УДК 519.719.2

С. В. Дорохин[1,2], *С. С. Качков*[1,3], *А. А. Сидоренко*[1,3]

[1]Московский физико-технический институт
[2]АО "ПКК Миландр"
[3]АО "Интел"

Реализация блочного шифра «Кузнечик» с использованием векторных инструкций

Целью данной работы является создание оптимизированной программной реализации блочного шифра ГОСТ Р 34.12 2015, известного как «Кузнечик». В ходе исследования был проведён анализ возможных средств улучшения скорости работы шифра. Основное внимание уделено использованию SIMD (Single Instruction Multiple Data) инструкций и учёту общедоступной информации об устройстве Execution Engine процессоров Intel®Core™ поколения Sandy Bridge и новее. Отличительной особенностью статьи является то, что в ней представлены измерения скорости зашифрования и расшифрования в режимах ECB, CBC, CFB, OFB на процессорах четырёх различных поколений (включая Kaby Lake R), в открытом доступе выложен исходный код высокоскоростной реализации и скриптового интерфейса для автоматического тестирования. Для дальнейшего ускорения используется набор команд AVX2. Предлагается использование 256-битных регистров для ускорения зашифрования и расшифрования в режиме ECB, расшифрования в режиме CFB.

Ключевые слова: ГОСТ Р 34.12 2015, высокоскоростная реализация, LSX-преобразование, блочный шифр, SSE, AVX.

S. V. Dorokhin[1,2], *S. S. Kachkov*[1,3], *A. A. Sidorenko*[1,3]

[1]Moscow Institute of Physics and Technology
[2]JSC "Milandr"
[3]JSC "Intel"

Implementation of Kuznyechik cipher using vector instructions

This article is concentrated on highly-optimized implementation of block cipher GOST R 34.12 2015, also known as Kuznyechik. A comparative analysis of possible imporvements is presented, with the SIMD (Single Instruction Multiple Data) instructions being in focus. The publicly available information about Intel®Core™ Execution Engine (starting with Sandy Bridge) was taken into consideration. AVX2 instruction set gets our special attention. The key feature of the article is that a cutting-edge open-source implementation is presented. The abovementioned implementation allows to compare 4 modes of operation, such as ECB, CBC, CFB and OFB. The results in this paper are given for 4 modern generations of Intel®Core™ line (with Kaby Lake R being the newest one). We also suggest using 256-bit ymm registers and AVX2 instruction set to boost ECB mode encryption & decryption and CFB mode decryption.

Key words: GOST R 34.12 2015, high-speed implementation, LSX-transform, block cipher, SSE, AVX.

1. Введение

«Кузнечик» представляет собой блочный шифр с размером блока 128 бит. Обозначим множество всевозможных двоичных строк длинны s как V_s. Алгоритм зашифрования сводится к последовательному применению следующих операций:

- Побитовая операция сложения по модулю 2: $X[k](a) = k \oplus a$, где $k, a \in V_{128}$;
- Биективное нелинейное преобразование: $S(a) = S(a_{15}||...||a_0) = \pi(a_{15})||...||\pi(a_0)$, где $a_i \in V_8$, символом $||$ обозначена операция конкатенации, а $\pi : V_8 \to V_8$ – некоторая известная подстановка;
- Линейное преобразование: $L : V_{128} \to V_{128}$.

С учётом этих обозначений функции зашифрования $E[k]$ и расшифрования $D[k]$ могут быть представлены в следующем виде:

$$E_{k_1,...,k_{10}}(a) = X[k_{10}]LSX[k_9]...LSX[k_2]LSX[k_1](a);$$
$$D_{k_1,...,k_{10}}(a) = X[k_1]S^{-1}L^{-1}X[k_2]...S^{-1}L^{-1}X[k_9]S^{-1}L^{-1}X[k_{10}](a),$$

где k_1, ... k_{10} – раундовые ключи. Эти ключи вырабатываются один раз в начале алгоритма, вычисляются также с помощью X, S, и L преобразований и не оказывают существенного влияния на скорость работы шифра, поэтому в рамках статьи алгоритм развёртывания ключа не обсуждается. Здесь и далее раундовые ключи предполагаются уже вычисленными. Линейное преобразование L может быть осуществленно с помощью 16 циклов работы РСЛОС (регистра сдвига с линейной обратной связью), показанного на рис. 1.

