

# РЕАЛИЗАЦИЯ АЛГОРИТМА ШИФРОВАНИЯ ГОСТ Р 34.12-2015 С ДЛИНОЙ БЛОКА 128 БИТ ДЛЯ ПРОЦЕССОРОВ ARM

Фахрутдинов Р.Ш., заведующий лабораторией безопасности  
информационных систем СПИИРАН, к.т.н., [fahr@cobra.ru](mailto:fahr@cobra.ru)

## Аннотация

Рассмотрен реализация алгоритма шифрования  
«Кузнечик» с длиной блока 128 бит с оптимизацией для  
процессоров ARM.

## Введение

19 июня 2015 года приказами Федерального агентства по техническому регулированию и метрологии № 749-ст и 750-ст были утверждены национальные стандарты ГОСТ Р 34.12–2015 и ГОСТ Р 34.13–2015. В стандартах [1] подробно описаны алгоритмы шифрования с длиной блока 128 бит «Кузнечик» и длиной блока 64 бита «Магма», определены режимы их работы, даны контрольные примеры для верификации разработанных на основе данных стандартов криптографических модулей. Мы рассмотрим особенности оптимизации данного алгоритма применительно к процессорам ARM.

## Алгоритм шифрования «Кузнечик»





Основой алгоритма является подстановочно-перестановочная сеть (SP-сеть), над входными данными в 10 раундов последовательно выполняются следующие преобразования :

1. Преобразование X :  $V_i = A_i \text{ XOR } K_i$  , где  $A_0$  — исходный текст, остальные  $A_i$  — данные на  $i$ -том сеансе,  $K_i$  - сеансовый ключ шифрования,  $V_i$  - результат преобразования;
2. Преобразование S : побайтная замена каждого байта  $V_i$  по его индексу из таблицы  $\pi$ , определённой в стандарте, результат заносится в  $C_i$ ;
3. Преобразование L : линейное преобразование, которое представляет собой регистр сдвига с обратной связью, реализованный над полем Галуа по модулю неприводимого многочлена степени 8. Для ускорения процедуры, большинство реализаций используют предвычисления для формирования готовой таблицы, в качестве индекса используется очередной обрабатываемый байт.

Входной ключ преобразовывается в 10 сеансовых за 32 раунда, для этого используются те же самые криптографические примитивы, что и при шифровании данных.

## Почему ARM?

ARM-процессоры занимают доминирующее положение в качестве MPU (Mobile Processor Unit) для планшетов, смартфонов, электронных книг, переносных игровых приставок и т. д. Кроме того, на их базе (наряду с архитектурой MIPS), выпускаются роутеры, промышленные платы для управления оборудованием и КИТ-наборы для разработки электронных устройств. Ряд отечественных компаний выпускает микропроцессоры на базе ядра ARM.

Название	Компания	Архитектура	Кэш	Частота
Baikal-M * 64-бит, 28нм		до 8 ядер Cortex-A57 (ARMv8-A), SIMD ARM- NEON	L2 512 KB/ядро L3 8 MB	до 2ГГц
СБИС К1879ХБ1Я , 32-бит, 65нм		ARM1176JZF- S, NeuroMatrix® NMC3**	L1: Кэш команд 16KB Кэш данных 16KB	324МГц +324МГц (сопроцессор)
СБИС 1879ВЯ1Я, 32-бит, 65нм		ARM1176JZF- S, 2х NeuroMatrix® NMC3**	L1: Кэш команд 16KB Кэш данных 16KB	От 164МГц до 328МГц + 2х 328МГц (сопроц.)
1892ВМ14 Я, 32-бит, 40 нм		2 ядра ARM Cortex-A9 (ARMv7), SIMD ARM- NEON, 2-х DSP-кластер “DELcore- 30M”	L1: Кэш команд 32 KB Кэш данных 32 KB L2 1 MB	От 744 до 1104МГц + 480- 912МГц (сопроц.)

\* Имеет аппаратный ускоритель, поддерживающий алгоритмы AES, SHA-2, ГОСТ 28147-89, ГОСТ Р 34.10-2012

\*\* 64-битный DSP-процессор

При грамотной реализации, процессор может обеспечить достаточную

скорость шифрования при высокой стойкости нового алгоритма. Программная реализация позволяет использовать алгоритм на большом количестве платформ (планшеты, телефоны, терминалы, оконечное оборудование и т. д.)

