первой группы является Bluetooth, который несмотря на свою повсеместную распространённость остаётся далеко не лучшим решением, особенно при передаче больших объёмов данных. К последней группе относят такие протоколы как NB-IoT, LoRaWAN и SigFox. Эти решения являются весьма современными и продвинутыми, однако используются часто в масштабах предприятий. Наша же цель заключается в изучении решений, применимых к простым пользователям IoT систем, поэтому данный раздел будет преимущественно сконцентрирован вокруг второй группы, а именно протоколов средней зоны действия.

Однако для полноты картины посмотрим вначале на протоколы ближнего и дальнего действия.

1.2.1 Bluetooth

Bluetooth – это стандарт беспроводной связи, предназначенный для обмена информацией на небольших расстояниях между мобильными и некоторыми другими типами устройств. Этот стандарт описан в спецификации IEEE 802.15.1. Первая версия Bluetooth 1.0 была выпущена в 1998 году. Начиная с версии Bluetooth 4.2 (2014 год) было заявлено о поддержке IoT, а Bluetooth 5 и вовсе позиционируется как одна из лидирующих технологий в среде IoT. Однако у неё есть свои нюансы. До версии 5.0 главным из них было малое количество устройств, способных объединяться в одну сеть. Оно ограничивалось лишь 8-ю узлами. Но в последних версиях этот показатель был увеличен на порядок. На данный момент максимальное количество устройств в Bluetooth-сети составляет 32767.

Bluetooth функционирует в частотах от 2,4 до 2,485 ГГц. Дальность действия в 5-й версии составляет 40 метров, а скорость передачи данных — до 2 Mbit/s. Была проведена существенная работа над энергопотреблением устройств. А кроме того в последней версии заявлено о поддержке ячеистой топологии сети (более подробно о ней в разделе о ZigBee), что в будущем, при

появлении большего количество устройств на новых версиях, позволит технологии как минимум сравниться по популярности и сценариям использования с другими стандартами, такими как ZigBee, Z-Wave и Wi-Fi.

1.2.2 Протоколы дальнего действия

Рассмотрим лишь некоторые, самые популярные решения.

- NarrowBand-IoT. Это новый стандарт радио технологий, который обеспечивает экстремально низкое энергопотребление устройств (до 10 лет от одной батареи). Кроме того, он использует уже существующую инфраструктуру сетей сотовой связи, что обеспечивает глобальное покрытие и гарантированное качество сигнала.

- LoRaWAN. Расшифровывается как Long Range Wide-Area Networking. Он заточен под низкое энергопотребление и поддерживает огромные сети с миллионами устройств. Используется в масштабах умных городов. Типичными примером может послужить дистанционное считывание показаний счётчиков электроэнергии и водоснабжения в многоквартирных домах.

- SigFox. Технология обеспечивает простую и быструю связь между сенсорными устройствами в пределах беспроводной сети. Характеризуется ещё более низким энергопотреблением (до 20 лет от одной батарее), однако имеет зависимость от существующей сотовой инфраструктуры.

Поскольку перечисленные протоколы сильно отличаются по своим характеристикам от протоколов ближнего и среднего действия, имеет смысл сравнить их отдельно в приведенной ниже таблице.

	NB-IoT	LoRaWAN	Sigfox
Скорость передачи	100 Kbit/s	50 Kbit/s	100 bit/s
Энергопотребление	Низкое	Низкое	Низкое
Частота	$<$ 1 GHz	150 MHz - 1 GHz	900 MHz
Топология сети	Звезда	Звезда	Звезда

Таблица 1.1: Сравнение протоколов дальнего действия

1.2.3 ZigBee

ZigBee — это один из протоколов верхнего уровня, используемый в домашней автоматизации и других сферах IoT, построенный на базе стандарта IEEE 802.15.4. Он поддерживает высокую отказоустойчивость, низкое энергопотребление, безопасность и надёжность.

Протокол ZigBee описывает беспроводные персональные сети (Wireless personal area network, WPAN). Индивидуальные устройства в подобной сети могут работать на одной батарее до двух лет. Сети на основе ZigBee характеризуются довольно низкой пропускной способностью (до 250 Кбит/с) и дальностью связи между узлами до 100 метров. Протокол был задуман в 1998 году. Первоначальная спецификация была признана стандартом IEEE в 2003 году, а первые модули, совместимые с ZigBee, появились в массовой продаже в начале 2006 года [1].

Существует три типа устройств ZigBee:

1. Координатор ZigBee (ZigBee Coordinator, ZC). Каждая сеть ZigBee должна иметь один координатор, который управляет всей сетью. Координатор выполняет функцию «центра доверия» (Trust Center, TC), обеспечивая генерацию, хранение и распространение ключей безопасности. Он отвечает за создание сети путём выбора наиболее свободного канала для коммуникации устройств. После этого координатор разрешает другим устройствам присоединяться к сети и покидать её, ведя учёт всех устройств в сети. ZigBee координатор не может находиться в спящем режиме и должен быть непрерывно подключён к источнику питания.

2. Маршрутизатор ZigBee (ZigBee Router, ZR). Маршрутизатор выступает в качестве промежуточного звена между координатором и конечными устройствами. Сперва он должен получить разрешение на присоединение к сети от координатора, а после может перенаправлять трафик между устройствами и позволять новым устройствам подключаться к сети. По аналогии с координатором маршрутизатор не может находиться в спящем режиме. Является опциональным устройством в сети.

3. Конечное устройство ZigBee (ZigBee End Device, ZED). Это самый простой тип устройств в сети ZigBee. Обычно они питаются от батареи. Потребитель чаще всего знаком именно с этим типом устройства, к которому относятся, например, умные лампочки или датчики движения. Они содержат достаточно функций, чтобы общаться с координатором или маршрутизатором, но не могут передавать данные от других устройств. Такое взаимодействие позволяет узлу находиться в спящем режиме значительную часть времени, что обеспечивает длительность автономной работы. ZED требует наименьшего объёма памяти, поэтому является более дешёвым в производстве, чем координатор или маршрутизатор.

ZigBee был разработан как стандарт для радиосетей с ячеистой (mesh) топологией. Кроме того, ZigBee поддерживает сети с топологией «звезда» и «дерево». В каждой сети должно быть одно устройство-координатор. В сетях с топологией «звезда» координатор должен быть центральным узлом. Как

	ZC	ZR	ZED
Создание сети ZigBee	x		
Разрешение на присоединение к сети другим устройствам	x	x	
Назначение 16-битного сетевого адреса	x	x	
Обнаружение и запись путей для эффективной доставки сообщений	x	x	
Маршрутизация сетевых пакетов	x	x	x
Присоединение и выход из сети	x	x	x
Режим сна			x

Таблица 1.2: Типы устройств ZigBee

древовидные, так и ячеистые сети позволяют использовать маршрутизаторы ZigBee для расширения связи на сетевом уровне. Благодаря этому, используя несколько координаторов, дальность соединения может быть намного более базовых 100 метров. А за счёт построения оптимального маршрута в сетях с ячеистой топологией сеть в некоторых случаях может продолжать работу даже при выходе их строя одного из координаторов.

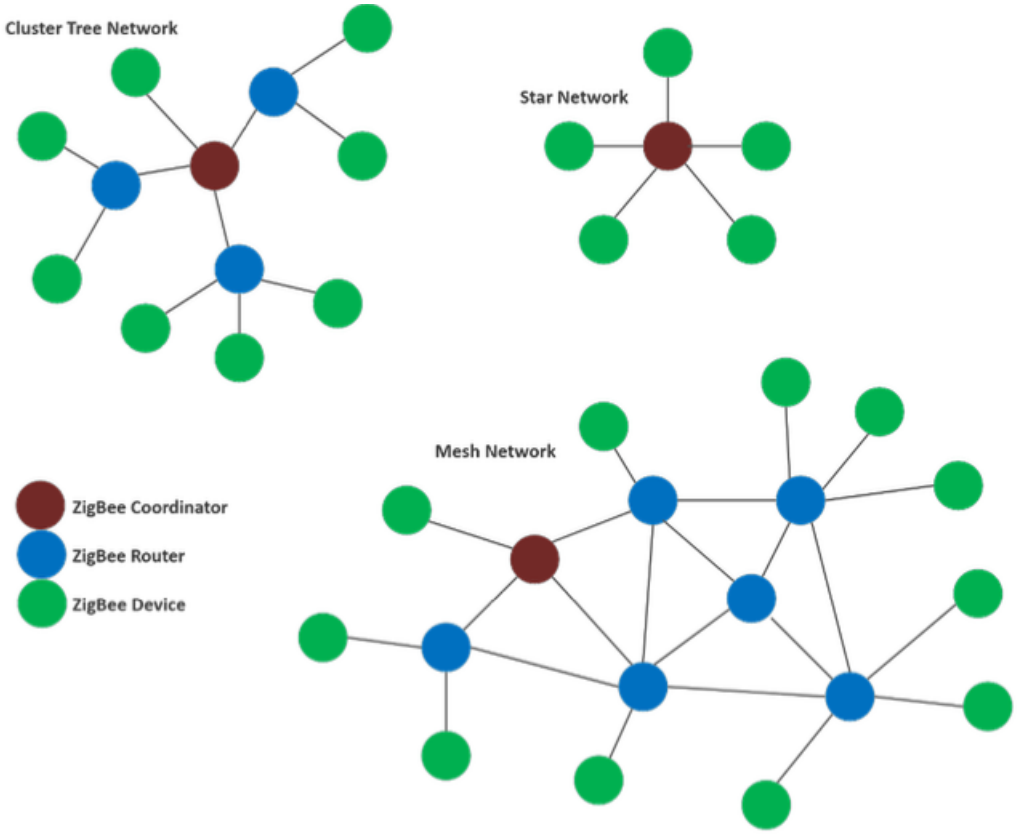

Рисунок 1.1: Топологии сети ZigBee

Архитектура протокола состоит их четырёх уровней:

1. Физический (PHY). Физический уровень относится к стандарту IEEE 802.15.4. Этот уровень является ответственным за передачу битов информации путём отправки и получения пакетов данных, а также за выбор канала (и, соответственно, частоты).

2. MAC (Medium Access Control). MAC — ещё один уровень, относящийся к стандарту IEEE 802.15.4. Он служит интерфейсом между физическим и сетевым уровнями. MAC обеспечивает механизмы адресации и управления доступом к каналам.

3. Сетевой (NWK). Этот уровень определяется уже ZigBee-спецификацией. Он ответственен в первую очередь за выбор топологии сети. После выбора канала координатор назначает каждому присоединяющемуся к сети устройству специальный идентификатор — PAN ID (Personal Area Network ID). Этот идентификатор (16-битное число) используется для логического отделения узлов одной сети ZigBee от узлов другой. На сетевом уровне также устанавливается адрес каждого узла в сети.

4. Прикладной (APL). Прикладной уровень (или уровень Приложения) включает несколько подуровней. Application Support Sublayer (APS) предоставляет программный интерфейс между уровнем NWK и приложениями, которые могут работать на устройстве. Именно он отвечает за безопасность всего прикладного уровня. ZigBee Device Object (ZDO) определяет, будет ли устройство координатором или конечным устройством. Application Framework является окружением для приложений ZigBee. Он определяется вендором, реализующим спецификацию, однако в последнее время много усилий направлено на совместимость устройств от различных производителей.

Таким образом, ZigBee — это спецификация протоколов APS (прикладного) и NWK (сетевого) уровней, использующих сервисы нижних уровней (MAC и PHY), регламентированных стандартом IEEE 802.15.4.

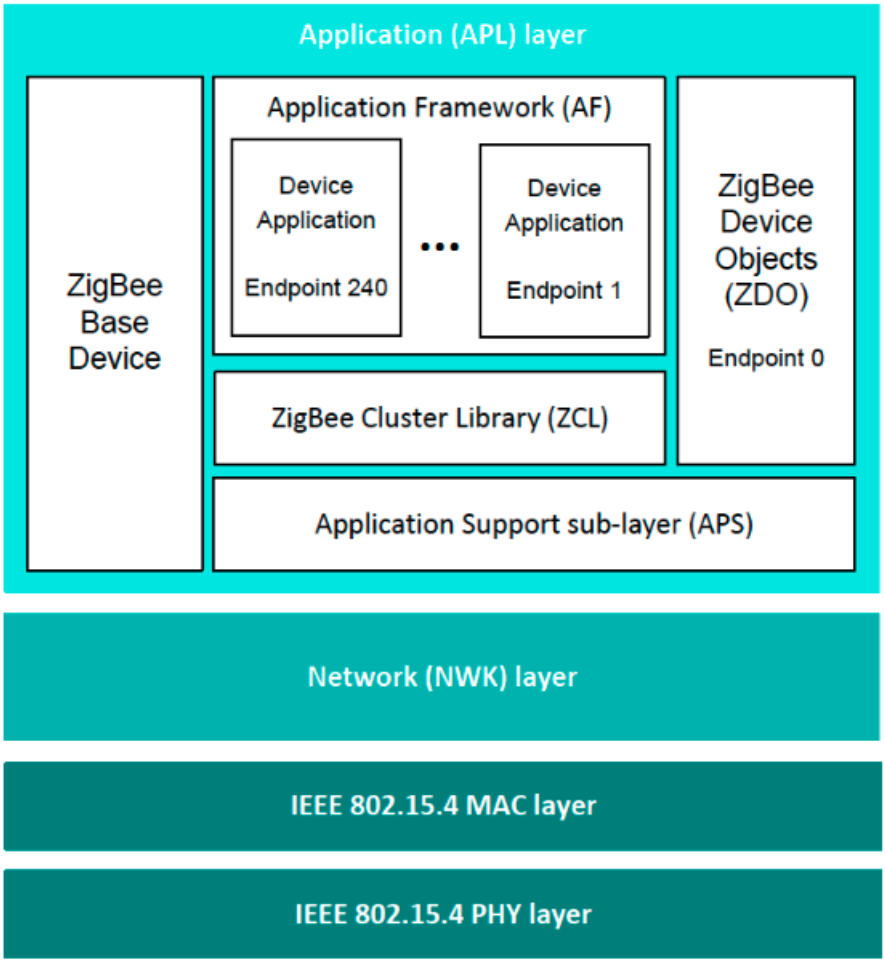

Рисунок 1.2: Сетевые уровни ZigBee

Типичные сферы применения протокола:

- домашняя автоматизация;
- промышленные системы управления;
- сбор медицинских данных;
- оповещение о задымлении и несанкционированном проникновении;
- автоматизация зданий.