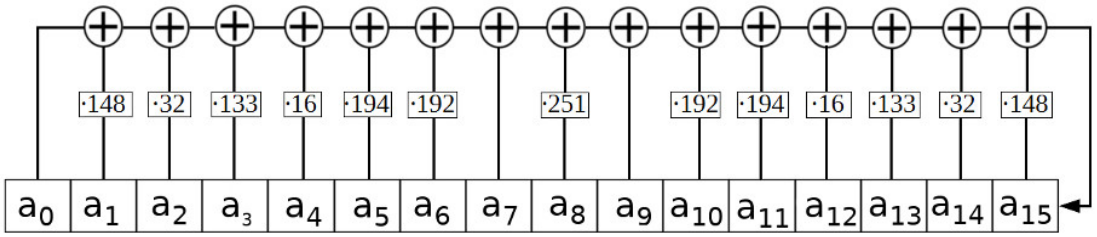

Рис. 1. РСЛОС, реализующий L-преобразование

Такое решение является предпочтительным при аппаратной реализации шифра, однако при создании программной реализации удобнее представить L преобразование в матричной форме. Результат работы одного такта РСЛОС можно записать в виде

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{pmatrix}$$

Соответсвенно, результат работы после k тактов:
формулка
Применение L-преобразования в такой форме существенно быстрее, чем при моделировании РСЛОЛ. Однако профилирование программы при помощи утилиты callgrind показало, что на L преобразование приходится приблизительно 75% времени исполнения программы. Следуя принципу make common case fast, необходимо оптимизировать именно L-преобразование, чему, по существу, и посвящена эта статья.

2. Построение таблиц предвычислений

Для ускорения работы объединённого LS-преобразования используются заранее вычисленные таблицы (LUT, Lookup Table) - одна для прямого преобразования (шифрования), другая для обатного (расшифрования). На первом этапе строится матрица преобразования, соответствующая одному шагу работы сдвигового регистра, которая затем возводится в 4 степень (для получения 16 тактов работы регистра) - результатом является матрица L-преобразования. LUT имеет размер 16x256, каждый элемент которого является блоком данных из 16 байт, и строится по следующему принципу: LUT[i][j] есть результат покомпонентого умножения S[j] на i-ый столбец матрицы L-преобразования. Таким образом, для осуществления LS-преобразования над блоком необходимо произвести сложения по модулю 2 всех 16 блоков из LUT:

Листинг 1.1. XSL-преобразование с использованием LUT

```
void Grasshopper::ApplyXSL(Block& data, const Block& key)
{
    ApplyX(data, key);
    Block tmp{};
    for (size_t i = 0; i < block_size; i++)
        ApplyX(tmp, enc_ls_table[i][data[i]]);
    data = tmp;
}
```

3. Реализация с использованием векторных инструкций

3.1. Набор инструкций SSE

SSE добавить описание команд, посчитать суммарный CPI и latency, исполльзуя интеловский онлайн-справочник

3.2. Учёт особенностей планировщика

Дальнейшая оптимизация LS-преобразования может быть произведена с учётом возможностей суперскалярной архитектуры современных процессоров. Например, микроархитектура Intel Sandy Bridge имеет в своём распоряжении 2 исполнительных устройства для вычисления адреса (AGU - address generation unit), а также 3 ALU (arithmetic and logic unit), способные производить операции над векторными регистрами (ссылка на Intel 64 and IA-32 Architectures Optimization Reference Manual). Чтобы помочь планировщику выполнять загрузки блока из таблицы и суммирование по модулю 2 с результатом параллельно, можно использовать чередование регистров в этих командах:

Листинг 1.2. LS-преобразование с использованием чередования регистров

```
__m128i vec1 = _mm_load_si128(reinterpret_cast<const __m128i*>(table+
    _mm_extract_epi16(tmp2, 0)+0x0000));
__m128i vec2 = _mm_load_si128(reinterpret_cast<const __m128i*>(table+
    _mm_extract_epi16(tmp1, 0)+0x1000));

vec1 = _mm_xor_si128(vec1, CastBlock(table+_mm_extract_epi16(tmp2, 1)+0x2000));
vec2 = _mm_xor_si128(vec2, CastBlock(table+_mm_extract_epi16(tmp1, 1)+0x3000));

vec1 = _mm_xor_si128(vec1, CastBlock(table+_mm_extract_epi16(tmp2, 2)+0x4000));
vec2 = _mm_xor_si128(vec2, CastBlock(table+_mm_extract_epi16(tmp1, 2)+0x5000));
...
vec1 = _mm_xor_si128(vec1, CastBlock(table+_mm_extract_epi16(tmp2, 7)+0xE000));
vec2 = _mm_xor_si128(vec2, CastBlock(table+_mm_extract_epi16(tmp1, 7)+0xF000));
data = _mm_xor_si128(vec1, vec2);
```

