# Методология сравнения легковесных криптографических алгоритмов

## 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 [[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. 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 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.

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 вроде такой[[6]](#footnote-6), 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, которого затем умножить на долю процессорного времени, выделяемого тестируемому процессу. Либо же можно сразу измерять процессорное время, например, как описано в статье[[7]](#footnote-7).

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[[8]](#footnote-8). 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 займет время, равное не удвоенному времени исполнения ’а, а лишь чуть дольше одного ’а. 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 . Коэффициент будет пропускной способностью, а – задержкой данного алгоритма. Подробнее данный классический метод описан, например, в пособии [[9]](#footnote-9).

## 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[[10]](#footnote-10). Единственный способ минимизировать влияние служб на энергопотребление – отключить их.

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

Можно выделить два компонента энергопотребления ПК.

1. Энергопотребление компонентов . Оно является независимым от загрузки ПК и выполняемых задач. Под компонентами имеются в виду вентилятор, материнская плата, ОЗУ (согласно [[11]](#footnote-11)), видеокарта (если не производятся вычисления на ней и не рисуются сложные графические объекты), HDD.
2. Энергопотребление ЦП . Оно в разные моменты разное: при исполнении тестируемого процесса имеет величину (собственно, ее и следует измерить), при исполнении служебных (не относящихся к процессу тестирования) процессов – усредненную величину .

Существует два способа измерения энергопотребления процессора: измерить энергопотребление всего ПК (на уровне блока питания) и затем вычесть , или же напрямую измерить энергопотребление процессора [[12]](#footnote-12) [[13]](#footnote-13) (учитывается потребление на входе системной платы по разъему питания процессора [[14]](#footnote-14)), при этом следует вычесть энергопотребление регулятора напряжения. Второй подход предпочтителен.

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

Откуда получаем измеряемую величину

Величину можно найти как энергопотребление ЦП в состоянии простоя, разделенное на долю его загрузки в этом режиме (утилиты для определения загрузки приводятся здесь [[15]](#footnote-15)). Способ оценки величины описан в предыдущем разделе.

Далее описывается измерение энергопотребление для конкретного алгоритма. По этому вопросу существует работа[[16]](#footnote-16). На ее основе предлагается следующий способ.

Измеряется , … - 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 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.

Измеряется частота процессора . Это можно сделать, например, с помощью утилиты . По таблице (например, [[17]](#footnote-17)) определяется – число тактов на инструкцию (пропускная способность ЦП для данной инструкции). Откуда энергия, затрачиваемая на одну операцию, равна

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.

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

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.

# 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 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.

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://habr.com/ru/post/282301/> [↑](#footnote-ref-7)
8. <https://smallbusiness.chron.com/turn-down-cpu-speed-bios-71566.html> [↑](#footnote-ref-8)
9. <http://npm.mipt.ru/books/lab-intro/main.pdf> [↑](#footnote-ref-9)
10. <https://cyberleninka.ru/article/n/otsenka-vliyaniya-apparatnyh-tehnologiy-energosberezheniya-personalnogo-kompyutera-na-proizvoditelnost-i-energopotreblenie> [↑](#footnote-ref-10)
11. <https://qastack.ru/superuser/40113/does-installing-larger-ram-means-consuming-more-energy> [↑](#footnote-ref-11)
12. <https://www.overclockers.ua/cpu/cpu-power/> [↑](#footnote-ref-12)
13. <https://www.ixbt.com/cpu/cpu-power-measurement.shtml> [↑](#footnote-ref-13)
14. <http://al-tm.ru/stati/stati-po-blokam-pitaniya/bloki-pitaniya-standarta-eps12v> [↑](#footnote-ref-14)
15. <https://www.softsalad.ru/articles/best-programms/cpu-viewer-software> [↑](#footnote-ref-15)
16. <https://cyberleninka.ru/article/n/o-metode-otsenki-sootvetstviya-energozatratnosti-vychislitelnoy-zadachi-trebovaniyam-k-energopotrebleniyu-i-ego-realizatsii-s> [↑](#footnote-ref-16)
17. <https://www.agner.org/optimize/instruction_tables.pdf> [↑](#footnote-ref-17)