ZigBee Alliance — это группа компаний, которые поддерживают и публикуют стандарт ZigBee [2]. Название ZigBee является зарегистрированной торговой маркой этой группы и представляет собой не просто технический стандарт. Организация публикует материалы, которые позволяют производителям создавать совместимые продукты. Связь между IEEE 802.15.4 и ZigBee похожа на связь между IEEE 802.11 и Wi-Fi Alliance. За годы существования альянса его членами стали более 500 компаний.

1.2.4 Z-Wave

Z-Wave — это протокол беспроводной связи, используемый в основном для домашней автоматизации. Он применяется преимущественно для управления бытовой техникой и другими устройствами, такими как освещение, охранные системы, термостаты, окна, замки, бассейны и открыватели гаражных дверей. Как и другие протоколы, предназначенные для рынка автоматизации дома и офиса, система Z-Wave может управляться через Интернет со смартфона, планшета или компьютера, а также локально через умную колонку или хаб, настенную панель со шлюзом Z-Wave или центральным устройством управления. Z-Wave обеспечивает совместимость на прикладном уровне между системами управления домом различных производителей, входящих в её альянс. Число совместимых продуктов Z-Wave значительно растёт, к 2019 году их количество составляло более 2600 [3].

Протокол Z-Wave был разработан датской компанией Zensys, расположенной в Копенгагене, в 1999 году. В этом же году была представлена потребительская систему управления светом. Набор микросхем серии 100 был выпущен в 2003 году, а серии 200 — в мае 2005 года. Микросхема серии 500, также известная как Z-Wave Plus, была выпущена в марте 2013 года, с увеличенным в четыре раза объёмом памяти, улучшенным радиусом действия беспроводной связи и увеличенным временем автономной работы. Технология начала распространяться в Северной Америке примерно в 2005 году, когда пять компаний приняли Z-Wave и сформировали Z-Wave Alliance, целью которого является продвижение использования технологии Z-Wave [4]. При этом все продукты компаний, входящих в альянс, должны быть совместимы. В том же 2005 году технология получила первые инвестиции.

В настоящее время Z-Wave Alliance насчитывает более 700 производителей. Основными членами альянса являются ADT Corporation, Assa Abloy, Jasco, Leedarson, LG Uplus, Nortek Security & Control, Ring, Silicon Labs, SmartThings, Trane Technologies и Vivint.

Взаимодействие Z-Wave на уровне приложений обеспечивает обмен информацией между устройствами и позволяет всем аппаратным и программным средствам Z-Wave работать вместе. Технология беспроводной ячеистой сети (аналогичная с ZigBee) позволяет любому узлу напрямую или косвенно общаться с соседними узлами, управляя любыми дополнительными узлами. Узлы, находящиеся в радиусе действия, общаются друг с другом напрямую. Если они не находятся в радиусе действия, они могут связаться с другим узлом, который расположен в зоне действия обоих узлов, чтобы получить доступ и обменяться информацией.

Z-Wave разработан для обеспечения надёжной передачи небольших пакетов данных с низкой задержкой на скорости до 100 кбит/с. Пропускная способность составляет 40 кбит/с и подходит для приложений управления и датчиков, в отличие от Wi-Fi и других систем беспроводных локальных

сетей на базе IEEE 802.11, которые предназначены в основном для высокой скорости передачи данных. Расстояние связи между двумя узлами составляет около 40 метров.

Z-Wave функционирует в диапазоне частот до 1 ГГц. Этот диапазон конкурирует с некоторыми беспроводными телефонами и другими устройствами бытовой электроники, но позволяет избежать помех в виде Wi-Fi, Bluetooth и других систем, работающих в переполненном диапазоне 2,4 ГГц.

Z-Wave использует архитектуру ячеистой сети с маршрутизацией от источника. Устройства могут связываться друг с другом, используя промежуточные узлы для активной маршрутизации и обхода бытовых препятствий или мертвых зон. Таким образом, сеть Z-Wave может охватывать гораздо большее расстояние, чем радиус действия одного узла. Однако при наличии нескольких таких переходов может возникнуть небольшая задержка между управляющей командой и желаемым результатом.

Простейшая сеть представляет собой одно управляемое устройство и первичный контроллер. Дополнительные устройства могут быть добавлены в любое время, как и вторичные контроллеры, включая приложения для смартфонов и ПК, разработанные для управления и контроля сети Z-Wave. Сеть может включать до 232 устройств, а при необходимости увеличения количества устройств возможно объединение сетей.

Каждой сети Z-Wave назначается идентификатор, а каждое устройство содержит идентификатор узла. Идентификатор сети (Home ID) — это общая идентификация всех узлов, принадлежащих к одной логической сети Z-Wave. Сетевой ID имеет длину 4 байта и присваивается каждому устройству первичным контроллером, когда устройство добавляется в сеть. Узлы с разными сетевыми идентификаторами не могут взаимодействовать друг с другом. Идентификатор узла — это адрес одного узла в сети. Идентификатор узла имеет длину 1 байт и является уникальным в своей сети.

Чип Z-Wave оптимизирован для устройств, работающих от батарей, и большую часть времени находится в режиме энергосбережения, чтобы потреблять меньше энергии, просыпаясь только для выполнения своей функции. В ячеистых сетях Z-Wave каждое устройство в доме распространяет беспроводные сигналы по всему дому, что приводит к низкому энергопотреблению, позволяя устройствам работать годами без необходимости замены батарей. Чтобы устройства Z-Wave могли передавать сторонние сообщения, они не должны быть в спящем режиме. Поэтому устройства, работающие от батарей, не предназначены для использования в качестве ретрансляторов.

1.2.5 Wi-Fi

Построенный на базе стандарта IEEE 802.11, Wi-Fi остаётся самым распространённым и наиболее известным беспроводным протоколом взаимодействия. Его широкое использование в мире IoT в основном ограничено энерго-

потреблением выше среднего по причине удержание качественного сигнала и быстрой передачи данных для лучшего соединения и надёжности. Несмотря на это Wi-Fi является ключевой технологией в развитии и распространении IoT.

Для создания сети Wi-Fi требуются устройства, способные передавать беспроводные сигналы, то есть такие устройства, как телефоны, компьютеры или маршрутизаторы. В домашних условиях маршрутизатор используется для передачи интернет-соединения из общественной сети в частную домашнюю или офисную сеть. Wi-Fi обеспечивает подключение к Интернету близлежащих устройств, находящихся в определенном радиусе действия. Другой способ использования Wi-Fi — создание точки доступа.

Wi-Fi использует радиоволны, которые передают информацию на определенных частотах. Двумя основными частотами являются 2,4 ГГц и 5 ГГц. Оба частотных диапазона имеют ряд каналов, по которым могут работать различные беспроводные устройства, что помогает распределить нагрузку таким образом, чтобы индивидуальные соединения устройств не прерывались. Это в значительной степени предотвращает переполнение беспроводных сетей.

Диапазон в 100 метров является типичным для стандартного Wi-Fi соединения. Однако чаще всего радиус действия ограничивается 10-35 метрами. На эффективное покрытие влияет мощность антенны и частота передачи. Дальность и скорость Wi-Fi подключения к Интернету зависит от окружающей среды и от того, обеспечивает ли оно внутреннее или внешнее покрытие.

Технология Wi-Fi была создана в 1998 году. В 2018 году Wi-Fi Alliance [5] ввёл упрощенную нумерацию поколений Wi-Fi для обозначения оборудования, поддерживающего Wi-Fi 4 (802.11n), Wi-Fi 5 (802.11ac) и Wi-Fi 6 (802.11ax). Эти поколения имеют высокую степень обратной совместимости с предыдущими версиями. Альянс заявил, что уровень поколения 4, 5 или 6 может быть указан в пользовательском интерфейсе при подключении, наряду с уровнем сигнала.

Поколение	Стандарт IEEE	Частота	Скорость передачи	Год
Wi-Fi 0	802.11	2.4 GHz	2 Mbit/s	1997
Wi-Fi 1	802.11b	2.4 GHz	11 Mbit/s	1999
Wi-Fi 2	802.11a	5 GHz	54 Mbit/s	1999
Wi-Fi 3	802.11g	2.4 GHz	54 Mbit/s	2003
Wi-Fi 4	802.11n	2.4/5 GHz	600 Mbit/s	2008
Wi-Fi 5	802.11ac	5 GHz	6933 Mbit/s	2014
Wi-Fi 6	802.11ax	2.4/5 GHz	9608 Mbit/s	2019
Wi-Fi 6E	802.11ax	6 GHz	9608 Mbit/s	2020

Таблица 1.3: Основные поколения Wi-Fi

Основные поколения Wi-Fi представлены в Таблице 1.3. Однако для тех-

нологии IoT особый интерес представляют только некоторые из этих поколений, а именно:

- IEEE 802.11b/g/n. Эти стандарты отличаются относительно небольшим радиусом действия. Они функционируют в полосе частот 2400—2483,5 МГц. В стандарте IEEE 802.11n, выпущенном в 2008 году, скорость соединения была существенно увеличена. Однако новая скорость может быть достигнута лишь в одном из трёх режимов работы, в котором не поддерживается обратная совместимость со стандартами IEEE 802.11b/g. Данные стандарты являются весьма популярными в устройствах для домашней автоматизации, где не всегда нужны огромные скорости передачи данных или большой радиус покрытия, а существенным параметром является энергоэффективность.

- IEEE 802.11ah. Этот протокол беспроводных сетей был опубликован в 2017 году под названием Wi-Fi HaLow. Он использует освобождённые от лицензий полосы частот 900 МГц для обеспечения сетей Wi-Fi с увеличенной дальностью действия по сравнению с обычными сетями Wi-Fi, работающими в диапазонах 2,4 ГГц и 5 ГГц. Он также отличается более низким энергопотреблением, что позволяет создавать большие группы станций или датчиков, которые взаимодействуют для обмена сигналами, поддерживая концепцию Интернета вещей. Благодаря низкому энергопотреблению протокол конкурирует с Bluetooth и имеет дополнительные преимущества в виде более высокой скорости передачи данных и более широкого радиуса действия.

1.3 Сравнение протоколов

	ZigBee	Z-Wave	Wi-Fi	Bluetooth
Стандарт IEEE	802.15.4	802.15.4	802.11	802.15.1
Скорость передачи	250 Kbit/s	100 Kbit/s	300+ Mbit/s	2 Mbit/s
Энергопотребление	Низкое	Низкое	Высокое	Низкое
Частота	2.4 GHz	908.42 MHz	2.4 GHz/5 GHz	2.4 GHz
Топология сети	Ячеистая	Ячеистая	Звезда	Ячеистая

Таблица 1.4: Сравнение основных протоколов IoT

В Таблице 1.4 приведено сравнение основных характеристик трёх подробно рассмотренных протоколов. Кроме того, для полноты картины в таблицу добавлен Bluetooth 5-го поколения. В сравнение не включена дальность действия: говоря о Wi-Fi, радиус непосредственного взаимодействия между двумя устройствами обычно является большим, однако сети на основе ZigBee и Z-Wave, как указано в таблице, имеют ячеистую топологию, за счёт чего

они могут использовать промежуточные устройства для передачи сигнала и увеличения радиуса действия. Что касается энергопотребления, устройства, работающие по Wi-Fi, находясь в активном режиме, способны потреблять в 5 раз больше энергии, чем их аналоги, использующие ZigBee или Z-Wave.

Говоря о технических характеристиках, отличие ZigBee от Z-Wave невелико: Z-Wave выделяется только используемой частотой. Но кроме этого существует разница в распространении устройств на основе обоих протоколов. Z-Wave получил более широкое распространение в США, в то время как ZigBee больше популярен в Европе. Существует ещё одно небольшое отличие, которое заключается в частотных диапазонах. В Северной Америке, Европе, и ряде других стран под Z-Wave и другие протоколы отводятся разные диапазоны частот. Однако в большинстве случаев это не оказывает большого влияния, поскольку вне зависимости от используемой частоты схожие устройства, как правило, обладают одинаковым функционалом.

Немаловажным фактором в сравнении является стоимость конечных устройств. В данном случае она может заметно варьироваться. Говоря, например, о домашней автоматизации, стоимость устройств, поддерживающих Wi-Fi, оказывается выше. Более дешёвая цена устройств на основе ZigBee и Z-Wave достигается за счёт специализированных модулей и чипов.

Но кроме цены комплектующих влияние на стоимость оказывает способ управления конечными устройствами. Гаджеты на основе Wi-Fi могут управляться с любого смартфона, в то время как для управления ZigBee и Z-Wave сетями в подавляющем большинстве случаев требуется некоторое промежуточное устройство: хаб. Хаб позволяет преобразовывать сигналы от устройств в тот же самый Wi-Fi, добавляя возможность управления практически с любого устройства. Хаб имеет дополнительную стоимость. Устройства, совместимые с Wi-Fi, являются более дорогими, поскольку дают возможность обходиться без него.

Наконец, стоит отметить, что для каждого протокола со временем появляется всё больше производителей. В связи с этим встаёт вопрос совместимости между устройствами различных производителей. Не редки случаи, когда несколько устройств не имеют возможности взаимодействия, несмотря на то, что они работают на одном протоколе. При совместном использовании устройств, работающих на различных протоколах, вероятность возникновения проблем совместимости возрастает.

Глава 2
Безопасность сетевых протоколов IoT

2.1 ZigBee

Безопасность работы ZigBee базируется в первую очередь на стандарте IEEE 802.15.4, который реализует протокол. Сам стандарт опирается на правильное управление симметричными ключами и корректную реализацию методов и политик безопасности. Кроме того, модель сети ZigBee уделяет особое внимание соображениям безопасности по причине формирования структуры сети «на лету» (ad-hoc сети), поскольку такие сети могут быть физически доступны для внешних устройств.

В силу своей дешевизны протокол предполагает открытую модель, в которой сетевые уровни доверяют друг другу. Следовательно, криптографическая защита существует только между устройствами, но не между уровнями одного устройствами. Этот факт позволяет переиспользовать ключи на разных уровнях.

