Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий

Работа допущена к защите
Руководитель ОП
_____ А.В. Щукин
«_____» _____ 2020 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
РАБОТА БАКАЛАВРА
СРАВНИТЕЛЬНЫЙ АНАЛИЗ АЛГОРИТМОВ ЛЕГКОВЕСНОЙ КРИПТОГРАФИИ ДЛЯ УСТРОЙСТВ ИНТЕРНЕТА ВЕЩЕЙ

по направлению подготовки 09.03.03 Прикладная информатика
Направленность (профиль) 09.03.03_03 Прикладная информатика в области информационных ресурсов

Выполнил
студент гр. 3530903/60301 Д.М. Момот

Руководитель
доцент ВШИСиСТ,
к. т. н. А.В. Сергеев

Консультант
по нормоконтролю В.А. Пархоменко

Санкт-Петербург
2020

САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО

Институт компьютерных наук и технологий

УТВЕРЖДАЮ
Руководитель ОП
_____ А.В. Щукин
«_____» _____ 2020г.

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

студенту Момоту Даниэлю Михайловичу гр. 3530903/60301

1. Тема работы: Сравнительный анализ алгоритмов легковесной криптографии для устройств интернета вещей.
2. Срок сдачи студентом законченной работы: 20.05.2020.
3. Исходные данные по работе: спецификация языка C++11 [**task_src1**], библиотеки языка C++ [**task_src2**].
4. Содержание работы (перечень подлежащих разработке вопросов):
   4.1. Обзор теоретических источников по темам легковесной криптографии и интернета вещей.
   4.2. Обзор существующих стандартов и технических решений.
   4.3. Анализ различных видов алгоритмов.
   4.4. Реализация избранных алгоритмов.
   4.5. Ранжирование видов алгоритмов по степени пригодности к использованию в устройствах IoT. Формулировка рекомендаций по их использованию.
5. Перечень графического материала (с указанием обязательных чертежей):
   5.1. Графики, отражающие результаты эксперимента.
6. Консультанты по работе:
   6.1. Ассистент ВШИСиСТ, В.А. Пархоменко (нормоконтроль).
7. Дата выдачи задания: 03.02.2020.

Руководитель ВКР _____ А.В. Сергеев

Задание принял к исполнению 03.02.2020

Студент _____ Д.М. Момот

РЕФЕРАТ

На ?? с., ?? рисунков, ?? таблиц, ?? приложений.

КЛЮЧЕВЫЕ СЛОВА: ЛЕГКОВЕСНАЯ КРИПТОГРАФИЯ, КРИПТОГРАФИЧЕСКИЕ АЛГОРИТМЫ, ИНТЕРНЕТ ВЕЩЕЙ, C, СРАВНИТЕЛЬНЫЙ АНАЛИЗ, СРАВНИТЕЛЬНОЕ ТЕСТИРОВАНИЕ.

Тема выпускной квалификационной работы: «Сравнительный анализ алгоритмов легковесной криптографии для устройств интернета вещей».

В данной работе рассматривается подход к сравнительному анализу легковесных криптографических алгоритмов применительно к устройствам интернета вещей. Выделяются возможные угрозы их безопасности, анализируются различные виды легковесных криптографических алгоритмов для защиты от них. Предлагается методология сравнительного тестирования производительности и мощности энергопотребления таких алгоритмов. Методология прошла апробацию в части тестирования производительности путем практической реализации тестирования трех различных реализаций алгоритма AES.

ABSTRACT

?? p., ?? figures, ?? tables, ?? appendices.

KEYWORDS: LIGHTWEIGHT CRYPTOGRAPHY, CRYPTOGRAPHIC ALGORITHMS, IOT, C, COMPARATIVE ANALYSIS, BENCHMARKING.

The subject of the graduate qualification work is «Comparative analysis of lightweight cryptography algorithms for IoT devices».

This paper considers an approach to the comparative analysis of lightweight cryptographic algorithms applied to the Internet of things devices. Possible threats to their security are identified, different types of lightweight cryptographic algorithms are analyzed to defend against threats. A methodology for performance and power consumption benchmarking of such algorithms is proposed. The methodology of performance benchmarking was verified through the practical implementation of testing three different implementations of the AES algorithm.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В последние десятилетия сетевые технологии прочно вошли в жизнь человека. Изначально они были представлены локальными сетями, затем была создана сеть Интернет. В настоящее время одним из бурно развивающихся направлений сетевых технологий является интернет вещей (Internet of Things, IoT). Так, за период 2015-2018 гг. доля IoT-устройств среди всех устройств увеличилась с 27% до 39%, и, согласно прогнозу, достигнет 63% к 2025 году [**src1**].

Интернет вещей – это вычислительная сеть физических предметов (устройств, «вещей»), оснащенных встроенной технологией для взаимодействия друг с другом или с внешней средой [**src2**].

Актуальность исследования. Одной из важных задач при проектировании IoT является обеспечение должного уровня безопасности передаваемых данных. Особенно это важно для медицинских устройств [**src3**]. Уже сейчас правительства развитых стран начинают принимать законы, регламентирующие защиту IoT–устройств [**src4**; **src5**; **src6**]. В то же время фактическая безопасность устройств интернета вещей оставляет желать лучшего. Так, согласно исследованию корпорации HP 2014-го года, 70% устройств IoT передавали данные, в том числе конфиденциального характера, вообще без шифрования [**src7**]! По этой причине изучение и развитие средств защиты IoT-устройств является актуальной задачей.

Безопасность устройств с ограниченными энергетическими ресурсами (а именно такими являются устройства интернета вещей) изучает раздел криптографии, называемый легковесной криптографией (lightweight cryptography, LWC). Также возможны термины «облегченная криптография», «малоресурсная криптография», «низкоэнергетическая криптография». Она рассматривает криптографические алгоритмы в контексте их требовательности к ресурсам устройства и количеству логических элементов, требуемых для реализации алгоритмов. Рассматриваемые ею алгоритмы называются алгоритмами легковесной криптографии (легковесными алгоритмами, LW-алгоритмами). LW-алгоритмы могут иметь программную или аппаратную реализацию.

Объектом исследования являются алгоритмы легковесной криптографии, ориентированные на использование в устройствах интернета вещей.

Предметом исследования являются следующие характеристики легковесных алгоритмов: тип алгоритма, требования к устройству, производительность. Алгоритмы рассматриваются с точки зрения программной реализации.

Целью исследования является сравнение легковесных алгоритмов и определение степени их пригодности к использованию в устройствах IoT. Для достижения выбранной цели поставлены следующие **задачи**:

A. Обзор теоретических источников по темам легковесной криптографии и интернета вещей.
B. Обзор существующих стандартов и технических решений.
C. Анализ различных видов алгоритмов.
D. Реализация избранных алгоритмов.
E. Ранжирование видов алгоритмов по степени пригодности к использованию в устройствах IoT. Формулировка рекомендаций по их использованию.

Гипотеза исследования. Предполагается, что значительное число классических шифров с определенным ослаблением могут использоваться в качестве легковесных алгоритмов. С другой стороны, многие алгоритмы и классы алгоритмов по тем или иным соображениям однозначно не могут быть использованы в качестве LW-алгоритмов. Ряд алгоритмов может быть использован наилучшим образом при определенных условиях. Может быть создана методика, позволяющая приблизительно протестировать производительность и можность энергопотребления реализации легковесного алгоритма с использованием только персонального компьютера, без необходимости взаимодействия с низкоресурсными устройствами.

В данной работе используются такие **методы исследования**, как:

– анализ технической литературы;
– изучение существующих спецификаций алгоритмов, стандартов, технологических систем;
– декомпозиция алгоритмов;
– сравнение алгоритмов, подходов к шифрованию, структурных частей алгоритмов;
– реализация алгоритмов на языке C;
– определение методологии их тестирования;
– реализация конкретной методики тестирования;
– анализ результатов тестирования;
– синтез выводов по результатам тестирования;
– обобщение результатов работы.

Данное исследование имеет высокую **практическую значимость**. Результаты исследования могут быть использованы при выборе алгоритма шифрования и

режима его работы при проектировании системы защиты вычислительной сети IoT. Кроме того, работа в данном направлении может быть продолжена: следующим шагом возможна оптимизация существующих легковесных алгоритмов или создание новых LW-алгоритмов. Предложенная методология тестирования времени работы алгоритмов и их энергопотребления может быть использована при тестировании других алгоритмов. Присутствует и **научная значимость**. Можно исследовать «побочные» вопросы, возникшие в процессе выполнения работы.

Введение раскрывает актуальность, определяет степень научной разработки темы, объект, предмет, цель, задачи и методы исследования, раскрывает теоретическую и практическую значимость работы.

В первой главе содержатся краткое теоретическое введение в тему криптографии и интернета вещей, описываются основные угрозы безопасности систем IoT.

Во второй главе вводится понятие легковесных криптографических алгоритмов. Описываются требования к ним и их применение для нейтрализации угроз безопасности системам IoT, оно сравнивается с нейтрализацией угроз в системах общего назначения). Проводится анализ различных видов алгоритмов на пригодность их для такого применения, рассматриваются наиболее популярные их представители.

В третьей главе описывается методология тестирования производительности и энергопотребления легковесных криптоалгоритмов на персональном компьютере.

В четвертой главе содержится информация о реализации алгоритмов в программном коде, способе и порядке тестирования производительности алгоритмов, а также дальнейшей обработки результатов тестирования. Проводится анализ полученных результатов.

В заключении подводятся итоги работы, приводятся краткие перспективы использования результатов работы.

ГЛАВА 1. ОБЗОР ТЕОРЕТИЧЕСКОЙ ЛИТЕРАТУРЫ ПО КРИПТОГРАФИИ И ИНТЕРНЕТУ ВЕЩЕЙ

В данной главе приводится основная терминологеская и теоретическая база, на которой будут строиться дальнейшие главы. Рассматриваются основные понятия криптографии и интернета вещей. Описывается уровневая модель систем интернета вещей, выделяется каждый уровень и атаки на него.

1.1. Криптографические алгоритмы: основные понятия

1.1.1. Информация и криптография

Человеческая цивилизация в течение всего времени существования работает с различной информацией. Информация может представлять собой как исторические сведения или описания технологических процессов, так и частные данные человека или группы лиц. Сведения, составляющие информацию и представляющие определенную ценность, должны быть защищены от лиц, не имеющих соответствующих прав доступа. Поэтому задача защиты информации всегда была актуальной.

