5em 0pt

УДК Dorokhin.tex

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 шифра ГОСТ Р 34.12 2015 с использованием SIMD инструкций

В данной работе обсуждаются возможности увеличения скорости работы блочного шифра ГОСТ Р 34.12 2015, известного как "Кузнечик". В отличие от шифра AES, "Кузнечик"не имеет аппартной поддержки на процессорах Intel и AMD, поэтому вопрос о максимально возможной скорости за- и расшифрования стоит особенно остро. В первой части работы кратко описано устройство шифра и рассмотрены известные на текущий момент методы ускорения, связанные с использованием таблиц предвычислений. Далее описываются опубликованные в свободном доступе варианты реализации, основанные на наборах векторных инструкций SSE и AVX. В следующей части мы представляем наши модификации упомянутых реализаций с использованием набора инструкций AVX2 для некоторых режимов работы шифра. В заключение приводится сравнительная характеристика скорости работы шифра в различных реализациях.

Ключевые слова: SIMD инструкции, Кузнечик, SSE, AVX, ГОСТ Р 34.12 2015, программная реализация.

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 GOST R 34.12 2015 cipher using SIMD instructions

This article is dedicated to optimized implementations of GOST R 34.12 2015 block cipher. Compared with AES cipher, GOST cipher is not supported on hardware level by any known processors. That is why the task of exploring its maximum encryption/decryption speed is a crucial one. In the first section the cipher's algorithm and a brief description of relatively trivial LUT (lookup tables) optimizations are given. The next paragraph consists of a consise overview of currently available implementations which use vector command sets SSE and AVX. Finally, our AVX-based modifications are presented followed by a comparative analysis of open-source implementations.

Key words: SIMD instructions, SSE, AVX, GOST R 34.12 2015, block cipher, software implementation.

1. Введение

Для полноты изложения приведём краткое описание алгоритма, более подробную информацию можно найти в документации на сайте технического коммитета по стандартизации [1]. Пусть Vs – множество всевозможных двоичных строк длинны s. Алгоритм

зашифрования сводится к послеовательному применению следующих операций:
X[k](a) = k xor a, где K, a прин. V128 – побитовая операция сложиения по модулю 2;
S(a) = S(a15||...||a0) = pi(a15)||...||pi(a0), ai прин V8 – биективное нелинейное преобразование;
L: V128 -> V128 – некоторое линейное преобразование.
С учётом этих обозначений функции зашифрования E[k] и расшифрования D[k] могут быть представлены в следующем виде:
EK1,...,K10(a) = X[k10]*LSX[K9]...LSX[K2]LSX[K1](a);
DK1,...,K10(a) = X[K1]S-1L-1X[K2]...S-1L-1X[K9]S-1L-1X[K10](a),
где K1, ... K10 – раундовые ключи. Эти ключи вырабатываются один раз в начале алгоритма и не оказывают существенного влияния на скорость работы шифра. Процедуру генерации этих ключей можно также найти на сайте комитета по стандартизации [1]. Линейное преобразование L может быть осуществленно с помощью 16 циклов работы РСЛОС (регистра сдвига с линейной обратной связью), показанного на рисунке 1.
рисунок 1, регистр сдвига
Такое решение является предпочтительным при аппаратной реализации шифра, однако при создании программной реализации удобнее представить его в матричной форме. Результат работы одного такта РСЛОС можно записать в виде
формулка
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% проверить! 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. *Шевченко Д. В., Шевченко В. П.* Выбор и оптимизация структуры построения автономных сейсмических средств обнаружения рубежного типа // Материалы VIII всероссийской научно-технической конференции «Современные охранные технологии и средства обеспечения комплексной безопасности объектов». — 2010. — С. 128–133.
2. *Diallo M. S., Kulesh M., Holschneider M., Sherbaum F., Adler F.* Characterization of polarization attributes of seismic waves using continuous wavelet transforms // Geophysics. — 2006. — V. 71, N. 3. — P. 67–77.
3. *Лайонс Р.* Цифровая обработка сигналов. — М.: Бином, 2006. — С. 361–369.

References

1. *Shevchenko D. V., Shevchenko V. P.* Selection and optimization of constructing autonomous seismic detection struction a landmark type // Proceedings of the VIII Russian scientific conference “Modern security technology and means of complex security objectives”. — 2010. — P. 128–133. — (in Russian).
2. *Diallo M. S., Kulesh M., Holschneider M., Sherbaum F., Adler F.* Characterization of polarization attributes of seismic waves using continuous wavelet transforms // Geophysics. — 2006. — V. 71, N. 3. — P. 67–77.
3. *Lyons R.* Digital signal processing. — M.: Binom, 2006. — P. 361–369. — (in Russian).

Поступила в редакцию дд.мм.гггг.

Сведения об авторах статей
(на момент подачи статьи)

Реализация шифра ГОСТ Р 34.12 2015 с использованием SIMD 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 шифра ГОСТ Р 34.12 2015 с использованием SIMD инструкций // Труды МФТИ. — 2018. — Т. 10, № 4. — С. 2–5.

Dorokhin S.V., Kachkov S.S., Sidorenko A.A. Implementation of GOST R 34.12 2015 cipher using SIMD instructions // Proceedings of MIPT. — 2018. — V. 10, N 4. — P. 2–5.