Среди других допущений предполагается надёжное хранение симметричных ключей шифрования. Ключи не должны быть доступны вне устройства в незашифрованном виде. Исключение в некоторых случаях составляет конфигурация нового устройства в сети, о чём речь будет идти далее. Из-за дешёвой природы устройств политика безопасности ZigBee не предполагает защиту от физических атак на аппаратное обеспечение. Это означает, что, имея физический доступ к устройству, злоумышленник будет иметь теоретическую возможность извлечь из него ключи безопасности. Наконец, предполагается, что производители устройств будут полностью следовать спецификации, а в устройствах будет присутствовать криптостойкий генератор случайных чисел. Очевидно, что на практике не всегда соблюдаются все из перечисленных выше допущений.

Рассмотрим типы ключей безопасности протокола ZigBee. Напомним, что спецификация протокола описывает только сетевой и прикладной уровни, а более низкие физический и MAC уровни определены в стандарте IEEE 802.15.4. На сетевом уровне используется следующий ключ:

- Сетевой ключ (network key). Это ключ шифрования сообщений между всеми устройствами сети (широковещательных сообщений). Генерируется TC (Trust Center, доверительным центром), в роли которого выступает координатор, случайным образом. Распределяется между всеми подключающимися к сети устройствами и зашифровывается с помощью pre-configured link key (о нём речь пойдёт ниже).

Прикладной уровень располагает бóльшим набором ключей:

- Предустановленный глобальный ключ соединения (pre-configured global link key). Используется для шифрования сетевого ключа в момент его передачи от TC. Данный ключ используется в версии протокола ZigBee 1.0 и является одинаковым для всех узлов в сети. Он может быть предустановлен либо ZigBee, либо производителем устройств. Ключ, предустановленный ZigBee (ZigbeeAlliance09), позволяет устройствам от различных производителей подключаться к одной сети. Он известен заранее всем участникам:

  ```
  5A 69 67 42 65 65 41 6C 6C 69 61 6E 63 65 30 39
  ```

  Ключ, установленный непосредственно производителем, позволяет объединяться в сеть только устройствам от этого производителя. Предустановлен во все устройства, только если не используется pre-configured unique link key.

- Предустановленный уникальный ключ соединения (pre-configured unique link key). В версии ZigBee 3.0 вместо pre-configured global link key для передачи сетевого ключа от TC может быть использован pre-configured unique link key. Данный ключ уникален для каждой пары TC-узел. Один из возможных сценариев использования следующий. Ключ располагается на устройстве в виде QR-кода. При подключении устройства в сеть пользователь считывает QR-код с помощью смартфона, а от смартфона ключ попадает в хаб (координатор). После этого хаб высылает устройству network key, зашифрованный на только что полученном ключе, для дальнейшей коммуникации.

- Ключ соединения доверительного центра (Trust Center Link Key, TCLK). Используется между TC и определённым узлом. Извлекается из pre-configured unique link key, передаётся от TC к узлу, шифруется с помощью network key и pre-configured unique link key. Предназначен для шифрования всех последующих сообщений между TC и узлом, заменяя pre-configured unique link key (однако узел продолжает хранить pre-configured unique link key для возможного повторного соединения в дальнейшем). Не является обязательным для использования ключом в сеансе.

- Ключ соединения приложения (Application Link Key). Используется между парой узлов (без TC). Запрашивается у TC одним из узлов, генерируется TC случайным образом. Шифруется с помощью network key и pre-configured unique link key. Может возникнуть вопрос, зачем для коммуникации двух устройств нужен отдельный ключ, когда уже есть network key. Network key известен всем устройствам в сети, поэтому

любое устройство сможет расшифровать сообщение. Если нужна защищённая коммуникация для двух отдельных устройств, TC выпускает данный ключ. Не является обязательным для использования ключом в сеансе.

ZigBee использует симметричное шифрование AES с длиной ключа 128 бит для реализации своих механизмов безопасности. Контроль целостности осуществляется за счёт имитовставки, которая в стандарте носит название MIC (Message Integrity Code) в целях избежания путаницы с названием уровня MAC.

2.2 Z-Wave

До 2008 года в спецификации Z-Wave не было никаких упоминаний о способах защиты каналов связи. Таким образом, все устройства Z-Wave коммуницировали открыто. Это означало, что любая сеть Z-Wave была доступна извне и взламывать её было не нужно. В 2008 в спецификацию было добавлено понятие шифрования (Z-Wave S0 Security), а в качестве алгоритма шифрования был выбран AES с длиной ключа 128 бит. Это изменение было призвано решить проблему распространения устройств Z-Wave. Однако разработчики не учли мелких деталей на этапе подключения новых устройств.

В 2013 году в спецификации Z-Wave S0 Security была обнаружена уязвимость. В момент первичной инициализации соединения перед началом сеанса передачи данных устройству отправляется ключ шифрования. На тот момент этот ключ представлял собой последовательность из 128 нулевых бит. Таким образом, злоумышленник мог легко подслушать первичный сеанс связи, ключ которого был заранее известен. Далее не составляло труда отследить последующие изменения ключей шифрования. В результате практически каждая Z-Wave сеть оказалась уязвимой.

После этой истории репутация Z-Wave была существенно испорчена. Для решения проблемы в 2016 году появилась улучшенная версии спецификации Z-Wave S2 Security. В ней для первичной выработки ключей используется протокол Диффи-Хеллмана. Однако более детальная информация о конкретном его варианте (на основе эллиптических кривых или конечных полей) и уровне стойкости, а также сроке жизни ключей шифрования в спецификации не раскрывается.

2.3 Wi-Fi

Основными протоколами защиты информации в сетях Wi-Fi являются WEP, WPA, WPA2 и WPA3.

Протокол WEP (Wired Equivalent Privacy) был разработан в 1997 году вместе с первой версией Wi-Fi. Для шифрования использовался алгоритм RC4 со статическим ключом длиной 64 или 128 бит. Некоторое время протокол успешно функционировал и был способен противостоять базовым атакам типа «человек посередине». Недостатки заключались в слабости самого шифра RC4, а также малой длине ключа. Wi-Fi Alliance отказался от использования WEP в 2004 году, официально объявив этот протокол небезопасным.

В 2003 году на замену WEP пришёл протокол WPA (Wi-Fi Protected Access). WPA использует протокол целостности временного ключа (TKIP) с всё той же длиной ключа 64 или 128 бит [6]. Для шифрования использовался тот же самый алгоритм RC4, но синхропосылка была увеличена вдвое: с 24 до 48 бит. TKIP с помощью уникального базового ключа для сеанса связи динамически генерирует новый 128-битный ключ для каждого пакета и таким образом предотвращает типы атак, которые скомпрометировали WEP. Несмотря на все улучшения протокол WPA позиционировался как временная мера для замены уязвимого протокола WEP.

Полноценной же заменой стал протокол WPA2, который появился в использовании с 2004 года. С этого же года началась сертификация, а с 2006 по 2020 год сертификация WPA2 была обязательной для всех новых устройств с торговой маркой Wi-Fi. В качестве замены TKIP был внедрён протокол блочного шифрования с имитовставкой и режимом сцепления блоков и счётчика: CCMP. TKIP использовался только для обратной совместимости. CCMP основан на алгоритме шифрования AES с длиной ключа 128 бит. Одним из преимуществ по сравнению со старыми версиями является генерация ключей шифрования во время соединения.

В январе 2018 года Wi-Fi Alliance объявил WPA3 в качестве замены WPA2. Сертификация началась в июне 2018 года. Самое крупное изменение связано с новым методом аутентификации. SAE (Simultaneous Authentication of Equals) явился заменой PSK (Pre-Shared Key), который использовал 4-этапное установление связи в протоколе WPA2. SAE работает на основе предположения о равноправности устройств, вместо того, чтобы считать одно устройство отправляющим запросы, а второе — предоставляющим право на подключение. Кроме того, SAE обеспечивает защиту от «чтения назад». 128-битное шифрование осталось минимально допустимой нормой, а для промышленных масштабов предложен вариант использования AES с длиной ключа 256 бит в режиме GCM с SHA-384 в качестве HMAC для контроля целостности информации. С выходом WPA3 появились две новые сопутствующие технологии: Easy Connect и Enhanced Open. Первая позволяет сделать процесс добавления новых устройств в сеть более простым и является весьма актуальной для домашней автоматизации. Отныне у каждого устройства будет уникальный QR-код, который сможет работать в качестве открытого ключа. Для добавления устройства необходимо будет просканировать код при помощи смартфона, который уже находится в сети. После сканирования

между устройством и сетью произойдёт обмен ключами аутентификации для установления последующего соединения. Вторая технология усиливает защиту пользователей в открытых сетях, где по умолчанию нет защиты с аутентификацией. Данные технологии не зависят напрямую от WPA3, но улучшают безопасность для определённых типов сетей. Таким образом, вместо полной переработки безопасности Wi-Fi, WPA3 концентрируется на новых технологиях, которые позволят устранить уязвимости, обнаруженные в WPA2.

2.4 Сравнение безопасности

Схожие в техническом плане ZigBee и Z-Wave, с точки зрения безопасности также не имеют существенных различий: оба протокола используют симметричный алгоритм блочного шифрования AES с длиной ключа 128 бит. Единственным отличием является метод распределения ключей: в Z-Wave используется протокол Диффи-Хеллмана, в то время как в ZigBee этот процесс по-прежнему доверен центру управления безопасностью.

В сетях на основе Wi-Fi безопасности уделяется значительно больше внимания в силу их повсеместного распространения. В контексте домашней автоматизации и некоторых других сфер, в которых Wi-Fi конкурирует с ZigBee и Z-Wave, отличия всё также незначительны. Для шифрования используются дополнительные надстройки и протоколы, но в основе лежит AES-128. В случае промышленного использования для Wi-Fi применяются более продвинутые технологии защиты информации.

Глава 3
Криптографические угрозы и атаки

3.1 ZigBee

Поскольку во всех трёх решения (ZigBee, Z-Wave, Wi-Fi) в том или ином виде используется алгоритм AES, шифрование данных оказывается весьма надёжным. Наиболее подверженным угрозам является этап подключения нового устройства в сеть. Рассмотрим этот этап для протокола ZigBee более подробно.

1. Устройство посылает в эфир запрос Beacon Request.
2. В ответ ближайшие роутеры (или координатор в случае их отсутствия) отправляют Beacon Resposnse.
3. Устройство выбирает роутер с наилучшими радио характеристиками и отправляет ему Association Request.
4. Роутер назначает новому устройству сетевой адрес и сообщает его в сообщении Association Response.
5. Роутер также информирует координатора о новом устройстве в сети.
6. После этого координатор посылает на устройство network key (зашифрованный, как известно из предыдущего раздела с помощью preconfigured link key).

Именно этот этап является самым уязвимым во всём сеансе сообщений. Основная задача злоумышленника — перехватить network key, так как с его помощью открываются возможности для чтения показателей датчиков, а также для отправки команд, которые будут считываться конечными устройствами. В случае, когда для присоединения новых устройств используется preconfigured global link key, заранее известный всем участникам, злоумышленнику достаточно прослушивать сеть в момент подключения. И когда координатор отправит новому устройству network key, зашифрованный на известном ключе, злоумышленнику останется только расшифровать его и использовать в своих целях.

Можно считать, что эта уязвимость осознанно оставлена разработчиками протокола, так как временное окно для подключения новых устройств обычно очень маленькое. При этом сохраняется совместимость между устройствами от различных производителей, а также удобство для потребителей, которым не нужно, например, самостоятельно записывать ключи шифрования на

устройства. Однако уязвимость остаётся, и у злоумышленника всегда будет шанс ей воспользоваться. Кроме того, он может попытаться заставить уже имеющиеся в сети устройства переподключаться, не дожидаясь тем самым момента для прослушивания сети, а инициируя его самостоятельно.

Стоит отметить, что при использовании версии ZigBee 3.0 и pre-configured unique link key (ключа, уникального для каждого устройства), описанная выше уязвимость исчезает, и весь протокол становится безопасным. Но существует ещё много устройств, работающих на более старых версиях, для которых риск взлома по-прежнему остаётся.

Несколько вариантов атак на предыдущие версии протокола ZigBee детально описаны в [7].

3.2 Z-Wave

Многие устройства домашней автоматизации напрямую влияют на нашу безопасность. К таким устройствам можно отнести, например, автоматизированные дверные замки. В случае их компрометации последствия могут оказаться весьма неудачными. И такой случай имел место. В зашифрованных алгоритмом AES дверных замках на основе протокола Z-Wave была обнаружена ранняя уязвимость. Используя её, злоумышленник получал возможность удалённо отпирать двери без знания ключей шифрования. А после изменения ключей последующие сетевые сообщения, например, «дверь открыта», игнорировались установленным контроллером сети. Уязвимость не была связана с недостатком в спецификации Z-Wave, а являлась ошибкой в реализации, допущенной производителем дверного замка.

Безопасность протокола Z-Wave была улучшена в 2016 году с появлением спецификации Z-Wave S2 Security. Однако из-за обратной совместимости с предыдущей версией спецификации S0 устройства оказались по-прежнему уязвимы в процессе подключения. В Z-Wave S0 Security использовался статический первичный ключ, состоящий из 128 нулевых бит, из-за чего дальнейший взлом и управление устройствами оказывались весьма тривиальными. В качестве улучшения в S2 для первичной выработки ключей используется протокол Диффи-Хеллмана, а также дополнительно может потребоваться ввод 5-символьного кода устройства. Но в 2018 году была обнародована атака, которая позволяла понижать версию спецификации с S2 до S0 и эксплуатировать старые уязвимости.

Рассмотрим более подробно процесс понижения версии. Первые шаги при сопряжении устройства и контроллера в спецификациях S0 и S2 аналогичны и заключаются в следующем:

1. На контроллере (управляющем устройстве) необходимо выбрать режим добавления нового устройства.

2. Далее пользователь нажимает кнопку (или последовательность кнопок) на присоединяемом устройстве.

3. После этого новое устройство посылает в сеть информацию о себе (Node info).

4. Контроллер получает эту информацию и приступает к процессу выработки ключей.