Защитить информацию можно тремя способами [**src8**].

A. Создание абсолютно надежного и изолированного от доступа извне хранилища информации и инфраструктуры для него. Это является крайне трудоемкой и дорогостоящей задачей.
B. Скрытие факта существования или передачи информации. Средства и методы такого скрытия изучает стеганография.
C. Хранение и передача информации, преобразованной таким образом, чтобы обратное преобразование могли совершить только определенные лица. Средства и методы такого преобразования информации изучает криптография.

Формализуем вышесказанное.

Информация – сведения, передаваемые одними людьми другим людям устным, письменным или каким-либо другим способом, а также сам процесс передачи или получения этих сведений [**src9**].

Защита информации – деятельность, направленная на предотвращение утечки защищаемой информации, несанкционированных и непреднамеренных

воздействий на защищаемую информацию [**src10**]. Утечка – неконтролируемое распространение защищаемой информации в результате ее разглашения, несанкционированного доступа к информации и получения защищаемой информации иностранными разведками [**src11**].

Криптография -— инженерно-техническая дисциплина, изучающая математические методы защиты информации (шифры). Криптография включает в себя криптосинтез и криптоанализ. Криптосинтез изучает подходы к разработке шифров. Криптоанализ изучает подходы к вскрытию шифров [**src8**].

1.1.2. Шифры: основные понятия и принципы

Итак, шифр (криптографический алгоритм, шифрующая функция) – преобразование данных, обеспечивающее их защищённость (зашифрованность). Это преобразование вида $f : x \rightarrow y$, где x – исходные, незашифрованные данные (представленные в виде целого числа), y – полученный зашифрованный текст (также целое число), f – шифрующая функция.

Дешифрующая функция – функция, обратная к шифрующей: $f^{-1} : y \rightarrow x$. Результатом применения дешифрующей функции к зашифрованным данным являются исходные данные.

Для того, чтобы шифрование было эффективным, необходимо, чтобы шифрующая функция f была легко вычислимой, для затруднения дешифровки дешифрующая функция f^{-1} должна быть достаточно сложна для вычисления [**src12**]. Функции такого вида называются односторонними.

Функция f называется *односторонней*, если выполняются условия [**src12**]:

A. для любого x из некоторого множества существует эффективный алгоритм вычисления $y = f(x)$;
B. не существует эффективного алгоритма обращения функции f.

Однако шифр должен быть не только недоступным для взлома злоумышленником. Законный получатель должен иметь возможность эффективно получить исходные данные. Это требование формализуется как «функция с секретом» или «односторонняя функция с секретом».

Функция f называется *односторонней функцией с секретом*, если выполняются условия [**src12**]:

A. для любого x из некоторого множества существует эффективный алгоритм вычисления $y = f(x)$;

В. функция f обладает «секретным свойством» k, таким что:.

1. при использовании свойства k можно построить эффективный алгоритм построения обратной функции f^{-1};
2. если свойство k неизвестно, то не существует эффективного алгоритма обращения функции f.

Итак, при использовании шифра на основе функции с секретом, шифровать (вычислять функцию f) могут все, а эффективно расшифровывать – только лица, которым известно секретное свойство k.

Секретное свойство также называют *ключом шифра*. Он представляет собой число или набор чисел (параметров алгоритма).

Главная часть шифра, его сердце – некоторая *трудная* задана, которая используется для гарантии того, что узнать секретное свойство (т. е. обратить шифрующую функцию и получить доступ к зашифрованным данным) очень трудно. Пара примеров трудных задач:

- Задача факторизации – разложение большого числа на простые множители (используется, например, в алгоритме RSA). В этом случае секретное свойство – факторизуемое число.
- Задача дискретного логарифмирования – обращение функции вида f(x)=aˆx mod p, где p – большое простое число, a – параметр, подобранный для конкретного p (используется, например, в схеме Эль-Гамаля).

Однако если будет представлен алгоритм, эффективно решающий трудную задачу (т. е. задача перестанет быть трудной), шифры на ее основе мгновенно устареют. Поэтому поиск трудных задач (и, наоборот, поиски быстро решающих их алгоритмов) является одним из важных направлений криптографии.

С другой стороны, в секрете хранится только ключ алгоритма, сам алгоритм должен быть открытым. То есть вся криптографическая стойкость алгоритма должна содержаться в ключе, а злоумышленник может знать о криптографической схеме все, кроме ключа. Это правило называется *принципом Керкгоффса*. Его выполнение обеспечивает надежность системы: даже в случае подбора ключа (методом перебора или, например, шантажа) злоумышленником достаточно его сменить, и начинать взлом нужно с самого начала. Похожий принцип «враг знает систему» был (вероятно, независимо) сформулирован Шенноном и называется *максимой Шеннона* [**src13**].

Это, однако, является только рекомендацией. В правительственных, военных и ряде других областей применяется противоположный принцип: безопасность

через неясность. Его суть состоит в скрытии внутренней структуры системы безопасности. Для шифров общего назначения он не рекомендуется, их, напротив, публикуют и обсуждают, чтобы коллективно найти уязвимости и возможности улучшения [**src14**].

1.1.3. Симметричная криптография

Изначально существовали только симметричные схемы шифрования. Они предполагают наличие одного ключа, используемого как для шифрования, так и для расшифровки сообщений. Он хранится в тайне, поэтому такие схемы также называются схемами с закрытым ключом. Классический пример – алгоритм DES.

Симметричные шифры в основном состоят из двух главных компонентов: шифры подстановки (перестановка символов/битов сообщения) и шифры замены (замена символов/битов по отдельности на другие по некоторому правилу) [**src8**].

Симметричные шифры делятся на две категории: блочные и потоковые. Потоковые криптоалгоритмы обрабатывают данные побитово или побайтово. Блочные алгоритмы обрабатывают данные целыми группами битов (блоками), обычно размер блока кратен 64 и составляет от 64 до 256 бит.

Симметричные шифры обладает рядом недостатков:

- Ключ должен храниться в тайне обеими сторонами и передаваться только по *защищенному* каналу (либо в зашифрованном виде). Это требует дополнительных ресурсов.
- Шифровать могут только те, кто знает ключ. Это означает либо жесткие ограничения на количество шифрующих лиц, либо значительный риск компрометации ключа в случае ослабления этих ограничений (компрометация ключа – его раскрытие не криптографическим способом [**src15**]).

1.1.4. Асимметричная криптография

После появления понятия односторонней функции появился другой вид алгоритмов – асимметричные. Такой способ шифрования предполагает наличие известного всем участникам схемы шифрования *открытого ключа* и известного только законным участникам *закрытого* ключа. Открытый ключ используется для шифрования данных, а закрытый – для расшифровки. Таким образом, шифровать сообщения может кто угодно, а вот расшифровывать – только законные пользователи.

Такой способ работы снимает основные проблемы симметричного шифрования: хранить в секрете шифрующий ключ нет необходимости, также он может передаваться по открытым каналам в незашифрованном виде и транслироваться на любую аудиторию.

По причине наличия открытого ключа такие алгоритмы также называются алгоритмами с открытым ключом. Классический пример – алгоритм RSA.

В некоторых случаях, например при использовании цифровой подписи, сообщения шифруются закрытым ключом, а расшифровываются открытым.

1.1.5. Криптографические хэш-функции

Другим направлением криптографии, основанным на понятии односторонней функции, являются криптографические (односторонние, однонаправленные) хэш-функции.

Хэш-функция – функция, принимающая строку произвольной (или почти произвольной) длины, и преобразующую ее в строку фиксированной, обычно меньшей, длины. Полученная строка называется отпечатком (дайджестом) входной строки или ее хэш-кодом.

Однонаправленная хэш-функция вычисляется только в одном направлении: легко вычислить значение дайджеста по входной строке, но крайне трудно создать прообраз, дайджест которого соответствует заданной строке [**src15**].

Криптографическая хэш-функция является открытой. Безопасность обеспечивается именно однонаправленностью функции. Одним из необходимых условий является изменение значения половины битов дайджеста при изменении в одном бите входа: невозможно путем сравнения близких входов обратить хэширующую функцию.

Хэш-функции применяются при вычислении контрольных сумм для проверки подлинности файлов и транзакций: почти невозможно подобрать поддельный файл, отличный от настоящего, но с таким же дайджестом. При таком применении от функции требуется высокое быстродействие, т. к. файлы могут иметь значительный размер, а транзакции часто должны обрабатываться в реальном времени.

Еще одно использование контрольных сумм – при использовании цифровой подписи можно подписывать не весь документ целиком, а только его контрольную сумму. Это значительно ускоряет работу с подписью и уменьшает потребление памяти при хранении подписей [**src15**].

Также криптографические хэш-функции используются для хранения паролей. Такие хэш-функции называются Key Derivation Functions, KDF. В этом случае на носителе пароль не сохраняется, а хранится только его хэш, и каждый раз при вводе строки вычисляется ее хэш-код и сравнивается с хэшем пароля. Если произошло совпадение, значит, считаем, пароль введен верно. Для минимизации вероятности коллизии можно хранить два дайджеста от разных хэш-функций.

Такой способ хранения паролей является в настоящее время наиболее распространенным, так как сам по себе обеспечивает дополнительных уровень защищенности: даже если злоумышленник получил доступ к хэшам, для восстановления по ним паролей ему потребуется много времени (или большие вычислительные мощности). При этом от хэширующей пароли функции требуется, чтобы она вычислялась долго: время вычисления порядка 100 мс незаметно при авторизации, однако значительно затрудняет подбор пароля методом грубой силы. Кроме того, по тем же соображениям желательно большое потребление оперативной памяти, это также усложняет перебор, особенно в многопоточном режиме. К их стойкости также предъявляются повышенные требования.

Для дополнительного повышения защищенности хэшированных паролей используется *соль*. Это случайная строка, добавляемая к шифруемым данным, она должна храниться вместе с хэшем. Она не позволит понять, что захэшированы одинаковые строки, так как они будут иметь разную соль. Благодаря этому, брутфорс одного пароля из базы не позволяет найти другие такие же пароли в базе [**src15**].

