# 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 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 для измерения времени, модуль для точного измерения времени, а также для получения псевдослучайных чисел и для удобной работы с массивами переменной длины.

Тестированию подвергаются три реализации алгоритма AES на языке Си. Это две реализации, взятые из открытых источников ([[1]](#footnote-1) и [[2]](#footnote-2)), а также собственная реализация. Все три реализации были предварительно протестированы. Разработка собственной 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 С/С++. В этом случае необходимо использовать массив чисел – опкодов процессорных команд.

Порядок тестирования времени исполнения таков. Для каждого алгоритма производится серий измерений времени шифрования. На каждой из серий производится измерений для различных объемов входных данных, . Под входными данными понимается набор случайно сгенерированных блоков (количеством ), которые последовательно подвергаются шифрованию, ключ для каждого блока свой, он тоже генерируется случайно. Для уменьшения влияния погрешности, каждое измерение выполняется раз, время суммируется. Итого для каждого алгоритма имеется рядов по 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:

1. Есть специальный контрольный байт, значение которого обновляется при каждом шифровании (увеличивается на значение первого байта шифротекста). Он выводится в консоль.
2. На каждом из 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. Причина такого поведения не вполне ясна, возможно это связано с внутрипроцессорной оптимизацией. Этот эффект оказывает влияние на результаты тестирования, причем чем больше объем данных, тем более 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 . Затем для каждой из серий измерений по точкам 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.

# Результаты тестирования

Использовались следующие параметры.

|  |  |
| --- | --- |
| **Параметр** | **Значение** |
| n | 10 |
| t | 100 |
| m | 11 |
|  | 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. 1-3. 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. Можно сделать предположение, что это связано с особенностями оптимизатора языка С++.

Для сравнения, на рис. 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 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.

Также на рис. 1-4 приводятся усредненные экстраполированные прямые. Они демонстрируют неплохую устойчивость к выбросам, регулярно появляющимся при тестировании алгоритмов. Так, на рис. 1 и 2 первая экспериментальная точка является выбросом.

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

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

Как указано выше, коэффициент при х – 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.

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.

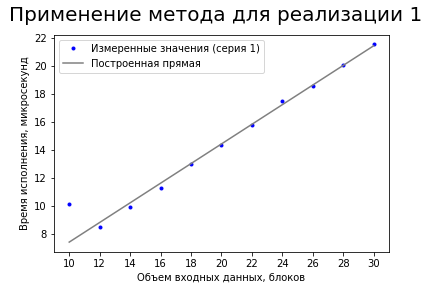


Рисунок 1

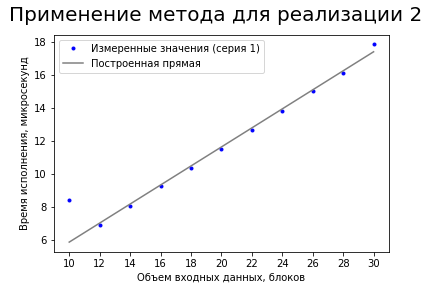


Рисунок 2

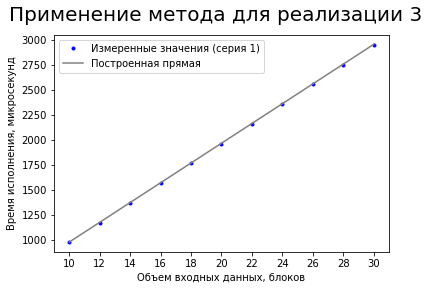


Рисунок 3

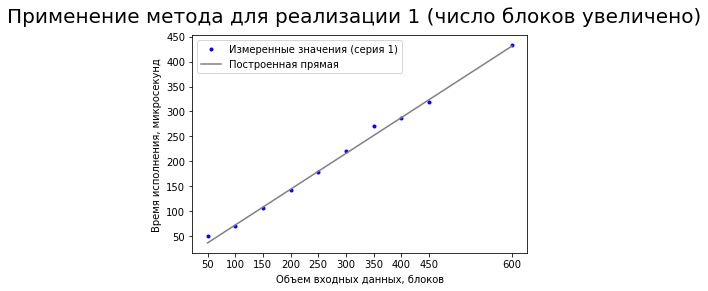


Рисунок 4

1. <https://github.com/chrishulbert/crypto/blob/master/c/c_aes.c> [↑](#footnote-ref-1)
2. <https://github.com/devershichandra27/C-implementation-of-AES> [↑](#footnote-ref-2)