Единственным отличием в полезной нагрузке Node info для устройств на основе S2 Security является поддержка класса команды 0x9F – COMMAND_ _CLASS_SECURITY_2. Стоит отметить, что вся информация в Node info является незашифрованной. Таким образом, активный атакующий может убрать соответствующую команду из Node info и отправить подделанные данные контроллеру. Контроллер посчитает, что устройство не поддерживает спецификацию S2 Security, и будет выбрана уязвимая к атакам спецификация S0. Справедливо заметить, что в этом случае контроллер должен выдать предупреждение пользователю об использовании более ранней версии, однако данное предупреждение, как правило, игнорируется. Подделанный Node info должен содержать тот же Home ID, что и у оригинального устройства. Поле Home ID не является константным, а генерируется каждый раз при добавлении или перезагрузке устройства. Это значит, что злоумышленник сперва должен завладеть Home ID. Эта задача выполнима и требует лишь пересчитывания длины сообщения и контрольной суммы после удаления команды о поддержке S2.

Резюмируя, можно сказать, что данная атака подчёркивает проблему множества протоколов, а именно проблему улучшения безопасности при необходимости поддержания старых устройств. Стоит отметить, что уже подключённые в сеть с использованием S2 Security и успешно функционирующие устройства находятся в относительной безопасности, однако новые устройства остаются подвержены уязвимости.

3.3 Wi-Fi

WPA2 являлся основным стандартом шифрования для Wi-Fi на протяжении 14 лет: с 2004 по 2018 год. Однако в 2016 году на WPA2 была разработана атака с переустановкой ключа (Key Reinstallation Attack, Krack). Krack — это атака повторного воспроизведения [8]. Многократно сбрасывая одноразовый код, передаваемый на третьем этапе установки соединения WPA2, злоумышленник может постепенно сопоставить зашифрованные пакеты, сохранённые ранее, и узнать ключ шифрования. Уязвимость проявляется в самом стандарте Wi-Fi и не связана с ошибками реализации. В связи с этим любая корректная реализация WPA2 вероятнее всего является уязвимой. Уязвимость затрагивает основные программные платформы.

При подключении нового клиента к сети Wi-Fi с защитой по WPA2 общий ключ шифрования согласуется за 4 этапа. Данный ключ служит для шифрования всех пакетов данных. Однако, из-за потери отдельных сообщений точка доступа (роутер) может повторно отправлять сообщения третьего этапа до подтверждения о его получении. Каждый раз при получении подобного сообщения клиент устанавливает уже имеющийся ключ шифрования. На практике злоумышленник заставляет жертву выполнить переустановку ключа.

Благодаря повторному использованию ключа шифрования появляется возможность воспроизведение пакетов, расшифрования и подделки содержания. При определённых условиях злоумышленник способен осуществлять атаки типа «человек посередине».

С появлением стандарта WPA3 данная уязвимость была устранена благодаря введению SAE (Simultaneous Authentication of Equals).

3.4 Матрица угроз

В данном разделе было решено сравнить устойчивость выбранных технологий к некоторому общему набору угроз в целях выявления наиболее криптостойкого решения.

1. Атака «человек посередине» (Man in the middle, MITM). Злоумышленник ретранслирует и изменяет сообщения между участниками протокола.

2. Атака повторного воспроизведения (Replay attack). Злоумышленник записывает сообщения из одного сеанса протокола в целях воспроизведения их в другом сеансе, выдавая себя за одного из участников.

3. Защита от «чтения назад» (Perfect forward secrecy, PFS). Компрометация долговременных ключей не должна приводить к компрометации предыдущих сеансовых ключей.

4. Атака понижения версии (Downgrade attack). Злоумышленник принуждает устройства к использованию более старых и, соответственно, менее защищённых версий протокола.

В отношении протокола ZigBee:

- При присоединении новых устройств здесь не используется протокол Диффи-Хеллмана (в отличие от, например, Z-Wave). Вследствие использования первоначального статического ключа, а также ключей, генерируемых координатором сети, данные первого же сообщения оказываются зашифрованными. Задача распределения ключей здесь решена другим способом (см. раздел про безопасность ZigBee).

- Для предотвращения от атак повторного воспроизведения в протоколе реализован специальный счётчик как часть процесса шифрования и аутентификации сообщений. С каждым новым пакетом данных значения счётчика увеличивается. Поскольку все пакеты зашифрованы, значение счётчика, как правило, не удаётся подменить. Однако в особых случая (компрометация сетевого ключа или долговременное использования сеансовых ключей без смены их координатором) теоретическая возможность данной атаки остаётся. Попытки использования этой возможности описаны в [7] и [9].

- В ZigBee долговременным можно считать лишь pre-configured link key. Даже при выпуске дополнительных ключей этот ключ продолжает храниться на устройстве на случай переприсоединения или других нестандартных ситуаций. Теоретически его компрометация возможна только в случае физического доступа к устройству. Тогда появляется возможность получения других ключей, которые шифруются на нём, а вместе с ними и доступа к зашифрованным сообщения сеанса. В этом случае свойство PFS может не соблюдаться, однако даже здесь всё зависит от координатора сети и его действий по перевыпуску сеансовых ключей. В общем случае, при использовании последней версии протокола, компрометации pre-configured link key исключается.

- Версии ZigBee принципиально отличаются с позиции подхода к распределению ключей. В разных версиях pre-configured link key встраивается в устройства по-разному, о чём подробно шла речь в разделе про безопасность ZigBee. В связи с этим у злоумышленника отсутствует возможность понизить версию, если на устройствах уже используется более современная версия протокола.

Рассмотрим те же пункты касательно протокола Z-Wave:

- Как и в случае с ZigBee, процесс подключения новых устройств оказывается самым уязвимым. Для распределения ключей здесь, в отличие от ZigBee, используется протокол Диффи-Хеллмана. Для того, чтобы противостоять MITM-атаке, при реализации протокола используются различные модификации. Однако в [10] приводится успешная атака несмотря на все методы защиты. При этом атака может быть выполнена только в определённый временный промежуток и лишь при использовании одного из трёх возможных методов аутентификации в процессе подключения нового устройства. Поэтому есть большой шанс, что уязвимость будет устранена в ближайшем будущем.

- При использовании S2 Security атака повторного воспроизведения на протокол Z-Wave невозможна.

- Выше была подробна описаны атака понижения версии на протокол Z-Wave в процессе подключения в сеть нового устройства. До тех пока, будет обеспечиваться обратная совместимость с предыдущими уязвимыми версиями, вопрос защиты от атак подобного типа будет оставаться открытым.

Наконец посмотрим на те же угрозы применительно к Wi-Fi:

- В последних обновлениях WPA3 была представлена технология OCV (Operating Channel Validation), предотвращающая атаки типа «Человек посередине» и повторного воспроизведения. Основным же минусом является тот факт, что роутеры с поддержкой Wi-Fi 6 и WPA3 только начинают появляться на рынке, а уже имеющиеся решения весьма велики в цене.

- Протокол WPA2 спроектирован таким образом, что свойство PFS не может быть соблюдено. Здесь не используется криптография с открытым ключом, и у злоумышленника есть возможность сделать всё необходимое для чтения данных при компрометации долговременного ключа (пароля) в будущем. В WPA3 работают над невозможностью для злоумышленника записывать весь трафик между точкой доступа и устройством с целью расшифровки его в дальнейшем.

- Версия WPA3 тесно связана с WPA2 в контексте обратной совместимости, поэтому атаки, направленные на понижение версии, всё ещё имеют место быть. Единственным надёжным способом защиты от них может быть запрет на подключение в сеть WPA3 устройств с более слабым уровням защиты. Основная угроза, как и в других протоколах, появляется в момент рукопожатия (присоединения нового устройства в сеть).

В сводной сравнительной таблице символом «+» обозначено наличие защиты в данном протоколе от соответствующей угрозы, «−» − отсутствие защиты, а «∼» − зависимость от версии протокола и прочих условий.

	ZigBee	Z-Wave	Wi-Fi
«Человек посередине»	+	∼	+
Атака повторного воспроизведения	∼	+	+
Защита от «чтения назад»	∼	∼	∼
Атака понижения версии	+	−	∼

Таблица 3.1: Матрица угроз IoT протоколов

Глава 4
Разработка собственного решения с использованием белорусской криптографии

4.1 Поиск существующих имплементаций

Первой задачей практической части было нахождение существующих реализаций подробно описанных выше протоколов с целью дальнейшей работы над криптографическим аспектом этих реализаций. В качестве основного протокола для модификаций был выбран протокол ZigBee. Были поставлены следующие задачи:

- поиск имплементации протокола ZigBee, позволяющей вносить модификации;
- поиск устройств (микроконтроллеров), способных работать на этой имплементации;
- изменение или полная замена криптографической составляющей в имплементации.

К сожалению, открытых реализация оказалось немного. Практически не было найдено библиотек, реализующих в полной мере последнюю версию протокола. Проблема заключается в том, что сама спецификация находится в закрытом доступе. Для получения спецификации необходимо стать членом ZigBee Alliance, что осуществляется на платной основе. Аналогично, все имплементации протокола ведущими технологическими компания также являются закрытыми. Это связано с коммерческой составляющей, поскольку компании получают прибыль, реализуя устройства на собственных прошивках.

По совокупности вышеописанных факторов был выбран другой подход, который не привязан к определённому протоколу. Суть данного подхода заключается в самостоятельной реализации криптографического уровня защиты и применении его поверх установленного соединения между управляющим устройством (хабом) и конечным устройством. В качестве конечного устройства в данной работе был выбран прототип умной лампочки.

4.2 Выбор компонентов и технологий для реализации

Обновлённые практические задачи были сформулированы следующим образом:

- выбор микроконтроллера, который будет служить прототипом умного устройства, с возможностью его программирования и прошивки;
- установка соединения между управляющим и умным устройствами. Для простоты в качестве управляющего устройства в данной работе используется компьютер;
- разработка кода (прошивки) для умного устройства (контроллера), а также клиентского приложения для управляющего устройства;
- реализация защищённого обмена сообщениями с использование белорусского криптографического стандарта СТБ 34.101.77;
- модификация стандарта с изменением значения некоторых его параметров;
- оценка стойкости видоизменённого решения.

В качестве микроконтроллера был выбран ESP8266 (модель NodeMCU V3). Это недорогая модель от китайской компании Espressif Systems. Её большим преимуществом является встроенный Wi-Fi модуль. Помимо поддерживаемых протоколов Wi-Fi 802.11 b/g/n и режимов работы как в качестве точки доступа, так и клиента, микроконтроллер отличается наличием встроенного стека TCP/IP.

Для программирования данной модели существует широкий выбор языков, платформ и сред, среди которых Arduino IDE, Espressif IoT Development Framework (официальный фреймворк от разработчика) и многие другие. В данной работе был выбран инструмент PlatformIO [11]. Это кроссплатформенная интегрированная среда разработки, построенная на основе редактора Microsoft Visual Studio Code, а также встроенный отладчик, статический анализатор кода и система сборки. Платформа поддерживает большое количество микроконтроллеров, среди которых в том числе есть ESP8266 NodeMCU. На сайте также представлен широкий выбор библиотек и примеров с кодом. Языком разработки на платформе является C++.

При выборе инструментов и технологий для разработки также рассматривались варианты использования более высокоуровневых языков программировая, таких как Java [12] или .NET [13]. Однако наличие примеров, библиотек и большого сообщества разработчиков стало определяющим фактором при выборе PlatformIO. При этом для клиентского веб-приложения на

управляющем устройстве (компьютере) был выбран язык программирования Java.

4.3 Работа с микроконтроллером ESP8266

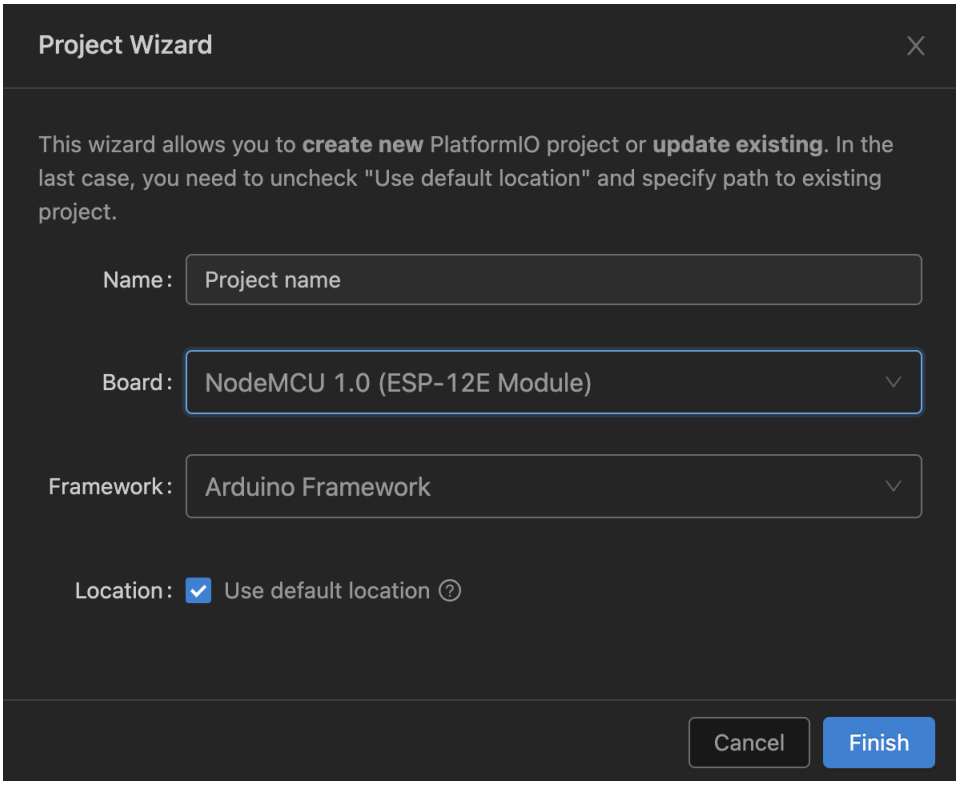

Рисунок 4.1: Меню создания нового проекта в PlatformIO

Первым шагом при программировании микроконтроллера стало создание проекта при помощи PlatformIO. На этапе создания нового проекта появляется возможность выбора контроллера. Основными составляющими в проекте являются конфигурационный файл platformio.ini, в котором указывается модель контроллера, а также подключаемые к проекту библиотеки, и директива src, в которой должен содержаться код проекта.

При первоначально настройке в папке src содержится единственный файл main.cpp. Он включает два метода. Метод setup() отвечает за первоначальную конфигурацию контроллера, а метод loop() повторяется всё время, пока контроллер подключен к питанию.