Итак, есть два вида криптографических хэш-функций:

- Быстрые, используются для вычисления контрольных сумм и должны потреблять как можно меньше ресурсов. Примеры: MD5, семейство SHA2, из новых – SHA3 и BLAKE2.
- Медленные, используются для хэширования паролей. Должны потреблять много ресурсов и быть очень стойкими. Примеры: bcrypt и scrypt, новая – Argon2.

1.2. Интернет вещей

1.2.1. Основные понятия

Сэмюэл Грингард (Samuel Greengard) – журналист, специализируется на новых технологиях. Директор по маркетингу во многих технологических и бизнес–изданиях, бывший президент Американского общества журналистов. В области IoT известен главным образом книгой [**src19**]. Далее в этом разделе приводятся избранные положения этой работы.

Подключаемые устройства – устройства, которые обмениваются данными по обычному интернет-соединению и получают дополнительные преимущества при подключении, например, через закрытую или частную сеть. Подключаемые устройства необязательно подсоединяются именно к Интернету вещей, но это происходит все чаще.

Радиочастотная идентификация (RFID) – это основной инструмент, который позволяет устройствам стать подключаемыми. Эта технология автоматической идентификации основана на считывании или записи данных, хранящихся в RFID–метках. RFID-метки могут быть как активными (с собственным источником питания), так и пассивными (им не требуется источник питания). И те, и другие позволяют считывателям автоматически получать сигнал и данные с меток. При этом метка должна находиться не дальше определенного допустимого расстояния.

Пассивные радиочастотные метки особенно востребованы благодаря низкой стоимости, долговечности и отсутствию необходимости в постоянном электропитании, они получают питание от ближайшего считывателя. Они могут быть встроены в наклейки для удобства использования или имплантированы под кожу (например, продукт VeriChip [**src16**]). Стоимость определяется мощностью считывающего устройства. Чем больше его мощность, тем меньше требования к размеру и качеству метки, что, в свою очередь, определяет более низкую стоимость [**src17**]. Уже в 2004 году некоторые метки стоили всего 5 центов [**src18**].

Промышленный Интернет – оборудование и аппаратура, оборудованные датчиками. Датчики позволяют получать данные с самых различных устройств унифицированным образом. Затем данные со считывающих устройств могут быть собраны и централизованно обработаны.

Таким образом, возможность сделать большинство устройств подключаемыми позволяет централизованно (в смысле единого канала, Интернета) управлять

практически всей техникой. В результате возникает *Интернет всего* (этот термин введен компанией Cisco). Интернет всего позволяет значительно увеличить автоматизацию за счет упрощения взаимодействия отдельных устройств, модулей и систем. В конечном счете это повышает управляемость системой и, если необходимо, целым кластером систем.

В книге упоминается «парадокс автоматизации»: по мере развития автоматизированных систем вероятность аварии или сбоя снижается, однако степень тяжести потенциальной опасности во много раз повышается. Это создает дополнительные требования к персоналу и пользователям таких систем. Кроме того, возникает психологический эффект расслабления, когда человек целиком полагается на технику.

Автор отмечает дилемму, стоящую перед разработчиками. Создание функциональных интерфейсов и средств управления делают устройства (и, если посмотреть шире, системы) удобнее, но также делают их мишенью для атак. И если прямого доступа к управлению отдельными модулями нет (или нет соответствующих навыков у пользователя или обслуживающего персонала), неисправность или уязвимость не могут быть обнаружены до тех пор, пока проблема не проявится сама, причинив немалый ущерб. Следовательно, разработчики и производители должны находить новые способы обеспечения безопасности как системы в целом, так и отдельных модулей. При этом система защиты должна иметь максимально простой интерфейс управления или, еще лучше, вовсе не требовать управления.

1.2.2. Уровневая модель IoT и проблемы безопасности

Стоит отметить хорошую работу [**src20**], посвященную обзору элементов сети IoT, ее «слоистой» архитектуры, а также актуальных проблем безопасности. Этот раздел написан в основном по материалам данной работы.

Для идентификации устройств используются адреса IPv4 и IPv6, для именования может использоваться система ucode. Сбор информации осуществляется, например, с помощью RFID-меток, носимых устройства. Для сетевого взаимодействия – основного компонента IoT – используются такие технологии, как RFID, NFC, Bluetooth, Wi-Fi, LTE. Обработка информации включает в себя отбор информации и затем необходимые вычисления. Примеры аппаратных платформ: Arduino, Raspberry Pi, Intel Galileo. Среди ОС можно назвать TinyOS, LiteOS,

Android. Для обеспечения общей семантики используются унифицированные модели представления данных RDF, OWL, EXI.

Приложения IoT предоставляют следующие 4 типа услуг пользователю:

– распределенное распознавание объектов;
– сбор, обработка и распределенное хранение информации;
– распределенная решающая (управляющая) система;
– контроль и динамическая корректировка работы различных устройств.

Архитектура сети IoT состоит из нескольких уровней. Базовый трехуровневый вариант отражает главные черты архитектуры. Он состоит из следующих уровней:

– прикладной уровень (application layer);
– сетевой уровень (network layer, transmission layer);
– сенсорный уровень (perception layer, sensor layer).

Сенсорный уровень. На этом уровне осуществляется идентификация «вещей» и сбор поставляемых ими данных. Данные могут быть самыми разными, в зависимости от конкретной области применения: местоположение, температура, вибрация и другие. Этот уровень зачастую является основной целью злоумышленников. См, например, статью [**src21**]. Основные атаки:

– Подслушивание (eavesdropping) – перехват информации. Для таких атак уязвимы данные, передаваемые в открытом или слабо зашифрованном виде.
– Захват узла (node capture) – захват контроля над важным узлом передачи или временного хранения данных. Может привести к утечке сразу большого количества информации, в том числе ключей шифрования.
– Введение фальшивого узла (fake and malicious node) – добавление нового узла в систему. Эта атака направлена на прекращение передачи реальной информации и передачу вместо нее фальшивой. Кроме того, узел может пытаться получить контроль над другими узлами или заставить их повысить энергопотребление для причинения максимального урона сети.
– Атака повторного воспроизведения (атака повторением пакетов, replay attack, playback attack) – злоумышленник собирает информацию, передаваемую от отправителя к получателю, и фиксирует реакцию получателя. После чего позднее отправляет такую же информацию с целью заставить получателя выполнить желаемые действия. Эта атака весьма

- Атака по времени (timing attack) – злоумышленники фиксирует время реакции системы на различные запросы, чтобы получить сведения о ее устройстве и обнаружить потенциальные уязвимости. Может быть также направлена на одно устройство, в этом случае наиболее удобны слабые устройства (для них различие во времени зафиксировать легче).

Сетевой уровень. Он выступает как промежуточное звено между сенсорным уровнем и прикладным уровнем. Занимается передачей информации, а также отвечает за взаимодействие различных устройств и сетей между собой. Он уязвим для следующих атак:

- DoS-атака – создание условий для прекращения или затруднения доступа пользователей к вычислительной сети. Обычно это достигается путем искусственного наводнения сети большим количеством запросов.
- Атака посредника (man in the middle, MITM) – ретрансляция и изменение связи между узлами, которые считают, что общаются друг с другом. Является серьезной угрозой, так как атака может быть осуществлена в реальном времени.
- Атака на хранилище (storage attack) – как при централизованном, так и при распределенном способе хранения информации, она может быть украдена, подделана или удалена.
- Эксплойт-атака (exploit attack) – внедрение фрагментов кода, использующих уязвимости в системе безопасности, в приложении или в аппаратном обеспечении. Целью атаки является получение над контролем системы и кража информации, также возможно нарушение функционирования системы.

На прикладном уровне находятся все приложения, использующие технологию IoT. Атаки на данный уровень часто определяются конкретным назначением сети. Примеры атак:

- Межсайтовый скриптинг (cross-site scripting) – введение инъекции на стороне клиента. Эта атака позволяет злоумышленнику полностью изменить содержимое приложения в своих целях, а также украсть информацию.
- Атака вредоносного кода (malicious code attack) – вредоносный код в любой части приложения. Данные атаки часто предотвращаются антивирусом.
- Атаки на данные – в случае большого количества пользователей, в некоторых случаях данные могут передаваться в слабо защищенном виде, что может быть использовано злоумышленником.

С развитием технологии IoT, распространение получает более сложная, пятиуровневая модель. Модель выглядит следующим образом.

- уровень бизнес-логики (business level);
- прикладной уровень (application layer);
- уровень обработки (processing layer);
- транспортный уровень (transport layer);
- сенсорный уровень (perception layer, sensor layer).

Уровень обработки собирает, отбирает и обрабатывает информацию, полученную от транспортного уровня. Возможные атаки:

- Атака истощения ресурсов (resource exhaustion attack) – атака, в результате которой происходит не «зависание» и перегрузка устройств, как при DoS-атаке, а сбои программного или аппаратного обеспечения системы. Благодаря распределенной природе системы интернета вещей, эти атаки не слишком опасны.
- Вредоносное ПО – атака на конфиденциальную информацию. Вирусы, шпионское ПО, реклама, троянские программы и черви.

Уровень бизнес-логики занимается управлением всей системой. Устанавливает политики приложений, конфиденциальности данных, обработки данных. Уязвимость этого слоя позволит злоумышленникам «легально» использовать приложения в обход бизнес-логики. Возможные варианты атак:

- Атака на бизнес-логику (business logic attack) – использование ошибок программирования бизнес-уровня. Позволяет изменить взаимодействие между пользователем и БД приложения в сторону, выгодную злоумышленнику. Это может быть достигнуто при использовании уязвимостей кода, слабостей процедур валидации при восстановлении пароля и введении входных данных, или слабостей шифрования.
- Атака нулевого дня (zero-day attack) – использование уязвимостей системы безопасности, которые ранее были неизвестны.

1.3. Выводы

В данной главе была приведена основная терминологеская и теоретическая база, на которой будут строиться дальнейшие главы. Рассмотрены основные понятия криптографии и интернета вещей. Описана уровневая модель систем интернета вещей, рассмотрен каждый уровень и атаки на него.

ГЛАВА 2. ТЕОРЕТИЧЕСКИЙ АНАЛИЗ ЛЕГКОВЕСНЫХ КРИПТОГРАФИЧЕСКИХ АЛГОРИТМОВ

В данной главе проводится теоретический анализ алгоритмов легковесной криптографии. Вводится их определение, требования к ним. Анализируются возможные подходы к защите от различных видов атак на устройства интернета вещей. Определяются виды алгоритмов, наиболее подходящие для реализации данных подходов, а также сравниваются со стандартными протоколами "общего назначения"для этой цели.

