ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

***«*САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»**

**ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК И ТЕХНОЛОГИЙ**

**ВЫСШАЯ ШКОЛА ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ И СУПЕРКОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ**

**Отчет о прохождении   
стационарной производственной преддипломной практики**

**на тему: «Сравнительный анализ легковесных криптографических алгоритмов для устройств интернета вещей»**

Момота Даниэля Михайловича, гр. 3530903/60301

**Направление подготовки:** 09.03.03 Прикладная информатика

|  |
| --- |
| **Место прохождения практики:** СПбПУ, ИКНТ, ВШИСиСТ |

*(указывается наименование профильной организации или наименование структурного подразделения*

|  |
| --- |
|  |

*ФГАОУ ВО «СПбПУ», фактический адрес)*

**Сроки практики:** с 21.04.2020 по 04.05.2020.

**Руководитель практики от ФГАОУ ВО «СПбПУ»:**

|  |
| --- |
| Сергеев Анатолий Васильевич, к.т.н., доцент ВШИСиСТ ИКНТ |

*(Ф.И.О., уч. степень, должность)*

*(Ф.И.О., должность)*

**Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Руководитель практики

от ФГАОУ ВО «СПбПУ» Сергеев А. В.

Обучающийся Момот Д. М.

Дата: \_\_\_\_\_\_\_\_\_\_

# Методология сравнения легковесных качеств криптографических алгоритмов

## Общие принципы

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 [[1]](#footnote-1) [[2]](#footnote-2) [[3]](#footnote-3) [[4]](#footnote-4). Для теоретической оценки производительности можно применять обычные методы оценки времени работы алгоритма. Они описаны, например, в [[5]](#footnote-5).

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.

## Оценка производительности на ПК

Производительность наиболее легко поддается оценке. 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.
* 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.
* 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.

Из вышеописанных ограничений точности измерения следуют рекомендации, как минимизировать влияние этих ограничений (т. е. систематическую погрешность).

* Минимизация использования «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 C с отключенной оптимизацией. Наихудшим вариантом является использование наиболее высокоуровневых языков вроде С++ или Java, так как порядок их трансляции в машинные команды весьма запутанный.
* Проблема различной скорости исполнения одной и той же инструкции процессорами разных архитектур не поддается простому решению. Можно пользоваться уже готовыми таблицами вроде такой[[6]](#footnote-6), однако кастомные реализации нуждаются в отдельном тестировании для каждой используемой команды.
* Для минимизации времени работы процессора над другими задачами следует по возможности отключить все другие приложения и службы перед началом тестирования. Для точного учета доли времени, выделяемого процессором на работу алгоритма, следует учесть количество других выполняемых задач, а также их приоритет и приоритет работы приложения, тестирующего алгоритм. Не следует забывать и о модуле для фиксирования времени работы тестируемого алгоритма. Этот модуль должен требовать минимальной загрузки ресурсов ЦП. Также необходимо оценить время переключения процессорного контекста. Разумеется, следует использовать для работы алгоритма только один поток, если не производится тестирование параллельной реализации

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[[7]](#footnote-7). 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.

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 . Коэффициент будет пропускной способностью, а – задержкой данного алгоритма. Подробнее данный классический метод описан, например, в пособии [[8]](#footnote-8).

## 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[[9]](#footnote-9). Единственный способ минимизировать влияние служб на энергопотребление – отключить их.

После отключения максимально возможного количества служб можно приступать к собственно оценке энергопотребления. По этому вопросу существует работа[[10]](#footnote-10). На ее основе предлагается следующий способ.

1. Вычисляется средний уровень энергопотребления ПК (ватт) в состоянии простоя. Он принимается за нуль.
2. Оценивается энергопотребление ПК на исполнение каждой из используемых в алгоритме инструкций: побитовый сдвиг, XOR и т. д. Каждый сеанс снятия этих данных следует проводить достаточно долго (хотя бы 30 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.
3. На основе измеренной величины энергопотребления, а также зная тактовую частоту процессора и число татов на каждую инструкцию, вычисляется энергия, необходимая на исполнение инструкции (микроватт\*микросек). Энергия на шифрование блока или байта вычисляется как сумма энергозатрат на каждую операцию. Если ранее была найдена пропускная способность (время шифрования одного блока или байта) данной реализации, то можно, разделив первую величину на вторую, найти энергопотребление (микроватт).
4. Полученное значение можно проверить путем непосредственного замера энергопотребления в единицу времени при исполнении алгоритма.