В качестве альтернативы были рассмотрены ПЛИС (Altera, Zinq), которые могут обеспечить очень высокое быстродействие при умеренном энергопотреблении.

## **Преимущества архитектуры ARM**

При реализации алгоритмов шифрования важными особенностями современных процессоров ARM являются:

- Одновременное выполнение нескольких инструкций (при отсутствии взаимозависимых данных);
- Большое число 32-битных регистров;
- Загрузка результата операции в регистр, не содержащий данных для обработки;
- Возможность загрузки большого количества данных одной операцией (операции групповой загрузки-выгрузки данных);
- Внутри команды можно дополнительно менять операнд перед обработкой;
- Наличие SIMD-расширения (ARM NEON), позволяющего выполнять инструкции над длинными данными в отдельном конвейере.

Большинство возможностей в той или иной мере реализуют современные компиляторы.

## **Рассмотренные реализации**

Основной задачей, которая ставилась при исследовании, являлась разработка оптимизации данного алгоритма под процессоры ARM. Были исследованы следующие реализации :

1. Базовая реализация от ОАО «ИнфоТеКС» (доступна на сайте TK26) [2]
2. Arseny <https://github.com/MaXaMaR/kuznezhikev> [arseny.aprelev@gmail.com](mailto:arseny.aprelev@gmail.com) [3]
3. Markku-Juhani O. Saarinen [mjos@iki.fi](mailto:mjos@iki.fi) [4]
4. Max Tishkov [maxamar@mail.ru](mailto:maxamar@mail.ru) [5]

Наиболее подходящей для дальнейшей работы по оптимизации, нам показалась версия [4]. В ней все преобразования сведены в таблицу

размером 65Кб, по которой функция шифрования выполняет операции XOR. Для расшифрования используются ещё 2 таблицы по 65Кб.

## Результаты тестирования

Исходный текст [4] был доработан для использования типов данных, отличных от 128-бит (в оригинальной версии для этого было необходим процессор с набором инструкций SSE4.1 и соответствующий компилятор). Была сделана версия, в которой типы данных для обработки можно было выбрать (8, 16, 32, 64 и 128-битные).

Для тестирования на процессорах ARM, с помощью кросскомпилятора было собрано тестовое приложение на базе версии [4].

Первично мы сделали версию, которая использует только инструкции ARM-процессора, а затем некоторые места оптимизировали с помощью SIMD NEON. Оптимизация заключалась в замене наиболее узких мест реализации [4] на реализацию с помощью инструкций сопроцессора NEON, которым оснащено большинство современных процессоров ARM.

	Процессор	Версия [2]	Версия [4]	Версия [4] + SIMD NEON
ARM Cortex-A9 Amlogic 8726-MX 800MHz	Кэш L1 Данные 32Кб Код 32Кб Кэш L2 512Кб 2 ядра 40нм	361Кб/с	4.5Мб/с	7.8Мб/с
То же, 1.2ГГц		548Кб/с	6.8Мб/с	11.8Мб/с
То же, 1.5ГГц		693Кб/с	8.6Мб/с	14.8Мб/с
ARM Cortex-A9 RockChip RK3188 1.8ГГц	Кэш L1 Данные 32Кб Код 32Кб Кэш L2 512Кб 4 ядра 28нм		9.6Мб/с	18.5Мб/с
ARMv7 Qualcomm MSM 8974 до 2.3ГГц	Кэш L1 Данные 32Кб Код 32Кб Кэш L2 2Мб 28нм			33Мб/с