Для знакомства с программированием данного типа контроллеров и изучения программных методов было реализовано несколько базовых примеров. Первым из них было мигание встроенного на контроллере индикатора. Затем были построены простейшие схемы с использованием макетной платы, диода и нескольких проводов. Мигание встроенного индикатора было заменено миганием диода. После этого был реализован пример управления диода

по нажатию кнопки. Наконец, был задействован Wi-Fi модуль для управления диодом с компьютера. Данный пример был взят за основу практического прототипа. Более подробно работа прототипа и все этапа обмена сообщениями будут рассмотрены позже. Но перед этим перейдём к криптографической части данной работы.

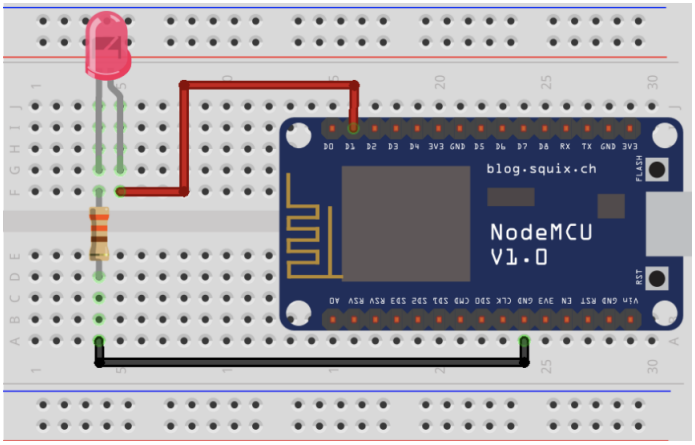

Рисунок 4.2: Схема подключения диода к микроконтроллеру

4.4 Реализация и использование криптографического стандарта СТБ 34.101.77

4.4.1 Описание стандарта

Официальное название стандарта — «Криптографические алгоритмы на основе sponge-функции» [14]. Разберём для начала, что из себя представляет sponge-функция.

Sponge-функция — это класс алгоритмов с конечным внутренним состоянием. Эти алгоритмы предназначены для обеспечения конфиденциальности, целостности и подлинности информации при её передачи и обработке. Сама функция задаёт сложное преобразование двоичных слов большой длины. Криптографические алгоритмы подразделяются на две большие группы:

1. алгоритмы хэширования, описанные в разделе 7 стандарта;
2. программируемые алгоритмы, описанные в разделе 8.

В практической части данной работы используются только программируемые алгоритмы, в частности алгоритмы аутентифицированного шифрования, которые представляют собой последовательности команд криптографического автомата. Более подробно о них речь пойдёт далее.

Стандарт описывает преобразование двоичных слов длины 1536 бит (192 байта). Преобразование задаётся алгоритмом bash-f, который в свою очередь использует алгоритм bash-s. bash-f и bash-s являются sponge-функциями, которые определяют программируемые алгоритмы. Стандартные уровни стойкости $l = 128, 192, 256$. Дополнительным параметром является ёмкость $d \in \{1, 2\}$. В программируемых алгоритмах также используется ключ K, длина которого должна быть не меньше уровня стойкости, но также не превосходить 480. Для контроля целостности и подлинности вычисляется имитовставка, длина которой равна уровню стойкости l.

Управление автомата, лежащего в основе программируемых алгоритмов, осуществляется командами. Допустимы следующие команды:

- start (инициализация). На этом этапе происходит загрузка ключа;
- restart (повторная инициализация);
- absorb (загрузка данных);
- squeeze (выгрузка данных). С помощью этой команды осуществляется вычисление имитовставки;
- encrypt (операция зашифрования);
- decrypt (операция расшифрования);
- ratchet (необратимое изменение состояния автомата);
- commit (подтверждение выполнения других комманд).

В практической части данной работы были реализованы команды start, absorb, squeeze, encrypt, decrypt, commit. Все перечисленные команды используются в аутентифицированном шифровании, которое и применяется для защиты данных, передаваемых между умными устройствами. Алгоритм установки защиты заключается в последовательном применении команд start, absorb, encrypt, squeeze, а алгоритм снятия защиты выполняет команды start, absorb, decrypt и squeeze соотвественно.

4.4.2 Практическая реализация

Алгоритмы bash-s и bash-f являются низкоуровневыми, поскольку оперируют с битами. Поэтому они были реализованы на языке программирования C. Так как язык прошивки микроконтроллера — C++, а язык клиентского приложения — Java, команды для управления автоматом при аутентифицированном шифровании необходимо было реализовать на двух этих языках. Рассмотрим сначала реализацию на Java.

Практически во всех командах используется алгоритм bash-f, в связи с чем возникла необходимость вызова нативного C кода из программы на Java. Для этого используется технология JNI (Java Native Interface). Однако перед этим необходимо скомпилировать код на C в динамическую библиотеку. Рассмотрим более детально шаги для вызова нативного кода из программы на Java:

1. Для начала нужно создать Java класс с методом, объявленным как native:

```
class LibraryNative {
    public static native byte[] bash_f(byte[] array);
}
```

2. После этого скомпилировать файл с опцией -h для генерация header-файла:

```
javac LibraryNative.java -h .
```

3. Создать файл LibraryNative.c. Скопировать определение метода из файла LibraryNative.h и добавить реализацию:

```
#include <jni.h>
#include <inttypes.h>
#include <string.h>
#include "LibraryNative.h"

JNIEXPORT jbyteArray JNICALL Java_LibraryNative_bash_1f(JNIEnv *env,
   jclass thisClass, jbyteArray inJNIArray) {
    // insert implementation here
}
```

4. Сгенерировать динамическую библиотеку. Практическая часть данной работы выполнены на операционной системе MacOS, поэтому генерируется файл с расширением .dylib:

```
gcc -I"$JAVA_HOME/include" -I"$JAVA_HOME/inc lude/darwin" -dynamiclib -o
    libLibraryNative.dylib LibraryNative.c
```

5. Запустить метод из программы на Java, предварительно загрузив библиотеку в явном виде:

```
public class Runner {

    static {
        System.loadLibrary("LibraryNative");
    }

    public static void main(String[] args) {
```

```
8           LibraryNative.bash_f(new byte[192]);
9       }
10  }
```

Далее были реализованы команды, а также методы для установки и снятия защиты в аутентифицированном шифровании строго в соответствии со стандартом.

Реализация стандарта на C, как и других белорусских криптографических стандартов, содержится в библиотеке bee2. Проблема заключается в несовместимости этой библиотеки с платформой PlatformIO, на базе которой написана прошивка для микроконтроллера. В связи с этим возникла необходимость реализовать часть стандарта, ответственную за шифрование, самостоятельно. За основу была взята имплементация из библиотеки bee2.

Также в коде были реализованы юнит тесты с данными из приложения А соответствующего стандарта для верификации корректности реализации алгоритмов и команд.

4.5 Модель прототипа

Результатом работы над практической частью является реализация протокола взаимодействия между двумя умными устройствами с применением белорусского криптографического стандарта. Протокол включает в себя защищённый обмен сообщениями. Рассмотрим более детально взаимодействие конечного устройства (умной лампочки на базе микроконтроллера ESP8266) и управляющего устройства (компьютера).

На этапе присоединения конечное устройство подключается к сети Wi-Fi, в которой уже находится управляющее устройство. После этого на конечном устройстве запускается упрощённый веб-сервер (с выделенным статическим IP адресом), который ожидает команды от управляющего устройства. Клиент посылает HTTP POST запросы на включение или выключение лампочки по REST API. При этом конечное устройство принимает только корректно зашифрованные запросы и не реагирует на все остальные.

Для осуществления подключения умного устройства к Wi-Fi сети используется специализированная библиотека WiFiManager. Процесс подключения происходит по следующей схеме:

1. При запуске Wi-Fi модуль на микроконтроллере работает в режиме программной точки доступа с предварительно заданным именем сети и паролем.

2. На управляющем устройстве (компьютере) необходимо подключиться к соответствующей Wi-Fi сети, после чего откроется страница со всеми доступными Wi-Fi сетями.

3. В списке необходимо выбрать нужную сеть и ввести пароль.

4. С этого момента умное устройство будет находится в выбранной сети в качестве Wi-Fi клиента.

Описанные действия необходимо осуществить единожды — при первом подключении устройства в сеть. В дальнейшем контроллер будет подключаться к сети автоматически при её наличии в зоне доступа.

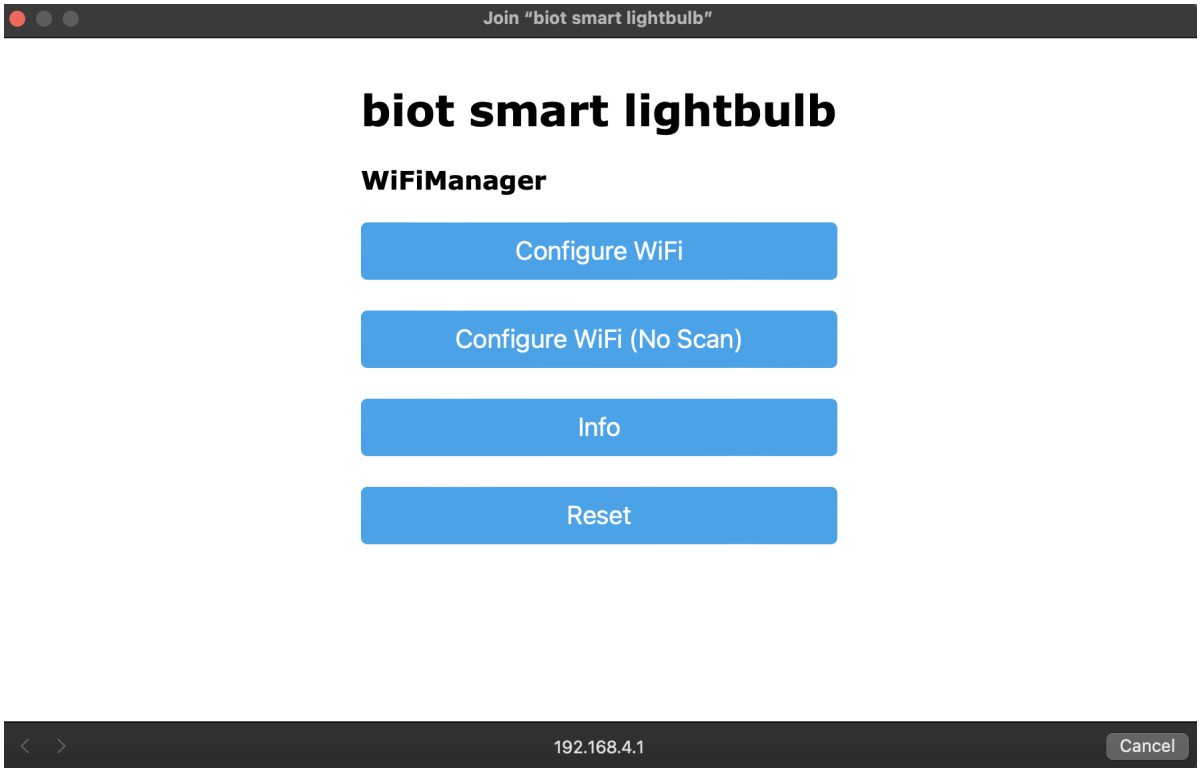

Рисунок 4.3: Меню подключения умного устройства к Wi-Fi сети

Основной проблемой при подключении любого умного устройства является распределение ключей шифрования. В практической части данной работы этот процесс осуществляется по аналогии с последней версией протокола ZigBee. Каждое конечное устройство имеет свой уникальный ключ шифрования. Предполагается наносить закодированный ключ на корпус устройства в виде QR кода. При первом подключении устройства необходимо считать QR код смартфоном и передать его на компьютер. Так управляющее устройство узнаёт о ключе. Таким образом, для каждой пары «управляющее устройство — конечное устройство» ключ шифрования будет уникальным.

Рисунок 4.4: Пример QR кода, содержащего ключ шифрования

Для клиента было разработано веб-приложение, отображающее текущее состояние лампочки и позволяющее изменить его.

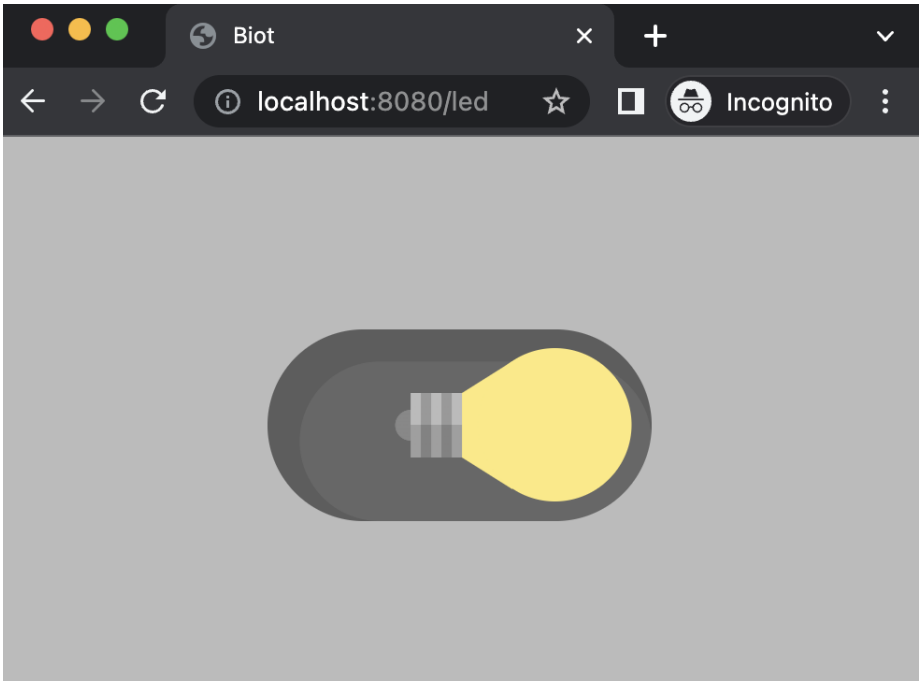

Рисунок 4.5: Интерфейс веб-приложения

При изменении положения переключателя лампочка меняет своё состояние. Прототип умного устройства состоит из диода, имитирующего лампочку, микроконтроллера, макетной платы и нескольких проводов, соединяющих всё в единое целое. Прототип получает питание через кабель от компьютера или внешнего аккумулятора.