2.1. Легковесные алгоритмы: понятие, требования

2.1.1. Понятие и область применения легковесной криптографии

Легковесная криптография (LW-криптография) изучает криптографические алгоритмы, ориентированные на использование в небольших вычислительных устройствах, заведомо не обладающих значительными ресурсами.

LW-криптография используется для обеспечения безопасности встраиваемых систем (англ. embedded systems). Встраиваемая система представляет собой микроконтроллер или небольшой компьютер, встроенный в управляемое устройство. Таким устройством может быть станок с ЧПУ или платежный терминал. Данные, которыми оперирует такое устройство, как платежный терминал, представляют собой существенную ценность, поэтому нуждаются в защите. Однако возможна разработка вирусов и «широкого действия», не нацеленных на определенное устройство, но могущих нанести большой вред из-за широкой распространенности встраиваемых систем. Примером вируса такого типа является RFID-вирус [**src22**].

Другим важным применением легковесной криптографии являются задачи обеспечения безопасности устройств интернета вещей. Этому варианту использования LW-криптография и посвящена данная работа.

В интернете вещей LW-криптография применяется главным образом для обеспечения безопасности устройств сенсорного уровня. В то время как безопасность узлов сетевого/транспортного уровня можно организовать на основе модели криптомаршрутизатора [**src23**], а остальные компоненты сети IoT (клиентские приложения прикладного уровня, сервера уровней обработки и бизнес-логики) обладают достаточными ресурсами для применения методов общей криптографии.

В контексте использования LW-криптографии в устройствах интернета вещей, эти устройства считаются «атомарными». Они только собирают данные, шифруют их и отправляют на узлы транспортного уровня. И защите подвергается главным образом отправляемый устройствами трафик. Среди основных направлений защиты пакетов данных можно выделить следующие.

- Подтверждение достоверности и целостности пакета данных. Требуется уверенность в том, что пакет отправил законный отправитель, и по пути данные не подверглись модификации злоумышленником или помехам канала передачи.
- Подтверждение подлинности пакета данных. Требуется уверенность в том, что пакет, который мы получаем, это именно тот пакет, который сейчас отправил законный отправитель. А не пакет, который он отправил ранее, а злоумышленник сейчас ретранслировал.
- Обеспечение конфиденциальности пакета данных. Требуется уверенность в том, что злоумышленник не смог расшифровать данные, даже если перехватит пакет.

Существенным фактором является *неинтерактивность* трафика. Это означает, что пакет передается устройством лишь однократно, без долгих процедур аутентификации. Неинтерактивность обуславливается, во-первых, весьма вероятным отсутствием постоянного канала беспроводной связи, а во-вторых, необходимостью быстро передать пакет данных при наличии такого подключения. Неинтерактивность сильно ограничивает количество доступных для использования криптографических методов и протоколов.

Можно выделить следующие требования к легковесным криптографическим алгоритмам и их реализации.

- Бизнес-требования:
    - требуемый уровень безопасности устройства;
    - стоимость устройства;
    - итоговая производительность устройства.
- Ограничения аппаратной части:
    - ограничения энергетических ресурсов;
    - ограничения по объему ОЗУ;
    - ограничения размера микросхемы (GE-мера).
- Ограничения программной реализации:
    - объем программного кода;

  - количество потребляемой оперативной памяти;
  - время работы.
- Возможные дополнительные ограничения:
  - ширина полосы рабочих частот и пропускная способность канала связи.

2.1.2. Бизнес-требования

Бизнес-требования определяют необходимость поиска компромисса между стоимостью, безопасностью и производительностью устройства. Достаточно легко оптимизировать два из трех свойств, а вот оптимизировать все три – обычно весьма трудная задача для разработчиков.

Процедура определения требуемого уровня безопасности определяется в соответствии с конкретной областью применения конкретной системы. В коммерческих структурах может использоваться, например, модель системы безопасности *с полным перекрытием*. В рамках данной модели перечисляются все защищаемые объекты системы и возможные атаки на них, а также вероятность появления и наносимый ущерб для каждой из атак. После чего определяются приоритетные направления защиты. В контексте систем интернета вещей представляется разумным ввести атаки, общие для всех уровней системы (например, DoS-атаки) и атаки, специфичные для конкретного слоя (например, кодовые инъекции нацелены главным образом на прикладной уровень). Атаки на каждый уровень и меры защиты от них кратко приведены в первой главе данной работы.

Стоимость и итоговая производительность устройства определяется, главным образом, используется ли в этом качестве устройство «общего назначения» (например, смартфон) или же некоторое «кастомное», уникальное аппаратное решение.

В первом случае и процессор, и системы сетевого взаимодействия, и иногда даже системы безопасности уже реализованы, и требуется реализовать только модули для сбора данных и для их шифрования. Обратной стороной являются возможные скрытые уязвимости или сознательно добавленные «бэкдоры» устройства, которые крайне трудно обнаружить. Кроме этого, использование функционала «из коробки» означает необходимость положиться на разработчиков этого функционала и потерю полного контроля над аппаратным обеспечением.

Во втором случае разработка начинается «с нуля» (или почти с нуля). С одной стороны, это требует гораздо больших навыков от команды разработчиков и больших затрат времени и денежных средств на разработку. Однако, с другой стороны, это может позволить создать в итоге более дешевое и надежное устройство – опять же в зависимости от уровня компетенций команды.

2.1.3. Требования к аппаратной части

В случае использования «кастомной» аппаратной реализации, аппаратная часть должна удовлетворять требованиям по количеству энергетических ресурсов, объему ОЗУ и площади микросхемы. При использовании аппаратного решения «общего назначения» количество имеющихся ресурсов обычно значительно больше, чем требуется для работы криптографических алгоритмов (но, естественно, тоже ограниченно, что необходимо учитывать).

Ограничения энергетических ресурсов могут явно задаваться конкретным технологическим решением (например, RFID-метки рассчитаны не более чем на 15 микроватт [**src24**]). Кроме того, для предупреждения *атак по энергетическим ресурсам* требуется по возможности уменьшить скачки в энергопотреблении устройства. Это отдельная тема, выходящая за рамки данной работы.

Ограничения размера (площади) микросхемы связаны с тем, что зачастую в слабых устройствах (в том числе в устройствах интернета вещей) используются «кастомные» аппаратные решения, направленные на удешевление и повышение надежности устройств. Это означает, что требуется, насколько возможно, уменьшить количество используемых функциональных элементов. Оно определяется, во-первых, используемым алгоритмом, а во-вторых, его программной и аппаратной реализацией. В качестве единицы измерения данного ресурса используется количество элементов NAND (Not And), необходимое для реализации. Обозначается эта величина GE (logic Gate Elements number, количество логических вентилей). Существуют следующие категории реализаций алгоритмов [**src25**]:

- легковесные реализации (lightweight) – не более 3000 GE;
- низкостоимостные реализации (low-cost) –не более 2000 GE;
- ультралегкие реализации (ultra-lightweight) – не более 1000 GE.

При создании конкретной аппаратной реализации необходимо искать баланс между скоростью работы и размером микросхемы. Оптимизация размера микросхемы достигается за счет использования последовательной архитекту-

ры (обрабатывающей информацию побайтно), оптимизация скорости – за счет распараллеливания и конвейеризации, что влечет увеличение размера.

Основной стандартом, определяющим количество доступных ресурсов для легковесных криптографических алгоритмов, является [**src26**].

2.1.4. Требования к программной части

При разработке программной реализации алгоритма необходимо оптимизировать те же направления, что при разработке любого ПО: оптимизация потребления памяти и времени, а также специфичный для данной области параметр – объем кода.

Количество потребляемой памяти особенно критично для «кастомных» реализаций, при использовании более мощной аппаратной части уже не столь критично, хотя все равно подлежит оптимизации.

Время работы состоит из двух величин. Задержка (англ. latency) – время инициализации алгоритма, и пропускная способность (англ. throughput) – количество информации, обрабатываемой в единицу времени. Обычно требования по каждому из этих параметров формулируются отдельно. Например, для систем автоматического осуществления дорожных сборов время реакции устройства должно быть менее 10 миллисекунд [**src27**], это ограничение на задержку. С другой стороны, если система должна еще и зафиксировать изображение машины, нарушившей ПДД, то требуется передать (а значит, зашифровать) достаточно большое количество информации, а это требует относительно большой пропускной способности.

Объем программного кода тесно связан с количеством используемого кода. Так, S-блоки, определяющие порядок перестановки блочного шифра, можно хранить в оперативной памяти, а можно напрямую «зашить» в код. Второй способ быстрее, однако исполнение кода тоже требует определенных накладных расходов (при исполнении код также хранится в ОЗУ). Поэтому конкретное соотношение хранимых в ОЗУ и заданных в коде данных определяется под конкретный сценарий использования.

2.2. Использование криптографических методов в легковесной криптографии

Как указывалось выше, для защиты трафика устройств IoT необходимо решить следующие задачи:

– подтверждение достоверности и целостности пакета данных;
– подтверждение подлинности пакета данных;
– обеспечение конфиденциальности пакета данных.

Проанализируем основные типы криптографических алгоритмов на пригодность для решения этих задач.

2.2.1. Подтверждение достоверности и целостности пакета

Фактически эта задача является задачей аутентификации. Она распадается на две части: аутентификация отправителя и аутентификация пакета.

В системах обмена пакетами общего назначения (не «легковесных») обычно используются протоколы аутентификации отправителя. В начале общения с помощью одного из протоколов (CHAP, Kerberos и другие) подтверждается подлинность отправителя или обеих сторон. После чего начинается доверенный обмен сообщениями. Спустя некоторое время требуется повторная аутентификация. Такая схема используется на следующих уровнях модели OSI: канальный (например, протоколы EAP, PPP), сеансовый (PAP) и прикладной (SSH).

Для использования в системах IoT такой метод не слишком подходит, так как пакеты могут отправляться через краткие, длительные или неравные промежутки времени. При этом каждый пакет может быть отправлен или модифицирован злоумышленником. Это обстоятельство диктует необходимость подтверждения достоверности и целостности каждого передаваемого пакета данных. Это позволяет распознать факт введения фальшивого узла.

Аутентификация также может производиться с использованием таких инструментов, как хешированный или нехешированный пароль, а также электронная цифровая подпись (ЭЦП). Эти методы подходят для легковесной криптографии.