## Пример оптимизации

В версии [4] процедура шифрования выглядит следующим образом :

```
void kuz_encrypt_block(kuz_key_t *key, void *blk)
{
    int i;
    __m128i x;

    x = *((__m128i *) blk);

    for (i = 0; i < 9; i++) {
        x ^= *((__m128i *) &key->k[i]);
        x = kuz_pil_enc128[ 0][((uint8_t *) &x)[ 0]] ^
            kuz_pil_enc128[ 1][((uint8_t *) &x)[ 1]] ^
            kuz_pil_enc128[ 2][((uint8_t *) &x)[ 2]] ^
            kuz_pil_enc128[ 3][((uint8_t *) &x)[ 3]] ^
            kuz_pil_enc128[ 4][((uint8_t *) &x)[ 4]] ^
            kuz_pil_enc128[ 5][((uint8_t *) &x)[ 5]] ^
            kuz_pil_enc128[ 6][((uint8_t *) &x)[ 6]] ^
            kuz_pil_enc128[ 7][((uint8_t *) &x)[ 7]] ^
            kuz_pil_enc128[ 8][((uint8_t *) &x)[ 8]] ^
            kuz_pil_enc128[ 9][((uint8_t *) &x)[ 9]] ^
            kuz_pil_enc128[10][((uint8_t *) &x)[10]] ^
            kuz_pil_enc128[11][((uint8_t *) &x)[11]] ^
            kuz_pil_enc128[12][((uint8_t *) &x)[12]] ^
            kuz_pil_enc128[13][((uint8_t *) &x)[13]] ^
            kuz_pil_enc128[14][((uint8_t *) &x)[14]] ^
            kuz_pil_enc128[15][((uint8_t *) &x)[15]];
    }
    x ^= *((__m128i *) &key->k[9]);

    *((__m128i *) blk) = x;
}
```

Как мы видим, основную часть процедуры составляет длинная операция XOR (исключающее ИЛИ) над шестнадцатью 128-битными словами, которые находятся в таблице предвычислений (kuz\_pil\_enc128). Размер таблицы составляет 64Кб, таким образом она не помещается в кэш данных процессоров ARM, что существенно замедляет шифрование.

Для оптимизации операции используем групповые операции загрузки в регистры данных и их последующую обработку с помощью eor (XOR):

```
ldmia    r10, {r2-r5} // load key data into R2-R5
eor       r2, r2, r0 // R2-R5 = R2-R5 ^ R0:R1:R7:R8
eor       r3, r3, r1
eor       r4, r4, r7
eor       r5, r5, r8
```

При обработке данных необходимо по возможности позже использовать любой вычисленный операнд, т. к. в большинстве современных процессоров следующая команда может начинаться выполняться, если она не использует регистров, формируемых предыдущими командами. Обратите внимание на использование R2 — R5 в командах выше.

Так как сопроцессор ARM NEON работает независимо от центрального процессора, передача данных из его регистров в регистры ЦП выполняется относительно медленно, а загрузка данных из регистров ЦП в регистры сопроцессора — быстро. Для эффективной работы программы с использованием сопроцессора, мы должны перемежать код, написанный для ЦП с кодом для сопроцессора. Тогда разница в скорости загрузки данных из сопроцессора в ЦП минимизируется.

Пример перемежения кода (выделенным шрифтом — команды сопроцессора):

```
and      r6, r2, #0xFF // R6 = _X.B[0]
add      r6, R14, r6, LSL #4

VLDmia  R6, {q0} /* D0:D1 is data */

and      R6, R2, #0xFF00
add      R6, R14, R6, LSL #4
add      R6, R6, #4096 * #1

VLDmia  R6, {q1}

and      R7, R2, #0xFF00
add      R7, R14, R7, LSR #4
add      R7, R7, #4096 * #1

VEOR    q0, q0, q1
VLDmia  R7, {q2}
```

Большую помощь в оптимизации с использованием инструкций ARM NEON оказал документ [6].

## Заключение

Рассмотрены результаты работы по оптимизации алгоритма шифрования ГОСТ Р 34.12-2015 с длиной блока 128 бит для процессоров ARM.

## Литература

1. Национальный стандарт Российской Федерации ГОСТ 34.12-2015  
[https://tc26.ru/standard/gost/GOST\\_R\\_3412-2015.pdf](https://tc26.ru/standard/gost/GOST_R_3412-2015.pdf)
2. Программная реализация ТК26 от ОАО «Инфотекс»  
[http://tc26.ru/standard/gost/PR\\_GOSTR\\_bch\\_v9.zip](http://tc26.ru/standard/gost/PR_GOSTR_bch_v9.zip)
3. Программная реализация от Arseny Aprelev  
<https://github.com/aprelev/libgost15>
4. Программная реализация от Markku-Juhani O. Saarinen  
<https://github.com/mjosaarinen/kuznechik>
5. Программная реализация от Max Tishkov  
<https://github.com/MaXaMaR/kuznezhik>
6. SIMD Assembly Tutorial: ARM NEON  
[https://people.xiph.org/~tterribе/daala/neon\\_tutorial.pdf](https://people.xiph.org/~tterribе/daala/neon_tutorial.pdf)