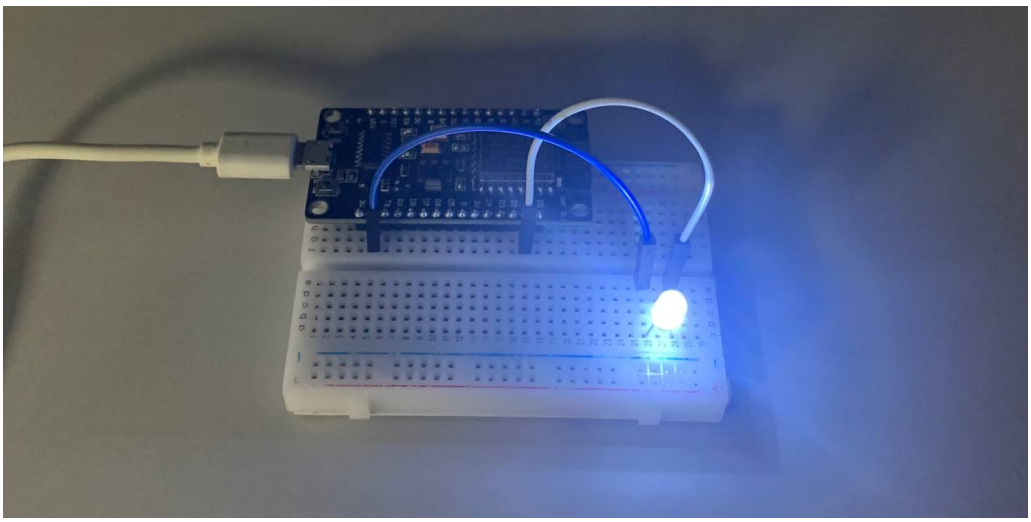

Рисунок 4.6: Прототип умной лампочки

Для демонстрации работы шифрования был проведён эксперимент с перехватыванием данных, отправляемых от компьютера к лампочке по Wi-Fi сети. Для осуществление эксперимента использовался анализатор трафика Wireshark. На рисунке 4.7 приведён пример незашифрованной команды по выключению лампочки, а на рисунке 4.8 — пример этой же команды, но уже в зашифрованном виде.

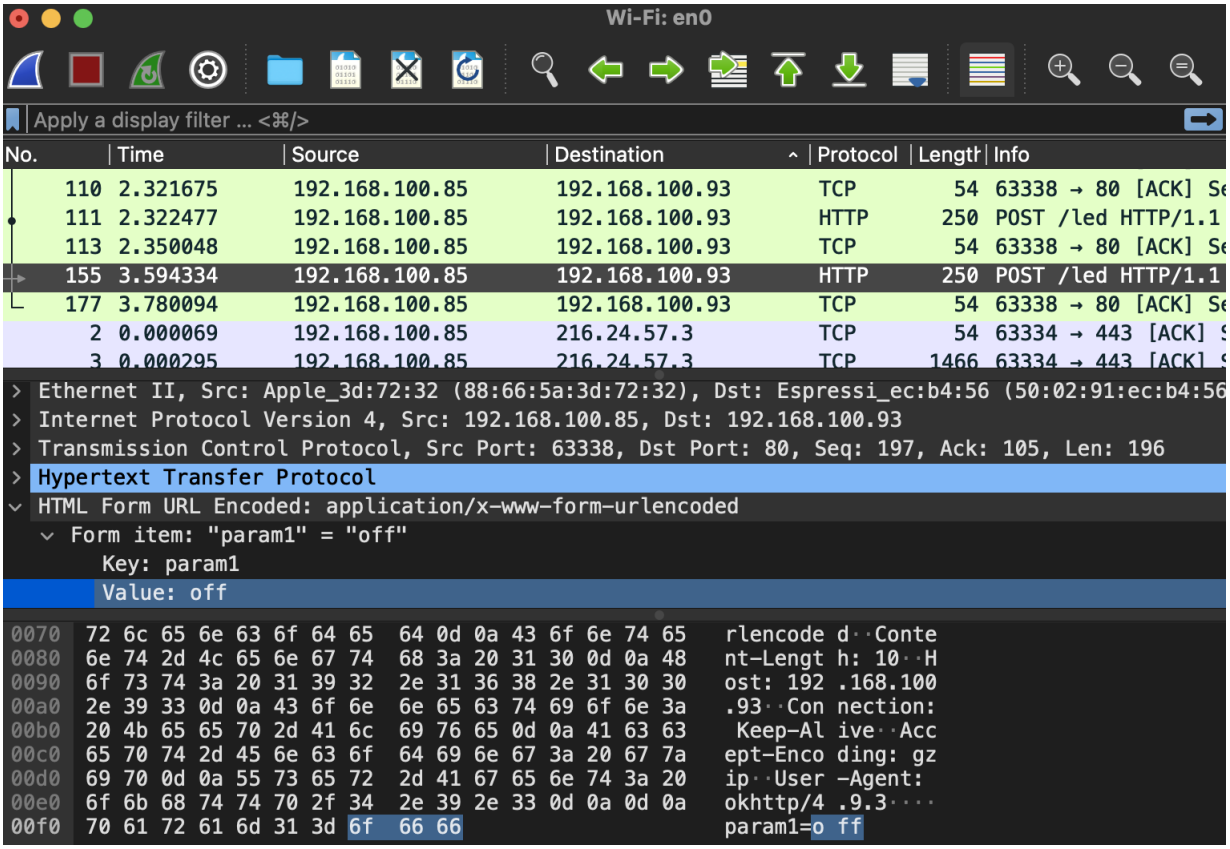

Рисунок 4.7: Пример незашифрованной команды

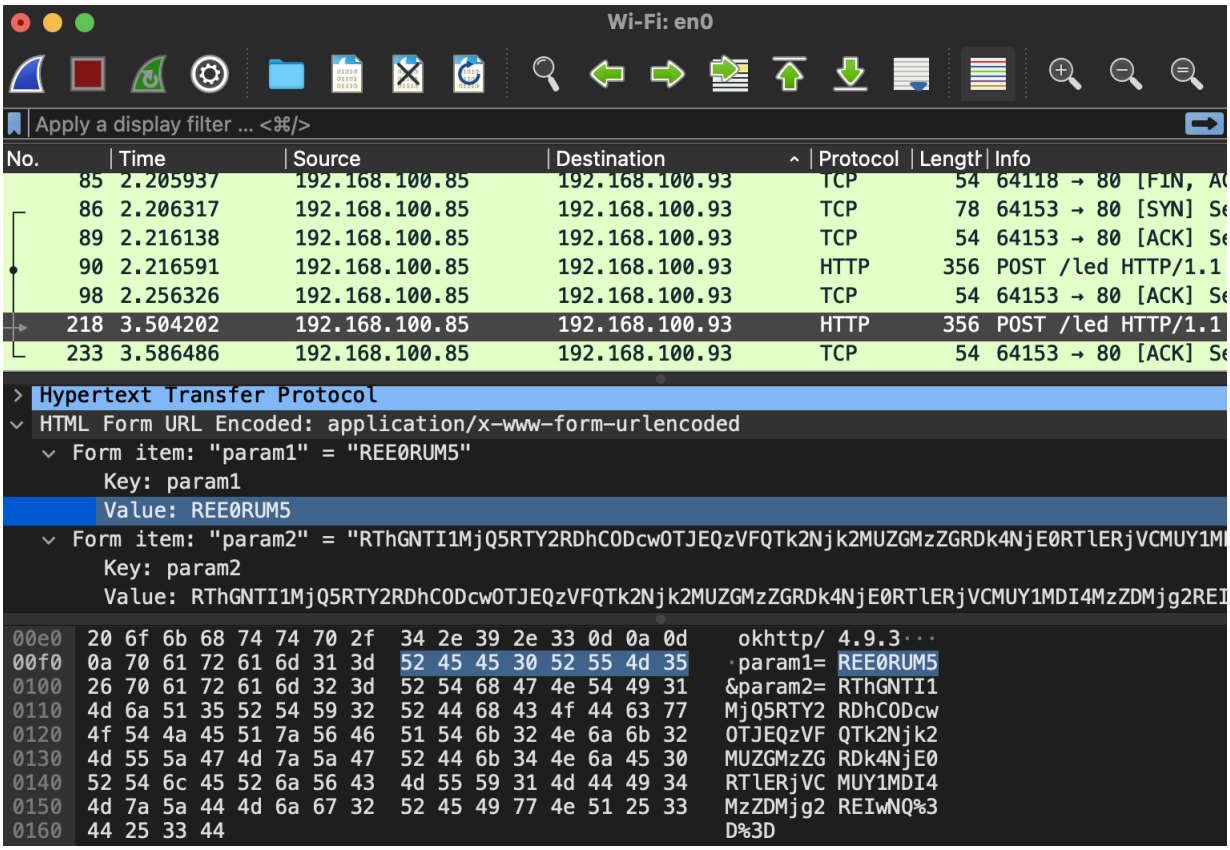

Рисунок 4.8: Пример зашифрованной команды

Заключение

Интернет вещей всё глубже проникает в жизнь конечного потребителя, набирая всё большую популярность. В связи с этим вопросы обеспечения безопасности конечных устройств и криптографической защиты данных пользователей остаются открытыми и актуальными.

В первой главе текущего исследования были описаны ключевые технологии, применимые в IoT. Основными технологическими уровнями являются программное и аппаратное обеспечение, уровень коммуникации между устройства и платформа, которая их объединяет. Среди большого разнообразия используемых протоколов были выбраны три основных решения: ZigBee, Z-Wave, Wi-Fi — и проведён сравнительный анализ их технических характеристик.

Вторая глава продолжает описание и сравнение выбранных протоколов уже с точки зрения безопасности. Были рассмотрены алгоритмы выработки и распределения ключей, шифрования и контроля целостности данных, используемые в протоколах.

Третья часть рассказывает об угрозах, свойственных сетям на основе ZigBee, Z-Wave и Wi-Fi, а также содержит описание успешно проведённых атак, с целью выявления уязвимостей и дальнейшего их устранения. В соответствии с поставленными во введении целями была построена матрицы угроз, которая отображает потенциальную уязвимость IoT протоколов к определённым типам криптографических атак. В качестве набора базовых угроз были выбраны следующие: атака «человек посередине», атака повторного воспроизведения, защита от «чтения назад», атака понижения версии.

В заключительной главе приведено описание процесса разработки прототипа, выбора технологий, обзор белорусского криптографического стандарта СТБ 34.101.77, мотивация использования и реализации аутентифицированного шифрования из этого стандарта. Также были детально описаны ключевые технические особенности разработанного решения, такие как установка соединения, выработка и распределение ключей шифрования, реализация счётчика отправленных и полученных сообщений. Код прошивки для умного устройства, а также ключевых компонентов приложения для управляющего устройства приведен в приложениях к данной работе.

Литература

1. List of ZigBee certified products. -Mode of access: https://zigbeealliance.org/product_type/certified_product/. -Date of access: 13.12.2021.

2. ZigBee Alliance. -Mode of access: https://zigbeealliance.org/solution/zigbee/. -Date of access: 13.12.2021.

3. List of Z-Wave certified products. -Mode of access: https://products.z-wavealliance.org. -Date of access: 13.12.2021.

4. Z-Wave Alliance. -Mode of access: https://z-wavealliance.org/. -Date of access: 13.12.2021.

5. Wi-Fi Alliance. -Mode of access: https://www.wi-fi.org/. -Date of access: 13.12.2021.

6. IEEE 802.11i, IEEE Computer Society, 2004.

7. Ján Ďurech, Mária Franeková: Security attacks to ZigBee technology and their practical realization. SAMI, Herl'any, Slovakia, 2014.

8. Common Vulnerabilities and Exposures: WPA2 reinstallation key. -Mode of access: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-13077. -Date of access: 13.12.2021.

9. Security Analysis of Zigbee. -Mode of access: https://courses.csail.mit.edu/6.857/2017/project/17.pdf. -Date of access: 20.03.2022.

10. Formal Proof of a Vulnerability in Z-Wave IoT Protocol. -Mode of access: https://www.scitepress.org/Papers/2021/105533/105533.pdf. -Date of access: 21.03.2022.

11. PlatformIO official documentation. -Mode of access: https://docs.platformio.org/en/latest/. -Date of access: 12.05.2022.

12. MicroEJ Software Development Kit. -Mode of access: https://developer.microej.com/microej-sdk-software-development-kit/. -Date of access: 12.05.2022.

13. .NET nanoFramework. -Mode of access: https://www.nanoframework.net/. -Date of access: 12.05.2022.

14. Информационные технологии и безопасность. Криптографические алгоритмы на основе sponge-функции / СТБ 34.101.77-2020.