ЭЦП применяется следующим образом. Имеется закрытый ключ, он хранится на отправляющем устройстве, и открытый ключ, он хранится на принимающем устройстве и сопоставлен отправляющему устройству. Открытый ключ однозначно вычисляется по закрытому.

A. Вычислить контрольную сумму (хэш) пакета данных.
B. Подписать хэш ЭЦП (т. е. зашифровать хэш закрытым ключом, получив в результате подпись).
C. Отправить пакет данных, снабдив ЭЦП. Принимающая сторона расшифровывает ЭЦП с помощью открытого ключа. Если получился верный хэш документа, значит хэш был зашифрован с помощью верного закрытого ключа, что подтверждает законность отправителя.

Закрытый ключ известен только отправляющему устройству. Злоумышленник его не знает и, следовательно, правильно зашифровать хэш не сможет.

Цифровая подпись, в отличие от пароля, еще и гарантирует целостность доставленного сообщения (защита от случайных или намеренных искажений по пути). Поэтому она является более предпочтительной, хотя и требует бОльших расходов времени, т. к. требуется предварительно вычислить хэш сообщения (зависит от длины сообщения, т. е. влияет на пропускную способность). Шифрование хэша требует уже фиксированного времени (т. е. влияет только на задержку).

2.2.2. Подтверждение подлинности пакета

Такое подтверждение необходимо для защиты от атаки повторного воспроизведения. В существующих системах представлены различные способы обеспечения такой защиты.

Один из способов реализован в протоколе Kerberos [**src28**]. Сообщение снабжается сроком действия и временной меткой. Если срок действия истек, можно соответствующим образом отреагировать: отбросить пакет, зафиксировать подозрительную активность или даже заблокировать трафик от скомпрометированного отправителя.

Схожий подход под названием Hop-by-hop transport [**src29**] используется в IPv4/IPv6 маршрутизации. Заголовок пакета (IP-заголовок) содержит специальное восьмиразрядное поле, в IPv4 оно называется TTL (Time to live), в IPv6 – Hop Limit. Это поле содержит максимальное количество прыжков (передач пакета между маршрутизаторами), и после каждого прохождения через маршрутизатор оно уменьшается на единицу. Если оно достигает нуля, пакет считается устаревшим и отбрасывается.

Другим возможным подходом является добавление так называемой «nonce–вставки». Nonce-вставка (от англ. number that can be used only once) – случайно

сгенерированное число, известное обеим сторонам и хранимое ими в тайне. Генерируется сервером, отправляется клиенту, который затем добавляет его к паролю при шифровании. Используется однократно, после чего меняется. Повторение или использование неверной nonce-вставки говорит о том, что пакет отправлен злоумышленником. Не подходит по причине неинтерактивности трафика, передаваемого устройствами IoT (устройство не ждет сигнала сервера, вся коммуникация состоит в одном пакете).

Таким образом, оптимальным вариантом является добавление к зашифрованному сообщению метки времени. Эта операция не является требовательной ни по времени, ни по памяти. Но необходимо наличие системного таймера. При этом исходим из того, что злоумышленник не сможет расшифровать сообщение, а значит и изменить метку не может. Кроме того, метку можно подписать цифровой подписью. Тогда злоумышленник не сможет ее изменить даже в случае, если сможет прочитать данные.

2.2.3. Обеспечение конфиденциальности пакета

Эта задача является одной из важнейших и наиболее сложных задач, которые необходимо решить при организации защиты данных, передаваемых IoT–устройствами. Данная задача является основным объектом изучения легковесной криптографии.

Итак, задача состоит в шифровании пакета данных. Ясно, что шифрование должно быть обратимым преобразованием. Принимающая сторона должна иметь возможность эффективно (достаточно быстро) расшифровать пришедший пакет данных. Как указывалось выше, требования к скорости работы определяются задержкой (определяется минимальным интервалом между сообщениями) и пропускной способностью (объем данных в пакете, который должен быть зашифрован до начала шифрования следующего пакета). Шифратор не должен потреблять больше доступного количества оперативной памяти, должен тратить как можно меньшее количество энергии (либо энергопотребление должно быть в пределах возможностей устройства, но близким к постоянному, чтобы затруднить криптоанализ по колебаниям энергопотребления). При этом следует помнить о необходимости минимизации требуемого размера микросхемы (хотя бы не более 3000 GE, в идеале – не более 1000 GE) и обеспечении максимального уровня стойкости шифрования.

Таким образом, задача формулируется следующим образом. Какие алгоритмы позволяют достичь максимального уровня производительности и стойкости, требуя при этом а) не более 3000 GE и б) не более 1000 GE?

При выборе способа шифрования в первую очередь следует проверить, не могут ли подойти известные криптоалгоритмы общего назначения. Если удастся, не изменяя сами алгоритмы, выполнить их программную и аппаратную реализацию, работающую в заданных условиях, это будет наилучшим вариантом. Потому что они уже хорошо исследованы, что позволяет рассчитывать на их стойкость.

Следующим шагом должно быть рассмотрение известных (в том числе закрепленных в стандартах, о них ниже) легковесных криптоалгоритмов. И только если и они не подходят, можно попробовать модифицировать их в сторону ослабления и дальнейшего облегчения. Впрочем, это весьма рискованный путь, так как требует от создателей значительных навыков в криптоанализе, чтобы гарантировать стойкость шифра.

Итак, ниже будут рассмотрены на пригодность для легковесного шифрования различные типы шифров общего назначения и легковесных шифров.

2.2.3.1. Блочное шифрование

В дело исследования блочных легковесных шифров внесли значительный вклад работы [**src30**; **src27**]. Легковесные блочные шифры являются одним из наиболее динамично развивающихся разделов низкоресурсной криптографии. Исследования идут по двум направлениям: создание оптимальных реализаций алгоритмов общего назначения и создание новых алгоритмов, нацеленных именно на использование в низкоресурсных устройствах.

Ниже будут отдельно рассмотрены *блочные шифры общего назначения* и *легковесные блочные шифры*.

Блочные шифры общего назначения. Одним из лучших на данный момент блочных шифров является алгоритм *AES (Rijndael)*. Он оперирует 128-битными блоками данных и 128-, 192- и 256-битными ключами, производя 10, 12 и 14 раундов шифрования соответственно.

Помимо высокой криптостойкости, он является весьма производительным. Существует его программная реализация с производительностью порядка 7 процессорных тактов на байт [**src31**] на стандартных ЦП. Добавление специальной

процессорной инструкции для этого алгоритма позволило достичь производительности примерно 0.7 тактов на байт [**src32**]. Помимо этого, стоит отметить низкое потребление памяти.

Он, однако, требует достаточно большого размера микросхемы, порядка 250 000 GE для достижения максимальной скорости (до 70 Гб/сек) [**src33**]. Наиболее компактная последовательная реализация требует 2400 GE и имеет производительность 226 циклов на блок [**src34**]. Это значительно лучше, и может подойти для многих устройств, даже очень простых. Тем не менее, «ультралегким» алгоритмом, т. е. оптимальная реализация которого требует менее 1000 GE, его все-таки назвать нельзя.

DES. Данный алгоритм также является весьма известным. Длина его ключа значительно меньше, чем у AES (56 бит), что означает меньшую стойкость. Также он был взломан методом линейного криптоанализа [**src35**]. В системах общего назначения (не легковесных) он применяется, в основном, в виде *Triple DES*, то есть троекратное шифрование с тремя ключами. Также существует вариация *DESX*, работающая с 184-битным ключом при таком же размере блока и числе раундов, как у DES. Наиболее компактные реализации DES и DESX требуют 2309 GE и 1848 GE соответственно [**src36**].

Легковесные блочные шифры. Наилучшие лешковесные блочные алгоритмы включены в стандарт [**src26**] (часть 2, Block ciphers). Это шифры *PRESENT* и *CLEFIA*.

CLEFIA. Данный алгоритм является очень популярным благодаря существующим очень производительным реализациям. Он оперирует 128-битными блоками с длиной ключа 128, 192 и 256 бит с 18, 22 и 26 раундами, соответственно. Наиболее компактная реализация шифрования требует 2488GE, дешифровки – 2604GE [**src37**]. В данной реализации используется технология *Clock Gating*. С точки зрения криптоанализа данная функция является весьма стойкой, существующие не вероятностные атаки лишь немногим лучше, чем полный перебор [**src38**].

PRESENT. Данный шифр использует 80- и 128-битные ключи для шифрования данных 64-битными блоками в 31 раунд. Он, вообще говоря, не совсем укладывается в границу, т. к. требует 1030 GE [**src39**]. Но даже такой результат явился почти революционным. Его удалось достичь за счет последовательной архитектуры – это первый шифр с такой архитектурой. Еще одним улучшением является уменьшение числа S-боксов с 8 (обычное количество для блочных шифров) до 1.

Платой за компактность данного алгоритма является не слишком высокая криптостойкость. Существуют атаки по сторонним каналам [**src40**; **src41**] и атака на связанных ключах [**src42**] на 17-раундовую версию данного алгоритма. Существует атака методом *DFA* (differential fault cryptanalysis) [**src43**]. Есть дифференциальная атака на 26-раундовую версию [**src44**].

Существуют и еще более компактные шифры. Это *Katan* (800-1000GE [**src45**]), *Ktantan* (460GE-690GE [**src45**]) и другие. Однако такие реализации оперируют ключами очень небольшой длины (32 или 48 бит), что позволяет взломать их простым перебором, либо они уязвимы для других атак. Такие алгоритмы подходят для шифрования не слишком важной информации.

2.2.3.2. Потоковое шифрование

Данное направление легковесной криптографии развито слабее, чем блочное шифрование. Основная причина в том, что потоковые шифры направлены главным образом на шифрование больших объемов информации сразу, в то время как легковесные устройства зачастую оперируют пакетами данных небольшого объема. Потоковые шифры часто имеют большое время инициализации и требуют большого количества памяти для хранения внутреннего состояния.

В стандарт [**src26**] (часть 3, Stream ciphers) включены два алгоритма. Это *Enoroco* и *Trivium*. Помимо закрытого ключа, эти алгоритмы используют также вектор инициализации, являющийся открытым ключом. Они требуют примерно 4900 [**src46**] и 1300 [**src47**] тактов процессора на инициализацию, соответственно. Занимаемое место на микросхеме – 4100GE и 2600GE [**src48**].

