# Методология сравнения легковесных качеств криптографических алгоритмов

## Общие принципы

Необходимо различать сравнительный анализ алгоритмов и сравнительное тестирование их реализаций. Сравнительный анализ алгоритмов направлен на оценку и сравнение их стойкости в зависимости от длины ключа, числа раундов и других параметров (задача криптоанализа), а также на приблизительную, теоретическую оценку их производительности в зависимости от тех же параметров. В то время как сравнительное тестирование их реализаций направлено на сравнение таких характеристик, как требуемая площадь микросхемы (GE-мера), фактическая производительность (число байт в секунду, плюс задержка), а также энергопотребление (микроватт).

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 сравнения производительности и энергопотребления. О них далее.

## Сравнение производительности на ПК

Производительность наиболее легко поддается сравнению. Методика тестирования понятна: реализовать алгоритм, после чего запустить его 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. Существуют объективные ограничения точности такого способа тестирования. Фактически, они относятся к систематической (методологической) погрешности данного порядка 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, в том числе и стандартные ЦП для системных плат десктопных машин и ноутбуков могут показывать различную производительность при исполнении одной и той же инструкции. Тем более это касается «кастомных» процессорных реализаций.
* 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. Наихудшим вариантом является использование наиболее высокоуровневых языков вроде С++ или Java, так как порядок их трансляции в машинные команды еще более запутанный.
* Проблема различной скорости исполнения одной и той же инструкции процессорами разных архитектур не поддается простому решению. Можно пользоваться уже готовыми таблицами вроде такой[[1]](#footnote-1), 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 ОЗУ и ЦП. Также следует оценить время переключения процессорного контекста. Разумеется, следует использовать для работы алгоритма только один поток, если не производится тестирование параллельной реализации.

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.

## Сравнение энергопотребления на ПК

1. <https://www.agner.org/optimize/instruction_tables.pdf> [↑](#footnote-ref-1)