ПРИЛОЖЕНИЕ А

Исходный код реализации алгоритмов bash-s и bash-f из стандарта СТБ 34.101.77 на языке программирования C

```c
#include <jni.h>
#include <inttypes.h>
#include <string.h>
#include "by_bsu_biot_library_LibraryNative.h"

typedef void *(*memset_t)(void *, int, size_t);

static volatile memset_t memset_func = memset;

void memory_clean(void *ptr, size_t len)
{
    memset_func(ptr, 0, len);
}

#define rol64(n, c) (((n) << (c)) | ((n) >> (64 - (c))))

#define bash_s(w0, w1, w2, m1, n1, m2, n2) \
do { \
    register uint64_t t0, t1, t2; \
    t0 = rol64(w0, m1); \
    w0 ^= w1 ^ w2; \
    t1 = w1 ^ rol64(w0, n1); \
    w1 = t0 ^ t1; \
    w2 ^= rol64(w2, m2) ^ rol64(t1, n2); \
    t1 = w0 | w2; \
    t2 = w0 & w1; \
    t0 = ~w2; \
    t0 |= w1; \
    w1 ^= t1; \
    w2 ^= t2; \
    w0 ^= t0; \
} while (0)

static const uint64_t __c1 = 0x3bf5080ac8ba94b1;
static const uint64_t __c2 = 0xc1d1659c1bbd92f6;
static const uint64_t __c3 = 0x60e8b2ce0ddec97b;
static const uint64_t __c4 = 0xec5fb8fe790fbc13;
static const uint64_t __c5 = 0xaa043de6436706a7;
static const uint64_t __c6 = 0x8929ff6a5e535bfd;
static const uint64_t __c7 = 0x98bf1e2c50c97550;
static const uint64_t __c8 = 0x4c5f8f162864baa8;
static const uint64_t __c9 = 0x262fc78b14325d54;
static const uint64_t __c10 = 0x1317e3c58a192eaa;
static const uint64_t __c11 = 0x098bf1e2c50c9755;
static const uint64_t __c12 = 0xd8ee19681d669304;
static const uint64_t __c13 = 0x6c770cb40eb34982;
```

```
47 static const uint64_t __c14 = 0x363b865a0759a4c1;
48 static const uint64_t __c15 = 0xc73622b47c4c0ace;
49 static const uint64_t __c16 = 0x639b115a3e260567;
50 static const uint64_t __c17 = 0xede6693460f3da1d;
51 static const uint64_t __c18 = 0xaad8d5034f9935a0;
52 static const uint64_t __c19 = 0x556c6a81a7cc9ad0;
53 static const uint64_t __c20 = 0x2ab63540d3e64d68;
54 static const uint64_t __c21 = 0x155b1aa069f326b4;
55 static const uint64_t __c22 = 0x0aad8d5034f9935a;
56 static const uint64_t __c23 = 0x0556c6a81a7cc9ad;
57 static const uint64_t __c24 = 0xde8082cd72debc78;
58
59 #define P0(x) x
60 #define P1(x) (((x) < 8) ? 8 + (((x) + 2 * ((x) & 1) + 7) % 8) : \
61 (((x) < 16) ? 8 + ((x) ^ 1) : (5 * (x) + 6) % 8))
62 #define P2(x) P1(P1(x))
63 #define P3(x) (8 * ((x) / 8) + (((x) % 8) + 4) % 8)
64 #define P4(x) P1(P3(x))
65 #define P5(x) P2(P3(x))
66
67 #define bash_r(s, p, p_next, i) \
68 do { \
69     bash_s(s[p( 0)], s[p( 8)], s[p(16)], 8, 53, 14, 1); \
70     bash_s(s[p( 1)], s[p( 9)], s[p(17)], 56, 51, 34, 7); \
71     bash_s(s[p( 2)], s[p(10)], s[p(18)], 8, 37, 46, 49); \
72     bash_s(s[p( 3)], s[p(11)], s[p(19)], 56, 3, 2, 23); \
73     bash_s(s[p( 4)], s[p(12)], s[p(20)], 8, 21, 14, 33); \
74     bash_s(s[p( 5)], s[p(13)], s[p(21)], 56, 19, 34, 39); \
75     bash_s(s[p( 6)], s[p(14)], s[p(22)], 8, 5, 46, 17); \
76     bash_s(s[p( 7)], s[p(15)], s[p(23)], 56, 35, 2, 55); \
77     s[p_next(23)] ^= __c##i; \
78 } while (0)
79
80 JNIEXPORT jbyteArray JNICALL Java_by_bsu_biot_library_LibraryNative_bash_1f(
       JNIEnv *env, jclass thisClass, jbyteArray inJNIArray) {
81     jbyte *inCArray = (*env)->GetByteArrayElements(env, inJNIArray, NULL);
82     if (NULL == inCArray) return NULL;
83     uint8_t arr[192];
84     for (int i = 0; i < 192; i++)
85     arr[i] = (uint8_t)inCArray[i];
86
87     uint64_t s[24];
88     memcpy(s, arr, 192);
89     bash_r(s, P0, P1, 1);
90     bash_r(s, P1, P2, 2);
91     bash_r(s, P2, P3, 3);
92     bash_r(s, P3, P4, 4);
93     bash_r(s, P4, P5, 5);
94     bash_r(s, P5, P0, 6);
95     bash_r(s, P0, P1, 7);
96     bash_r(s, P1, P2, 8);
97     bash_r(s, P2, P3, 9);
```

```
    bash_r(s, P3, P4, 10);
    bash_r(s, P4, P5, 11);
    bash_r(s, P5, P0, 12);
    bash_r(s, P0, P1, 13);
    bash_r(s, P1, P2, 14);
    bash_r(s, P2, P3, 15);
    bash_r(s, P3, P4, 16);
    bash_r(s, P4, P5, 17);
    bash_r(s, P5, P0, 18);
    bash_r(s, P0, P1, 19);
    bash_r(s, P1, P2, 20);
    bash_r(s, P2, P3, 21);
    bash_r(s, P3, P4, 22);
    bash_r(s, P4, P5, 23);
    bash_r(s, P5, P0, 24);
    memcpy(arr, s, 192);
    memory_clean(s, 192);

    jbyte outCArray[192];
    for (int i = 0; i < 192; i++)
    outCArray[i] = (jbyte)arr[i];

    (*env)->ReleaseByteArrayElements(env, inJNIArray, inCArray, 0);

    jbyteArray outJNIArray = (*env)->NewByteArray(env, 192);
    if (NULL == outJNIArray) return NULL;
    (*env)->SetByteArrayRegion(env, outJNIArray, 0, 192, outCArray);
    return outJNIArray;
}
```

ПРИЛОЖЕНИЕ Б

Исходный код реализации аутентифицированного шифрования из стандарта СТБ 34.101.77 на языке программирования Java

```
package by.bsu.biot.service;

import by.bsu.biot.dto.MachineDataType;
import by.bsu.biot.dto.EncryptionResult;
import by.bsu.biot.library.LibraryNative;
import org.apache.commons.lang3.ArrayUtils;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

@Service
public class EncryptionAndHashService {

    private static final int N = 1536;
    private static final int N_BYTES = 192;

    // уровеньстойкости
    private int l;

    // ёмкость
    private int d;

    // длинабуфера
    private int r;

    // текущеесмещениев буфере
    private int pos;

    // состояниеавтомата
    private byte[] S;

    private byte[] I;

    public void init(int l, int d) {
        this.l = l;
        this.d = d;
        S = new byte[N_BYTES];
    }

    private void start(byte[] A, byte[] K) {
        // 1.
        if (K.length != 0) {
            r = N - l - d * l / 2;
        }
```

```
        // 2.
        else {
            r = N - 2 * d * l;
        }
        // 3.
        pos = 8 * (1 + A.length + K.length);
        // 4.
        S[0] = (byte) ((8 * A.length / 2 + 8 * K.length / 32) % (Math.pow(2, 8)
            ));
        System.arraycopy(A, 0, S, 1, A.length);
        System.arraycopy(K, 0, S, 1 + A.length, K.length);
        // 5.
        for (int i = pos / 8; i < 1472 / 8; ++i) {
            S[i] = 0;
        }
        // 6.
        S[1472 / 8] = (byte) ((l / 4 + d) % (Math.pow(2, 64)));
        for (int i = 1472 / 8 + 1; i < N_BYTES; ++i) {
            S[i] = 0;
        }
    }

    private void commit(MachineDataType type) {
        // 1.
        S[pos / 8] = (byte) (S[pos / 8] ^ Byte.parseByte(type.code, 2));
        // 2.
        S[r / 8] = (byte) (S[r / 8] ^ 128); // 1000 0000
        // 3.
        S = LibraryNative.bash_f(S);
        // 4.
        pos = 0;
    }

    private void absorb(byte[] X) {
        // 1.
        commit(MachineDataType.DATA);
        // 2.
        byte[][] XSplit = split(X, r);
        // 3.
        for (byte[] Xi : XSplit) {
            // 3.1
            pos = Xi.length * 8;
            // 3.2
            for (int i = 0; i < pos / 8; ++i) {
                S[i] = (byte) (S[i] ^ Xi[i]);
            }
            // 3.3
            if (pos == r) {
                S = LibraryNative.bash_f(S);
                pos = 0;
            }
        }
```

```
    }

    private byte[] squeeze(int n) {
        // 1.
        commit(MachineDataType.OUT);
        // 2.
        byte[] Y = new byte[0];
        // 3.
        while (8 * Y.length + r <= n) {
            // 3.1
            Y = ArrayUtils.addAll(Y, ArrayUtils.subarray(S, 0, r / 8));
            // 3.2
            S = LibraryNative.bash_f(S);
        }
        // 4.
        pos = n - 8 * Y.length;
        // 5.
        Y = ArrayUtils.addAll(Y, ArrayUtils.subarray(S, 0, pos / 8));
        // 6.
        return Y;
    }

    private byte[] encrypt(byte[] X) {
        // 1.
        commit(MachineDataType.TEXT);
        // 2.
        byte[][] XSplit = split(X, r);
        // 3.
        byte[] Y = new byte[0];
        // 4.
        for (byte[] Xi : XSplit) {
            // 4.1
            pos = Xi.length * 8;
            // 4.2
            for (int i = 0; i < pos / 8; ++i) {
                S[i] = (byte) (S[i] ^ Xi[i]);
            }
            // 4.3
            Y = ArrayUtils.addAll(Y, ArrayUtils.subarray(S, 0, pos / 8));
            // 4.4
            if (pos == r) {
                S = LibraryNative.bash_f(S);
                pos = 0;
            }
        }
        // 5.
        return Y;
    }

    private byte[] decrypt(byte[] Y) {
        // 1.
        commit(MachineDataType.TEXT);
```

```
150         // 2.
151         byte[][] YSplit = split(Y, r);
152         // 3.
153         byte[] X = new byte[0];
154         // 4.
155         for (byte[] Yi : YSplit) {
156             // 4.1
157             pos = Yi.length * 8;
158             // 4.2
159             byte[] tempS = ArrayUtils.subarray(S, 0, pos / 8);
160             for (int i = 0; i < pos / 8; ++i) {
161                 tempS[i] = (byte) (tempS[i] ^ Yi[i]);
162             }
163             X = ArrayUtils.addAll(X, tempS);
164             // 4.3
165             System.arraycopy(Yi, 0, S, 0, pos / 8);
166             // 4.4
167             if (pos == r) {
168                 S = LibraryNative.bash_f(S);
169                 pos = 0;
170             }
171         }
172         // 5.
173         return X;
174     }
175
176     /**
177     * @param A - анонс
178     * @param K - ключ
179     * @param I - ассоциированныеданные
180     * @param X - сообщение
181     * @return зашифрованноесообщениеY иимитовставкаT
182     */
183     public EncryptionResult authEncrypt(byte[] A, byte[] K, byte[] I, byte[] X
        ) {
184         // 1.
185         start(A, K);
186         // 2.1
187         absorb(I);
188         // 2.2
189         byte[] Y = encrypt(X);
190         // 2.3
191         byte[] T = squeeze(l);
192         // 2.4
193         return EncryptionResult.builder()
194         .Y(Y)
195         .T(T)
196         .build();
197     }
198
199     /**
200      * @param A - анонс
```

```
201         * @param K - ключ
202         * @param I - ассоциированныеданные
203         * @param Y - зашифрованноесообщение
204         * @param T - имитовставка
205         * @return расшифрованноесообщениеX
206         */
207        public byte[] authDecrypt(byte[] A, byte[] K, byte[] I, byte[] Y, byte[] T
               ) {
208            // 1.
209            start(A, K);
210            // 2.1
211            absorb(I);
212            // 2.2
213            byte[] X = decrypt(Y);
214            // 2.3
215            if (!Arrays.equals(T, squeeze(l))) {
216                return null;
217            }
218            return X;
219        }
220
221        private static byte[][] split(byte[] bytes, int r) {
222            if (bytes.length * 8 <= r) {
223                return new byte[][]{bytes};
224            }
225            List<byte[]> result = new ArrayList<>();
226            int tempPos = 0;
227            while (tempPos * 8 + r < bytes.length * 8) {
228                result.add(ArrayUtils.subarray(bytes, tempPos, tempPos + r / 8));
229                tempPos += r / 8;
230            }
231            result.add(ArrayUtils.subarray(bytes, tempPos, bytes.length));
232
233            return result.toArray(new byte[0][]);
234        }
235    }
```