Алгоритм Trivium является весьма гибким: можно варьировать соотношение между числом логических элементов (GE) и скоростью работы. Наиболее компактная реализация занимает всего 700GE [**src49**]. К настоящему времени не известно атак быстрее полного перебора.

2.2.3.3. Асимметричное шифрование

Асимметричное шифрование не слишком подходит для шифрования трафика, передаваемого устройствами интернета вещей. Среди их недостатков выделяется бОльшая длина ключа ([**src50**], стр 673), более низкая скорость шифрования, бОльшие вычислительные ресурсы. В то же время их преимущества (нет необходимости передачи секретного ключа, ключ дешифрования может хранить только

одна сторона) диктуют область их применения главным образом как средство распределения систем или аутентификации. Их использование в системах ЭЦП описано выше.

Впрочем, существуют достаточно эффективные реализации асимметричных алгоритмов, которые можно использовать при создании ЭЦП. Например, алгоритм *cryptoGPS* (GPS – по фамилиям авторов) имеет реализацию, требующую 724 такта (на все шифрование) при размере схемы 2876GE [**src51**]. По стойкости она соответствует 80-битному симметричному ключу.

2.2.3.4. Хэширование и ЭЦП

Криптографическое хэширование однозначно не подходит для целей шифрования, так как является односторонним преобразованием: расшифровка хэша априори невозможна, возможно только сравнение хэшей. Впрочем, блочная шифрующая функция может быть интересна с криптографической точки зрения, и может быть использована как примитив при создании нового шифра, в том числе легковесного. Например, на базе алгоритма Present создана легковесная хэш-функция *H-Present-128* [**src52**].

Электронная цифровая подпись также не подходит для хэширования, так как использует в своем составе хэш. ЭЦП предназначены для гарантирования достоверности и целостности данных. В этом качестве она вполне может быть использована, о чем написано выше.

2.3. Выводы

В данной главе был проведен теоретический анализ алгоритмов легковесной криптографии. Введено их определение, требования к ним. Проанализированы возможные подходы к защите от различных видов атак на устройства интернета вещей. Определены виды алгоритмов, наиболее подходящие для реализации данных подходов, а также проведено их сравнение со стандартными протоколами "общего назначения"для этой цели.

ГЛАВА 3. МЕТОДОЛОГИЯ СРАВНЕНИТЕЛЬНОГО АНАЛИЗА ЛЕГКОВЕСНЫХ КРИПТОГРАФИЧЕСКИХ АЛГОРИТМОВ

В данной главе описывается методология сравнительного тестирования различных криптографических алгоритмов по производительности (времени работы) и по мощности энергопотребления.

3.1. Общие принципы. Теоретический анализ

Сравнительный анализ легковесных криптографических алгоритмов разбивается на два этапа. Первый этап представляет собой теоретический анализ алгоритмов, а второй – тестирование реализаций алгоритмов.

Теоретический анализ включает в себя оценку и сравнение их стойкости в зависимости от параметров (*задача криптоанализа*), а также теоретическую оценку их производительности в зависимости от тех же параметров. Под параметрами понимается конкретная конфигурация: длина ключа, число раундов, в некоторых алгоритмах также существуют различные режимы запуска и свободные параметры. Также может (и должна!) быть исследована стойкость алгоритма в зависимости от конкретного входа и конкретного ключа (слабые ключи и т. д.).

Для определения криптостойкости легковесных шифров подходят такие же методы криптоанализа, как для шифров общего назначения. Они могут быть самыми различными: линейный криптоанализ, дифференциальный криптоанализ, атаки встречи посередине и многие другие. Они описаны в большом числе работ, например, в [**src53**; **src54**; **src55**; **src56**]. Для теоретической оценки производительности можно применять обычные методы оценки времени работы алгоритма. Они описаны, например, в [**src57**].

В рамках тестирования реализаций алгоритмов определяют качество конкретных реализаций по таким характеристикам, как требуемая площадь микросхемы (GE-мера), фактическая производительность (производительность – число байт в секунду, плюс задержка (микро- или наносекунд)), а также энергопотребление (микроватт). Потребление ОЗУ обычно не требуется тестировать: оно легко определяется уже по псевдокоду алгоритма.

При сравнительном тестировании реализаций легковесных алгоритмов представляется разумным придерживаться следующих принципов.

- Следует сравнивать реализации алгоритмов с такими параметрами запуска, которые обеспечивают одинаковую стойкость. Это требует предварительного серьезного криптоанализа тестируемых алгоритмов.
- Следует сравнивать реализации, использующие архитектуры одного вида. То есть сравнивать только между собой последовательные, параллельные реализации, реализации с конвейеризацией, с кэшированием и другие.
- Инструменты сравнения производительности должны быть постоянными (фиксированными), не «оказывать предпочтения» какому-либо из сравниваемых алгоритмов, а также минимально влиять на производительность устройств.
- Результаты сравнения должны быть воспроизводимыми. Поэтому следует по возможности минимизировать использование зависимостей от библиотек и фреймворков (при тестировании на ПК), а также публиковать или подробно описывать программный код реализаций и конфигурацию аппаратной части.

При сравнительном тестировании на ПК не представляется возможным протестировать площадь микросхемы, это можно проверить уже на аппаратной или аппаратно-программной реализации. Однако существуют методы приблизительной оценки производительности и энергопотребления. После чего полученные результаты для двух алгоритмов подвергаются сравнению.

3.2. Оценка производительности на ПК

Производительность наиболее легко поддается измерению. Методика тестирования понятна: реализовать алгоритм, после чего запустить его несколько раз и измерить время работы. То же самое проделать для другого алгоритма и сравнить результаты.

Однако требуется, чтобы результаты по возможности были близкими к показателям на реальных малоресурсных устройствах. Ясно, что конкретные временнЫе показатели на ПК будут отличаться от таковых на низкоресурсных устройствах, однако целью является получение соотношения между производительностями алгоритмов, по возможности наиболее близкого к соотношению на низкоресурсных устройствах.

Существуют объективные ограничения точности такого способа тестирования. Фактически, они относятся к систематической (методологической) погрешности данного метода тестирования. Можно выделить следующие.

В то время как набор процессорных инструкций для устройств общего назначения более-менее стандартизирован (стандарт x86/x86-64 для ПК, ARM для мобильных и планшетных устройств), набор инструкций «кастомного» решения может несколько или даже значительно отличаться. Поэтому нужно минимизировать использование «необычных» процессорных команд. Лучше всего использовать для программной реализации язык ассемблера, это позволяет исключить процедуру трансляции языка С (бонусом также является отсутствие оптимизации последовательности команд, которая обычно происходит при компиляции программы на С). Также возможно использование языка С с отключенной оптимизацией. Наихудшим вариантом является использование наиболее высокоуровневых языков вроде C++ или Java, так как порядок их трансляции в машинные команды весьма запутанный.

Также при создании ассемблерной реализации следует учитывать предполагаемые аппаратные ограничения реализации. Они могут быть следующими:

- уменьшенное количество регистров (по сравнению с ЦП ПК) [**src58**];
- иная архитектура, чем у ЦП ПК. В этом случае можно моделировать задержки с помощью пустых циклов [**src59**].

Другой проблемой является различная скорость исполнения инструкций, т. е. задержка и пропускная способность процессора при исполнении данной инструкции. Процессоры различных производителей, линеек и поколений могут показывать различную производительность при исполнении одной и той же инструкции. Это касается стандартных ЦП для десктопных машин и ноутбуков, и уж тем более касается «кастомных» процессорных решений. Эта проблема не поддается простому решению. Можно пользоваться уже готовыми таблицами [**src60**], однако кастомные реализации нуждаются в отдельном тестировании для каждой используемой команды.

Еще одним неочевидным моментом является многопоточность. Ведь, даже если алгоритм запускается в однопоточном режиме, на его исполнение выделяется не 100% процессорного времени. Значительная часть времени уходит на исполнение задач других приложений, прерываний ОС и системных прерываний. Для минимизации времени работы процессора над другими задачами следует по возможности отключить все другие приложения и службы перед началом тестирования. Можно засекать определенное время, которое затем умножить на долю

q процессорного времени, выделяемого тестируемому процессу. Либо же можно сразу измерять процессорное время, например, как описано в статье [**src61**]. Также следует отключить технологию Hyper-Threading [**src62**].

Еще одним возможным подходом является искусственное понижение рабочей частоты процессора. Однако этот способ имеет свои нюансы. Так, это можно сделать на уровне ОС или утилит, и в этом случае появится дополнительная активная служба, которая будет требовать отдельных ресурсов ЦП, возможно не в постоянном количестве. Также это можно выполнить на уровне BIOS [**src63**]. Этот способ является более предпочтительным, так как не влечет появления новых служб, нагружающих ЦП. Однако этот способ (как и предыдущий) необходимо использовать с осторожностью, так как при понижении частоты ЦП некоторые службы могут отключаться. Это означает необходимость перерасчета доли процессорного времени, выделяемого тест-модулю, для каждого значения тактовой частоты ЦП.

Могут быть предложены два способа проведения измерения производительности алгоритмов. Можно либо рассчитать ее через время исполнения отдельных операций (частота процессора известна, задержки и число тактов на исполнение для каждой операции известны), либо же измерить напрямую. По мнению автора, первый подход даст очень низкую точность, либо потребует значительного тестирования или глубоких знаний в устройстве конкретного ЦП. Это связано с тем, что в процессоре имеется масса оптимизаций, которые между собой взаимодействуют. В качестве примера: последовательность команд xor rax rbx;xor rcx rdx займет время, равное не удвоенному времени исполнения xor'a, а лишь чуть дольше одного xor'a. Это связано с внутренней параллелизацией работы процессора. Большое количество разных оптимизаций не позволяет точно определить, как поведут себя операции в конкретной взаимной конфигурации, т. е. в конкретном алгоритме. Впрочем, это тема для отдельного исследования о предсказуемости времени исполнения последовательности процессорных команд.