3.3. Набор инструкций AVX

описать неудачную попытку использовать ymm регистры внутри ApplyXSL

4. Использование AVX2 в режимах ECB и CFB

В некоторых режимах шифрования можно обрабатывать сразу два блока данных, используя расширенные до 256 бит векторные регистры (расширение набора инструкций AVX2, поддерживаемое с Intel Haswell и AMD Excavator). Это возможно в тех случаях, когда шифрование (или расшифрование) текущего блока возможно производить независимо от предыдущего. В этой работе были реализованы следующие режимы работы шифра:

- ECB (Electronic Codebook)
- CBC (Cipher Block Chaining)
- CFB (Cipher Feedback)
- OFB (Output Feedback)

Данная оптимизация была осуществлена для шифрования и расшифрования в режиме ECB и для расшифрования в режиме CFB.

4.1. Модификация режима ECB

4.2. Модификация расшифрования в режиме CFB

5. Сравнительный анализ

Зависит от конкретного процессора! Зависит от архитектуры! (skylake и далее - лучше, РАЗОБРАТЬСЯ, ПОЧЕМУ) Посчитать cpb (clocks per byte)

6. Заключение

Текст заключения.

Литература

1. *Алексеев Е. К., Попов В. О., Прохоров А. С., Смышляев С.., Сонина Л. А.* Об эксплуатационных качествах одного перспективного блочного шифра типа LSX // Математические вопросы криптографии. — 2015. — Т. 6, вып. 2. — С. 6–17.
2. *Бородин М. А., Рыбкин А. С.* Высокоскоростные программные реализации блочного шифра Кузнечик // Проблемы информационной безопасности. Компьютерные системы. — 2014. — Вып. 3. — С. 67–73.
3. *Рыбкин А. С.* О программной реализации алгоритма Кузнечик на процессорах Intel // Математические вопросы криптографии. — 2018. — Т. 9, вып. 2. — С. 117–127.

References

1. *Alekseev E. K., Popov V. O., Prokhorov A. S., Smyshlyaev S. V., Sonina L. A.* On the performance of one perspective LSX-based block cipher // Mathematical Aspects of Cryptography. — 2015. — V. 6, N. 2. — P. 6–17.
2. *Borodin M. A., Rybkin A. S.* High-Speed Software Implementation of Kuznyetchik block cipher. — Information Security Problems. Computer Systems — 2014. — N. 3 —P. 67-73.

3. *Rybkin A. S.* On software implementation of Kuznyechik on Intel CPUs // Mathematical Aspects of Cryptography. — 2018. — V. 9, N. 1. — P. 117–127.

Поступила в редакцию дд.мм.гггг.

Сведения об авторах статей
(на момент подачи статьи)

Реализация блочного шифра «Кузнечик» с использованием векторных инструкций

Дорохин Семён Владимирович (нет, студент 4 курса, Московский физико-технический институт, студент 4 курса) dorohin.sv@phystech.edu
Качков Сергей Сергеевич (нет, студент 4 курса, Московский физико-технический институт, студент 4 курса) kachkov.ss@phystech.edu
Сидоренко Антон Андреевич (нет, студент 4 курса, Московский физико-технический институт, студент 4 курса) sidorenko.aa@phystech.edu

Ссылки на опубликованные статьи
(в соответствии с ГОСТ Р 7.0.5-2008)

Дорохин С.В., Качков С.С., Сидоренко А.А. Реализация блочного шифра «Кузнечик» с использованием векторных инструкций // Труды МФТИ. — 2018. — Т. 10, № 4. — С. 1–**??**.

Dorokhin S.V., Kachkov S.S., Sidorenko A.A. Implementation of Kuznyechik cipher using vector instructions // Proceedings of MIPT. — 2018. — V. 10, N 4. — P. 1–**??**.