ПРИЛОЖЕНИЕ В

Исходный код реализации аутентифицированного шифрования из стандарта СТБ 34.101.77 на языке программирования C++

```
#include <iostream>
#include <cmath>
#include <string>
#include <sstream>
#include <iomanip>
#include <algorithm>
extern "C" {
    #include "library.h"
}
using namespace std;

size_t decode(uint8_t dest[], size_t count, const char *src) {
    char buf[3];
    size_t i;
    for (i = 0; i < count && *src; i++) {
        buf[0] = *src++;
        buf[1] = '\0';
        if (*src) {
            buf[1] = *src++;
            buf[2] = '\0';
        }
        if (sscanf(buf, "%hhx", &dest[i]) != 1)
        break;
    }
    return i;
}

void reverseAndDecode(uint8_t array[], string s) {
    string temp;
    for (int i = (int) (s.length() - 1); i >= 0; i -= 2) {
        temp += s[i-1];
        temp += s[i];
    }
    decode(array, temp.length(), temp.c_str());
    // reverse
    for (int i = 0; i < s.length() / 4; ++i) {
        uint8_t temp = array[i];
        array[i] = array[s.length() / 2 - 1 - i];
        array[s.length() / 2 - 1 - i] = temp;
    }
}

string encode(const uint8_t v[], const size_t s) {
    stringstream ss;

    ss << hex << setfill('0');
```

```
47
48      for (int i = 0; i < s; i++) {
49          ss << hex << setw(2) << static_cast<int>(v[i]);
50      }
51
52      return ss.str();
53  }
54
55  size_t l;
56  size_t d;
57  size_t r; // buf_len
58  size_t pos;
59  uint8_t S[192] = {0};
60
61  #define BASH_PRG_NULL    1 // 0x01, 000000 01 */
62  #define BASH_PRG_KEY     5 // 0x05, 000001 01 */
63  #define BASH_PRG_DATA    9 // 0x09, 000010 01 */
64  #define BASH_PRG_TEXT   13 // 0x0D, 000011 01 */
65  #define BASH_PRG_OUT    17 // 0x11, 000100 01 */
66
67  void start(uint8_t A[], size_t A_len, uint8_t K[], size_t K_len) {
68      // 1.
69      if (K_len != 0) {
70          r = 1536 - l - d * l / 2;
71      }
72      // 2.
73      else {
74          r = 1536 - 2 * d * l;
75      }
76      // 3.
77      pos = 8 * (1 + A_len + K_len);
78      // 4.
79      S[0] = (8 * A_len / 2 + 8 * K_len / 32) % (size_t)(pow(2, 8));
80      for (int i = 0; i < A_len; ++i) {
81          S[1 + i] = A[i];
82      }
83      for (int i = 0; i < K_len; ++i) {
84          S[1 + A_len + i] = K[i];
85      }
86      // 5.
87      for (int i = (int) pos / 8; i < 1472 / 8; ++i) {
88          S[i] = 0;
89      }
90      // 6.
91      S[1472 / 8] = (l / 4 + d) % (size_t)(pow(2, 8));
92      for (int i = 1472 / 8 + 1; i < 192; ++i) {
93          S[i] = 0;
94      }
95  }
96
97  void commit(uint8_t type) {
98      // 1.
```

```
    S[pos / 8] = S[pos / 8] ^ type;
    // 2.
    S[r / 8] = S[r / 8] ^ 128; // 1000 0000
    // 3.
    bash_f(S);
    // 4.
    pos = 0;
}

void absorb(uint8_t X[], size_t X_len) {
    // 1.
    commit(BASH_PRG_DATA);
    // 2-3.
    int tempPos = 0;
    while (8 * tempPos + r < 8 * X_len) {
        // 3.1
        pos = r;
        // 3.2
        for (int i = 0; i < pos / 8; ++i) {
            S[i] = S[i] ^ X[tempPos + i];
        }
        // 3.3
        bash_f(S);
        pos = 0;

        tempPos += r / 8;
    }
    // finish steps 2 & 3
    pos = 8 * (X_len - tempPos);
    for (int i = 0; i < pos / 8; ++i) {
        S[i] = S[i] ^ X[tempPos + i];
    }
}

void squeeze(uint8_t Y[], size_t n) {
    // 1.
    commit(BASH_PRG_OUT);
    // 2-3.
    int tempPos = 0;
    while (8 * tempPos + r <= n) {
        // 3.1
        for (int i = 0; i < r / 8; ++i) {
            Y[tempPos + i] = S[i];
        }
        tempPos += r / 8;
        // 3.2
        bash_f(S);
    }
    // 4.
    pos = n - 8 * tempPos;
    // 5.
    for (int i = 0; i < pos / 8; ++i) {
```

```
151         Y[tempPos + i] = S[i];
152     }
153 }
154 
155 void encrypt(uint8_t X[], size_t X_len, uint8_t Y[]) {
156     // 1.
157     commit(BASH_PRG_TEXT);
158     // 2-4.
159     int tempPos = 0;
160     while (8 * tempPos + r < 8 * X_len) {
161         // 4.1
162         pos = r;
163         // 4.2
164         for (int i = 0; i < pos / 8; ++i) {
165             S[i] = S[i] ^ X[tempPos + i];
166         }
167         // 4.3
168         for (int i = 0; i < pos / 8; ++i) {
169             Y[tempPos + i] = S[i];
170         }
171         // 4.4
172         bash_f(S);
173         pos = 0;
174 
175         tempPos += r / 8;
176     }
177     // finish steps 2 & 4
178     pos = 8 * (X_len - tempPos);
179     for (int i = 0; i < pos / 8; ++i) {
180         S[i] = S[i] ^ X[tempPos + i];
181     }
182     for (int i = 0; i < pos / 8; ++i) {
183         Y[tempPos + i] = S[i];
184     }
185 }
186 
187 void decrypt(uint8_t Y[], size_t Y_len, uint8_t X[]) {
188     // 1.
189     commit(BASH_PRG_TEXT);
190     // 2-4.
191     int tempPos = 0;
192     while (8 * tempPos + r < 8 * Y_len) {
193         // 4.1
194         pos = r;
195         // 4.2
196         for (int i = 0; i < pos / 8; ++i) {
197             X[tempPos + i] = S[i] ^ Y[tempPos + i];
198         }
199         // 4.3
200         for (int i = 0; i < pos / 8; ++i) {
201             S[i] = Y[tempPos + i];
202         }
```

```
        // 4.4
        bash_f(S);
        pos = 0;

        tempPos += r / 8;
    }
    // finish steps 2 & 4
    pos = 8 * (Y_len - tempPos);
    for (int i = 0; i < pos / 8; ++i) {
        X[tempPos + i] = S[i] ^ Y[tempPos + i];
    }
    for (int i = 0; i < pos / 8; ++i) {
        S[i] = Y[tempPos + i];
    }
}

void authEncrypt(size_t varL, size_t varD,
uint8_t A[], size_t A_len,
uint8_t K[], size_t K_len,
uint8_t I[], size_t I_len,
uint8_t X[], size_t X_len,
uint8_t Y[], uint8_t T[]) {
    l = varL;
    d = varD;
    // 1.
    start(A, A_len, K, K_len);
    // 2.1
    absorb(I, I_len);
    // 2.2
    encrypt(X, X_len, Y);
    // 2.3
    squeeze(T, l);
}

void authDecrypt(size_t varL, size_t varD,
uint8_t A[], size_t A_len,
uint8_t K[], size_t K_len,
uint8_t I[], size_t I_len,
uint8_t Y[], size_t Y_len,
uint8_t X[], uint8_t T[],
bool error) {
    l = varL;
    d = varD;
    // 1.
    start(A, A_len, K, K_len);
    // 2.1
    absorb(I, I_len);
    // 2.2
    decrypt(Y, Y_len, X);
    // 2.3
    uint8_t tempT[l];
    squeeze(tempT, l);
```

```
255     for (int i = 0; i < 32; ++i) {
256         if (tempT[i] != T[i]) {
257             error = true;
258             break;
259         }
260     }
261 }
```

ПРИЛОЖЕНИЕ Г

Исходный код реализации сервиса по отправке команд на умное устройство на языке программирования Java

```java
package by.bsu.biot.service;

import by.bsu.biot.dto.EncryptionResult;
import by.bsu.biot.util.HexEncoder;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import okhttp3.FormBody;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.Response;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;

import javax.annotation.PostConstruct;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.util.Base64;
import java.util.concurrent.TimeUnit;

@Service
@RequiredArgsConstructor
@Slf4j
public class ClientService {

    private final EncryptionAndHashService encryptionService;

    @Value("${biot.encryption.enabled}")
    private boolean encryptionEnabled;

    private String encryptionKey;

    private int messageCount;

    private static final String LUMP_STATIC_IP_ADDRESS = "192.168.100.93";

    private static final String LED_PATH = "/led";

    private static final String KEY_PATH = "/key";

    private static final int HTTP_SUCCESS_CODE = 200;

    private static final OkHttpClient client = new OkHttpClient().newBuilder()
    .connectTimeout(10, TimeUnit.SECONDS)
    .readTimeout(30, TimeUnit.SECONDS)
    .build();

```

```
47      @PostConstruct
48      public void init() {
49          int l = 256;
50          int d = 1;
51          byte[] I = new byte[0];
52          encryptionService.init(l, d, I);
53          messageCount = 0;
54
55          encryptionKey = HexEncoder.generateRandomHexString(System.getenv("
                INITIAL_ENCRYPTION_KEY").length());
56          log.info("encryption key: " + encryptionKey);
57      }
58
59      public void turnOn() throws IOException {
60          log.info("onn command sent");
61          if (encryptionEnabled) {
62              sendEncryptedMessage("onn", encryptionKey, LED_PATH);
63          } else {
64              sendPostRequest("onn", "", LED_PATH);
65          }
66      }
67
68      public void turnOff() throws IOException {
69          log.info("off command sent");
70          if (encryptionEnabled) {
71              sendEncryptedMessage("off", encryptionKey, LED_PATH);
72          } else {
73              sendPostRequest("off", "", LED_PATH);
74          }
75      }
76
77      public void sendEncryptionKey() throws IOException {
78          sendEncryptedMessage(encryptionKey, System.getenv("
                INITIAL_ENCRYPTION_KEY"), KEY_PATH);
79      }
80
81      private void sendEncryptedMessage(String message, String key, String path)
             throws IOException {
82          byte[] A = String.valueOf(++messageCount).getBytes();
83          byte[] K = HexEncoder.decode(key);
84          byte[] X = message.getBytes(StandardCharsets.UTF_8);
85
86          EncryptionResult encryptionResult = encryptionService.authEncrypt(A, K,
                 X);
87
88          log.info("encrypted message: " + HexEncoder.encode(encryptionResult.
                getY()));
89          log.info("mac: " + HexEncoder.encode(encryptionResult.getT()));
90          String encryptedMessage = new String(
91          Base64.getEncoder().encode(HexEncoder.encode(encryptionResult.getY()).
                getBytes()));
92          String mac = new String(
```

```
93          Base64.getEncoder().encode(HexEncoder.encode(encryptionResult.getT()).
                getBytes()));
94
95          Response response = sendPostRequest(encryptedMessage, mac, path);
96          if (response.code() == HTTP_SUCCESS_CODE) {
97              log.info("message has been successfully processed");
98              log.info("count: " + ++messageCount);
99          }
100     }
101
102     private Response sendPostRequest(String param1, String param2, String path
            ) throws IOException {
103         FormBody body = new FormBody.Builder()
104         .add("param1", param1)
105         .add("param2", param2)
106         .build();
107         Request request = new Request.Builder()
108         .url("http://" + LUMP_STATIC_IP_ADDRESS + path)
109         .post(body)
110         .build();
111
112         return client.newCall(request).execute();
113     }
114 }
```

ПРИЛОЖЕНИЕ Д

Исходный код реализации прошивки для микроконтроллера ESP8266 на языке программирования C++

```
#include <ESP8266WiFi.h>
#include <DNSServer.h>
#include <ESP8266WebServer.h>
#include <WiFiManager.h>
#include "standard77.hpp"
#include "base64.hpp"
#include "secrets.h"

const char* PARAM_INPUT_1 = "param1";
const char* PARAM_INPUT_2 = "param2";
const char* ERROR_MESSAGE = "error";
const char* ENCODED_ON_COMMAND = "6f6e6e";
const char* ENCODED_OFF_COMMAND = "6f6666";

ESP8266WebServer server(80);

int LED = D1;

string ENCRYPTION_KEY;

int messageCount;

void healthCheck() {
    server.send(200, "text/plain", "Ok");
}

string decrypt(string key) {
    if (server.hasArg(PARAM_INPUT_1) && server.hasArg(PARAM_INPUT_2)) {
        string encodedMessage = server.arg(PARAM_INPUT_1).c_str();
        string encodedMac = server.arg(PARAM_INPUT_2).c_str();

        uint8_t hexMessageArray[BASE64::decodeLength(encodedMessage.c_str())];
        BASE64::decode(encodedMessage.c_str(), hexMessageArray);
        string hexMessage = reinterpret_cast<char *>(hexMessageArray);

        Serial.println("encrypted message:");
        Serial.println(hexMessage.c_str());

        uint8_t hexMacArray[32];
        BASE64::decode(encodedMac.c_str(), hexMacArray);
        string hexMac = reinterpret_cast<char *>(hexMacArray);

        Serial.println("mac:");
        Serial.println(hexMac.c_str());

        size_t l = 256;
```

```
47          size_t d = 1;
48
49          string countString = std::to_string(++messageCount);
50          uint8_t A[countString.length()];
51          Serial.println("count:");
52          Serial.println(messageCount);
53          char aChar[countString.length()];
54          strcpy(aChar, countString.c_str());
55          for (int i = 0; i < countString.length(); ++i) {
56              A[i] = (uint8_t)aChar[i];
57          }
58          uint8_t K[32];
59          reverseAndDecode(K, key);
60          uint8_t I[0];
61
62          size_t Y_len = hexMessage.length() / 2;
63          uint8_t Y[Y_len];
64          decode(Y, Y_len, hexMessage.c_str());
65
66          uint8_t X[Y_len];
67
68          size_t T_len = 16;
69          uint8_t T[T_len];
70          decode(T, T_len, hexMac.c_str());
71
72          bool error = false;
73          authDecrypt(l, d, A, countString.length(), K, 32, I, 0, Y, Y_len, X, T,
                error);
74          if (!error) {
75              return encode(X, Y_len);
76          }
77      }
78      return ERROR_MESSAGE;
79  }
80
81  void handleKey() {
82      ENCRYPTION_KEY.clear();
83      ENCRYPTION_KEY = decrypt(INITIAL_ENCRYPTION_KEY);
84      if (ENCRYPTION_KEY != ERROR_MESSAGE) {
85          Serial.println("encryption key:");
86          Serial.println(ENCRYPTION_KEY.c_str());
87          ++messageCount;
88          server.send(200);
89      }
90  }
91
92  void handleBody() {
93      if (ENCRYPTIN_ENABLED) {
94          string state = decrypt(ENCRYPTION_KEY);
95          if (state != ERROR_MESSAGE) {
96              if (state == ENCODED_ON_COMMAND) {
97                  Serial.println("encrypted command received: onn");
```

```
                digitalWrite(LED, HIGH);
                ++messageCount;
                server.send(200);
            }
            else if (state == ENCODED_OFF_COMMAND) {
                Serial.println("encrypted command received: off");
                digitalWrite(LED, LOW);
                ++messageCount;
                server.send(200);
            }
        }
    }
    else {
        if (server.hasArg(PARAM_INPUT_1)) {
            String state = server.arg(PARAM_INPUT_1);
            if (state == "onn") {
                Serial.println("command received: onn");
                digitalWrite(LED, HIGH);
                ++messageCount;
                server.send(200);
            }
            else if (state == "off") {
                Serial.println("command received: off");
                digitalWrite(LED, LOW);
                ++messageCount;
                server.send(200);
            }
        }
    }
}

void setup() {
    Serial.begin(115200);
    WiFiManager wifiManager;
    // wifiManager.resetSettings();

    // wifiManager.startConfigPortal(LIGHT_SSID, LIGHT_PASSWORD);
    wifiManager.autoConnect(LIGHT_SSID, LIGHT_PASSWORD);

    pinMode(LED, OUTPUT);
    digitalWrite(LED, HIGH);
    messageCount = 0;

    server.on("/health", HTTP_GET, healthCheck); // health check request
    server.on("/key", HTTP_POST, handleKey); // encryption key
    server.on("/led", HTTP_POST, handleBody); // led commands
    server.begin();
}

void loop() {
    server.handleClient();
}
```