# Порядок сравнения избранных LWC-алгоритмов

В рамках практической части данной работы тестируется производительность избранных алгоритмов легковесной криптографии. Оно 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 будет 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.

# Инфраструктура тестирования и обработки результатов

Реализация избранных алгоритмов исполнена на языке C. Также используются некоторые модули Стандартной библиотеки языка Си (libc). Это модуль для измерения времени, модуль для вывода на экран и в файл. Код реализации алгоритмов приводится в Приложении 1. Код функций для фиксирования времени и логирования приводится в Приложении 2.

Порядок тестирования времени исполнения таков. Производится серий измерений времени шифрования. На каждой из серий производится измерений для каждого объема входных данных, . Итого для каждого алгоритма имеется точек, объединенных в групп по кластеров. Результаты тестирования записываются в JSON-файл.

Следующим этапом идет обработка результатов измерения. Она производится на языке средствами библиотеки . Сначала вычисляется значение, полученное по измерениям. Оно считается атомарным, и его погрешность по двум осям вычисляется как случайная погрешность величины, измеренной раз. Это позволяет снизить влияние «промахов» и других случайных факторов. Затем для каждой из серий измерений по атомарных точек строится прямая . Она строится с помощью метода наименьших квадратов. Фиксируются ее значения и и их погрешности. Таким образом фиксируются пар и . Наконец, вычисляется итоговые значения и . Это и будут пропускная способность и задержка алгоритма, соответственно. Код на *Python* приводится в Приложении 3.

# Результаты тестирования

Использовались следующие параметры.

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
|  | 10 |
|  | 100 |
|  | 20 |
|  |  |

Примеры построенных прямых для различных алгоритмов.

Итоговые результаты.

|  |  |  |
| --- | --- | --- |
| **Алгоритм** | **Пропускная способность** | **Задержка** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

В то же время, для данных алгоритмов существуют следующие результаты по времени работы.

|  |  |  |
| --- | --- | --- |
| **Алгоритм** | **Пропускная способность** | **Задержка** |
|  |  |  |
|  |  |
|  |  |
|  |  |  |
|  |  |  |
|  |  |  |

1. <http://ecsocman.hse.ru/data/2011/11/28/1270195383/2009_2_%D1%81.3-11_%D0%90%D0%B2%D0%B4%D0%BE%D1%88%D0%B8%D0%BD.pdf> [↑](#footnote-ref-1)
2. <https://cyberleninka.ru/article/n/otsenka-kriptograficheskoy-stoykosti-algoritmov-asimmetrichnogo-shifrovaniya> [↑](#footnote-ref-2)
3. <https://cyberleninka.ru/article/n/issledovanie-vozmozhnosti-primeneniya-geneticheskih-algoritmov-dlya-realizatsii-kriptoanaliza-blochnyh-kriptosistem> [↑](#footnote-ref-3)
4. <https://cyberleninka.ru/article/n/analiz-simmetrichnyh-kriptosistem> [↑](#footnote-ref-4)
5. <https://cyberleninka.ru/article/n/slozhnost-algoritmov-i-programm> [↑](#footnote-ref-5)
6. <https://www.agner.org/optimize/instruction_tables.pdf> [↑](#footnote-ref-6)
7. <https://smallbusiness.chron.com/turn-down-cpu-speed-bios-71566.html> [↑](#footnote-ref-7)
8. <http://npm.mipt.ru/books/lab-intro/main.pdf> [↑](#footnote-ref-8)
9. <https://cyberleninka.ru/article/n/otsenka-vliyaniya-apparatnyh-tehnologiy-energosberezheniya-personalnogo-kompyutera-na-proizvoditelnost-i-energopotreblenie> [↑](#footnote-ref-9)
10. <https://cyberleninka.ru/article/n/o-metode-otsenki-sootvetstviya-energozatratnosti-vychislitelnoy-zadachi-trebovaniyam-k-energopotrebleniyu-i-ego-realizatsii-s> [↑](#footnote-ref-10)