В данной работе предлагается определять время работы алгоритма с помощью прямого тестирования времени. Случайную (статистическую) погрешность данного измерения можно уменьшить следующим способом. Произвести несколько серий измерений времени шифрования для различного объема шифруемых данных. Чем больше серий и измерений внутри серии, тем лучше. В результате точки на графике зависимости времени шифрования от объема шифруемых данных будут расположены примерно на одной прямой. После чего с помощью

метода наименьших квадратов можно восстановить уравнение этой прямой вида y=kx+b. Коэффициент k будет пропускной способностью, а b – задержкой данного алгоритма. Подробнее данный классический метод описан, например, в пособии [**src64**].

3.3. Оценка энергопотребления на ПК

Оценка энергопотребления реализации алгоритма на ПК является методологически более трудной задачей, чем оценка производительности. Основная причина в том, что на компьютере всегда в многозадачном режиме исполняются различные службы, некоторые из которых потребляют разное количество энергии в разное время. Кроме того, порядок работы служб может зависеть от загрузки ЦП, и даже от энергопотребления (например, ACPI). В результате сложно отличить колебания энергопотребления служб от работы алгоритма. Экспериментальная оценка влияния служб на энергопотребление проведена в работе [**src65**]. Единственный способ минимизировать влияние служб на энергопотребление – отключить их.

Далее используются следующие обозначения: P – мощность энергопотребления (ватт), E – потребленная энергия (ватт-час).

После отключения максимально возможного количества служб можно приступать к собственно оценке энергопотребления алгоритма. Вначале опишу методика измерения энергопотребления при исполнении задачи, далее приводится порядок измерения энергопотребления на конкретном алгоритме.

Можно выделить два компонента энергопотребления ПК.

A. Энергопотребление компонентов P_{const}. Оно является независимым от загрузки ПК и выполняемых задач. Под компонентами имеются в виду вентилятор, материнская плата, ОЗУ (согласно), видеокарта (если не производятся вычисления на ней и не рисуются сложные графические объекты), HDD.
B. Энергопотребление ЦП P_{cpu}. Оно в разные моменты разное: при исполнении тестируемого процесса имеет величину P_{test} (собственно, ее и следует измерить), при исполнении служебных (не относящихся к процессу тестирования) процессов – усредненную величину P_{serv}.

Существует два способа измерения энергопотребления процессора: измерить энергопотребление всего ПК (на уровне блока питания) и затем вычесть P_{const}, или же напрямую измерить энергопотребление процессора [**src66**; **src67**] (учитывается

потребление на входе системной платы по разъему питания процессора [**src68**]), при этом следует вычесть энергопотребление регулятора напряжения. Второй подход предпочтителен.

После того, как энергопотребление процессора найдено, требуется найти P_{test}, т. е. каким было бы энергопотребление, если бы выполнялся только тестируемый процесс. Если q – доля процессорного времени, выделяемого тестируемому процессу, то усредненное по времени энергопотребление ЦП равно

$$P_{cpu} = qP_{test} + (1 - q)P_{serv}. \tag{3.1}$$

Откуда получаем измеряемую величину:

$$P_{test} = 1/q(P_{cpu} - (1 - q)P_{serv}). \tag{3.2}$$

Величину P_{serv} можно найти как энергопотребление ЦП в состоянии простоя, разделенное на долю его загрузки в этом режиме (утилиты для определения загрузки приводятся в [**src69**]). Способ оценки величины q описан в предыдущем разделе (оценка производительности).

Далее описывается измерение энергопотребление для конкретного алгоритма. По этому вопросу существует работа [**src71**]. На ее основе предлагается следующий способ.

Измеряется $P_{bitshift}$, P_{xor}, ... – энергопотребление ПК при исполнении каждой из используемых в алгоритме инструкций: побитовый сдвиг, XOR и т. д. Каждый сеанс снятия этих данных следует проводить достаточно долго (хотя бы 5 минут), чтобы учесть колебания энергопотребления служб. После проведения достаточного числа таких сеансов можно узнать величину энергопотребления (микроватт) на исполнение каждой из инструкций, а также погрешность этой величины.

Измеряется частота процессора F. Это можно сделать, например, с помощью утилиты CPU-Z. По таблице (например, [**src60**]) определяется n – число тактов на инструкцию (пропускная способность ЦП для данной инструкции). Откуда энергия, затрачиваемая на одну операцию, равна

$$E_{op} = \frac{nP_{op}}{F}. \tag{3.3}$$

Энергия на шифрование блока или байта вычисляется как сумма энергозатрат на каждую операцию. Для обычных алгоритмов это не так просто, так как они

часто имеют ветвление, в этом случае для разных веток может быть разное энергопотребление. Но шифры обычно не имеют ветвления.

$$E_{alg} = \sum E_{op} \tag{3.4}$$

Если ранее была найдено время шифрования одного блока или байта, то можно найти энергопотребление алгоритма:

$$P_{alg} = \frac{E_{alg}}{t_{alg}}. \tag{3.5}$$

Если погрешности косвенных измерений окажутся не слишком большими, то можно, зная энергопотребление каждой операции в отдельности, вычислять энергопотребление криптоалгоритма целиком без прямого тестирования. В противном случае результат можно получить и путем прямого замера, но его потребуется проводить для каждого алгоритма.

3.4. Порядок сравнения избранных LWC-алгоритмов

В рамках практической части данной работы тестируется производительность избранных алгоритмов легковесной криптографии. Тестирование будет производиться на ПК с использованием вышеизложенной методологии. Результатом тестирования будет приблизительное соотношение производительностей алгоритмов. Далее полученное соотношение сравнивается с реальным соотношением производительностей для различных низкоресурсных программно-аппаратных реализаций этих же алгоритмов, полученных авторами других работ.

По результатам этого сравнения может быть сделан вывод об успешности или неуспешности предложенной методологии. Успешной она будет считаться в том случае, если результаты тестирования на ПК будут хоть в некоторой степени соотноситься с результатами сравнения реальных программно-аппаратных реализаций.

Неуспешность методологии будет означать, что тестирование производительности на ПК не отражает реальное соотношение, либо отражает в слабой степени. Успешность будет означать состоятельность данной методологии. Это позволит «прикинуть» соотношение скоростей алгоритмов-кандидатов на реализацию в конкретном программно-аппаратном решении еще до начала конструкторских работ по его созданию. Это, в свою очередь, позволит уменьшить время и стоимость таких работ.

3.5. Выводы

В данной главе была описана методология сравнительного тестирования различных криптографических алгоритмов по производительности (времени работы) и по мощности энергопотребления.

ГЛАВА 4. ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ ПРЕДЛАГАЕМОЙ МЕТОДИКИ ТЕСТИРОВАНИЯ

В данной главе описывается порядок апробации предложенной методики сравннительного тестирования реализаций алгоритмлв путем сравнительного тестирования пропускной способности трех реализаций алгоритма AES на языке C. Анализируются результаты тестирования.

4.1. Инфраструктура тестирования и обработки результатов

Реализация избранного алгоритма AES исполнена на языке Си. Выбран не ассемблер, так как Си значительно проще для реализации и отладки, а трансляцию его команд в ассемблер несложно увидеть, например, с использованием средств отладки IDE Visual Studio. Данный алгоритм выбран по причине его высокой стойкости, сочетающейся с существованием достаточно легковесных реализаций. Впрочем, предложенным способом может быть протестирован любой криптоалгоритм.

Также используются некоторые библиотеки языков C и C++. Это модуль time.h для измерения времени, модуль windows.h для точного измерения времени, а также stdlib.h для получения псевдослучайных чисел и vector.h для удобной работы с массивами переменной длины.

Тестированию подвергаются три реализации алгоритма AES на языке Си. Это две реализации, взятые из открытых источников ([**src72**; **src73**]), а также авторская реализация. Все три реализации были предварительно протестированы. Разработка авторской версии велась с упором на компактность кода и минимизацию «лишних» операций и потребления ОЗУ. На её основе может быть создана ассемблерная реализация. Причем тестировать ее можно с использованием того же окружения C/C++. В этом случае необходимо использовать массив чисел – опкодов процессорных команд.

Порядок тестирования времени исполнения таков. Для каждого алгоритма производится n серий измерений времени шифрования. На каждой из серий производится m измерений для различных объемов V_i входных данных, $i = 1 \ldots m$. Под входными данными понимается набор случайно сгенерированных блоков (количеством V_i), которые последовательно подвергаются шифрованию, ключ для каждого блока свой, он тоже генерируется случайно. Для уменьшения влияния

погрешности, каждое измерение выполняется t раз, время суммируется. Итого для каждого алгоритма имеется n рядов по m точек.

Необходимо принимать меры, предотвращающие «заоптимизирование» вызовов тестируемых функций, работающих «вхолостую». То есть требуется сделать так, чтобы оптимизатор не удалил блоки кода (вызов функции), результат которых никак не используется. Меры следующие:

A. Есть специальный контрольный байт, значение которого обновляется при каждом шифровании (увеличивается на значение первого байта шифротекста). Он выводится в консоль. Время обновления контрольного байта и вывода его на экран в общем времени не учитывалось.
B. На каждом из t измерений используются разные входные данные (но одного объема).

Еще одна особенность тестирования в том, что при большом количестве вызовов одной и той же функции подряд время ее исполнения уменьшается. Возможно, это связано с тем, что Windows требуется время, чтобы предоставить все требуемые ресурсы. Этот эффект оказывает влияние на результаты тестирования, причем чем больше объем данных, тем это влияние сильнее. Учесть его влияние не представляется возможным. К счастью, в данном тестировании его влияние не так велико, как могло бы быть, поскольку тестируемые алгоритмы планируется использовать для шифрования пакетов данных небольшого размера.

Следующим этапом идет обработка результатов измерения. Она производится на языке Python. Затем для каждой из серий измерений по m точкам строится прямая $y = kx + b$. Она строится с помощью метода наименьших квадратов. Фиксируются ее значения k и b и их погрешности. Таким образом фиксируются n пар k и b. Наконец, вычисляется итоговые значения k и b. Это и будут пропускная способность и задержка алгоритма, соответственно.

4.2. Результаты тестирования

При тестировании использовались следующие параметры.

Параметр	Значение
n	10
t	100
m	11
V_i	10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30

Для алгоритма AES размер блока составляет 128 бит (16 байт), размер ключа также 128 бит.

Полученные данные (временной ряд 1) для каждой из трех реализаций приводится на рис. 4.1-4.3. Можно видеть, что точки весьма хорошо ложатся на прямую, как и следовало ожидать. В реализациях 1 и 2 первый тест первой серии занимает больше времени, чем следует из теоретической модели. Возможно, это связано с тем, что Windows требуется время, чтобы предоставить все требуемые ресурсы.

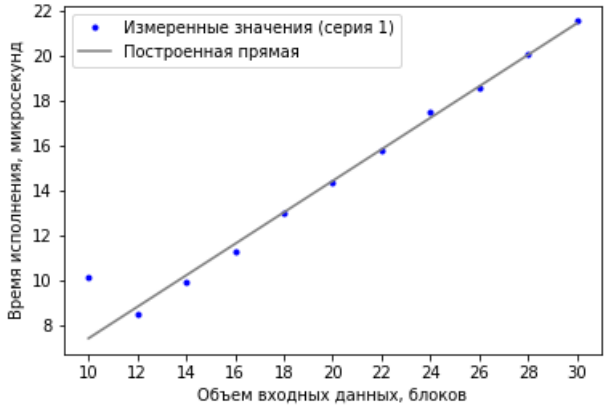

Рис.4.1. Применение метода для реализации 1

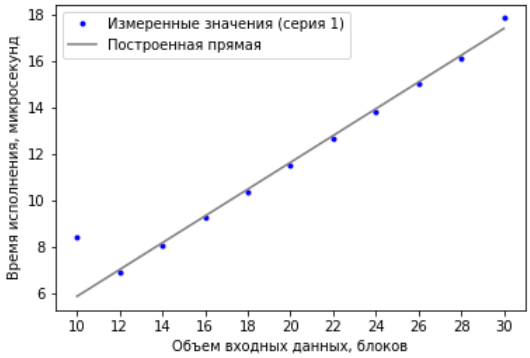

Рис.4.2. Применение метода для реализации 2

Для сравнения, на рис. 4.4 приводятся данные для реализации 1 с увеличенным числом блоков (примерно на порядок). Против ожидания, результаты не исказились: точки по-прежнему соответствуют теоретической модели, причем

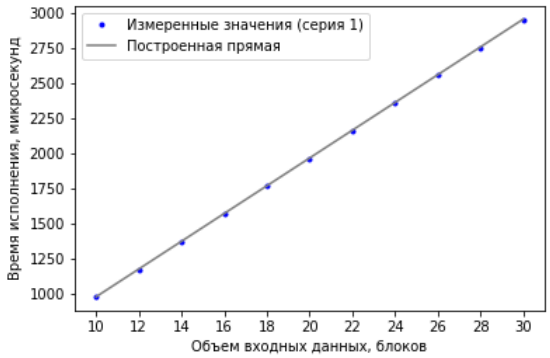

Рис.4.3. Применение метода для реализации 3

восстановленные коэффициенты прямой получены почти такие же (см. далее). Это говорит в пользу предложенной методологии: она устойчива к увеличению объема входных данных. Исследовано поведение до объема 600 блоков (9.6 Кб). Дальнейшее увеличение представляется нецелесообразным, так как низкоресурсные устройства обычно не передают пакетов такого большого размера.

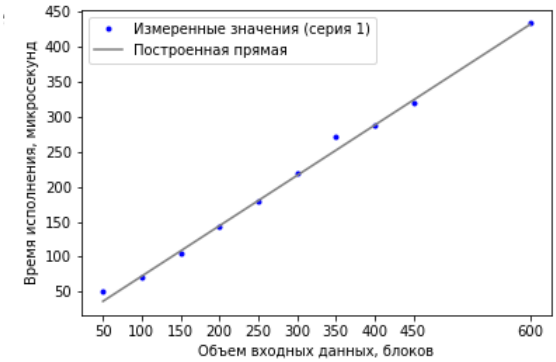

Рис.4.4. Применение метода для реализации 1 (число блоков увеличено)

Также на рис. 4.1-4.4 приводятся усредненные экстраполированные прямые. Они демонстрируют неплохую робастность (устойчивость к выбросам, регулярно появляющимся при тестировании алгоритмов). Так, на рис. 4.4 точка, соответствующая 350 блокам, является выбросом.

Восстановленные уравнения прямых (для объемов 10-30 блоков):

$$y_1 = 0.7x + 0.39, \tag{4.1}$$

$$y_2 = 0.58x + 0.14, \tag{4.2}$$

$$y_3 = 98.5x - 3. \tag{4.3}$$

Восстановленные уравнения прямых (для объемов 50-600 блоков):

$$y_1 = 0.72x + 0.68, \tag{4.4}$$

$$y_2 = 0.61x - 1.3, \quad (4.5)$$

$$y_3 = 100.3x - 37.9. \quad (4.6)$$

Как указано выше, коэффициент при x – пропускная способность алгоритма (микросекунд на блок), свободный член – время инициализации (микросекунд). Как видно, полученные значения пропускной способности достаточно стабильны на всех протестированных значениях. Время инициализации в действительности нулевое (поскольку алгоритм не потоковый, а блочный), поэтому оно значительно колеблется как по модулю, так и по знаку. Также, возможно, на него оказывает влияние планировщик Windows: с течением времени выделяет все больше и больше ресурсов на данную задачу.

Значения пропускной способности для реализаций 1 и 2 достаточно близки (0.7 и 0.6 микросекунд на блок соответственно). Они равны 23 Мб/с и 26 Мб/с, соответственно. Пропускная способность реализации 3 значительно меньше (100 микросекунд на блок, т. е. 0.16 Мб/с).

В качестве причин столь низкой производительности можно выделить отказ от чтения табличных данных (которые занимают оперативную память) и вместо этого вычисление этих данных. Второй причиной является «in-place» параметр, т.е. результат шифрования записывается по тому же указателю, из которого читаются входные данные. Это делает невозможной раскрутку циклов (которую выполняет оптимизатор), то есть очередная итерация не может начаться раньше, чем закончится предыдущая. Для тестирования алгоритмов, нацеленных на легковесную реализацию, такой режим тестирования гораздо более предпочтителен, так как уменьшает влияние процессорных оптимизаций, в том числе внутреннего параллелизма.

4.3. Выводы

В данной главе был описан порядок апробации предложенной методики сравннительного тестирования реализаций алгоритмлв путем сравнительного тестирования пропускной способности трех реализаций алгоритма AES на языке C. Проанализированы результаты тестирования.

ЗАКЛЮЧЕНИЕ

В начале работы ставились цели исследования легковесных криптографических алгоритмов с позиции таких характеристик, как требования к оперативной памяти устройства, энергопотребление, производительность.

В процессе изучения литературы по теме сформулированы основные (впрочем, уже классические) положения криптографии. Определены основные угрозы безопасности систем интернета вещей, они свои для каждого слоя таких систем. Для дальнейшего анализа выбираются те из них, которые направлены на сенсорный уровень. Основными из них являются подслушивание, введение фальшивого узла, атака повторного воспроизведения.

Вводится определение легковесного криптографического алгоритма, формулируются требования к нему.

Выделяются задачи, которые необходимо выполнить для противодействия указанным атакам. Это подтверждение достоверности и целостности пакета (аутентификация) – подтверждение, что пакет данных отправил законный отправитель, и пакет не был изменен по пути. Оно блокирует введение фальшивого узла отправителя или ретранслятора данных. Это подтверждение подлинности пакета (что это пакет, отправленный сейчас, а не отправленный ранее и перехваченный). Оно блокирует атаку повторного воспроизведения. Это, наконец, обеспечение конфиденциальности пакета данных – защита от подслушивания. При этом важно, чтобы процедура аутентификации была простой и однократной, так как интернет–соединение устройства может быть нестабильным и/или непостоянным.

Далее выделяются оптимальные подходы к решению данных задач. Для решения первой задачи наилучшим решением является применение цифровой подписи, которая, к тому же, гарантирует целостность сообщения. Подписывать рекомендуется не весь текст сообщения, а его хэш, это позволит ускорить процесс. Для реализации ЭЦП необходимо знание секретного ключа, который может быть, например, вшит при изготовлении устройства. Для подтверждения подлинности пакета наиболее разумным представляется добавление к сообщению метки времени. Так как сообщение будет хэшировано, ее не удастся изменить злоумышленнику. Необходимо наличие защищенного системного таймера.

Ключевой, главной задачей является третья – шифрование. Между блочными и потоковыми алгоритмами, более предпочтительными являются блочные алго-

ритмы, в особенности для устройств IoT (в силу небольших объемов передаваемых пакетов). Асимметричное шифрование подходит в наименьшей степени.

Далее описывается предлагаемая методология тестирования производительности и энергопотребления программной реализации легковесного криптоалгоритма. Главная задача такого тестирования – сравнения реализаций и алгоритмов. Описываются возможные препятствия и искажения такого тестирования, приводятся меры по их учету и преодолению.

Затем предложенная методология тестирования производительности применяется на практике. Сравниваются три различные реализации алгоритма AES. Результатом тестирования являются несколько серий измерений. Затем по ним вычисляются параметры алгоритмов.

Следует отметить, что расположение точек хорошо соответствует теоретическим прогнозам. Это свидетельствует главным образом об удачно построенном эксперименте, т.е. порядке вызова тестируемых функций. Полученные результаты являются достаточно стабильными, т.е. на всем исследованном промежутке (10-600 блоков) восстанавливается практически одинаковая производительность (разница не превышает 5%). Это также говорит об удачной методике, однако пока рано с уверенностью говорить о стабильности, для этого следует протестировать реализации бОльшего числа различных алгоритмов.

В процессе проведения работы возникло несколько «побочных» вопросов, которые могут лечь в основу дополнительных исследований. Насколько предсказуемо время выполнения команд процессором в каждом конкретном месте кода? Другими словами, как на время исполнения команды влияют соседние команды? Осталась непроверенной на практике описанная методика тестирования энергопотребления, в будущем её следует реализовать и проверить.

Таким образом, в результате работы достигнута поставленная цель – исследовать легковесные криптографические алгоритмы, а также выполнены задачи – определение перспективных видов легковесных криптоалгоритмов, формулирование рекомендаций по их применению. Создана методология тестирования производительности и энергопотребления алгоритмов, для сравнения алгоритмов. Тестирование производительности проверено, тестирование энергопотребления ожидает проверки в будущем.

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

ПК Персональный компьютер.

ЦП Центральный процессор.

CPU Центральный процессор.

ОЗУ Оперативное запоминающее устройство.

ОС Операционная система.

LW Легковесный.

LWC Lightweight cryptography (легковесная криптография).

ПДД Правила дорожного движения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