АЛГОРИТМЫ ХЭШИРОВАНИЯ BASH: БЫСТРОДЕЙСТВИЕ

С. В. Агиевич, В. И. Семенов

НИИ прикладных проблем математики и информатики
Белорусский государственный университет
Минск, Беларусь
e-mail: agievich@bsu.by, semenov.vlad.by@gmail.com

Обсуждается быстродействие алгоритмов хэширования Bash, которые вводятся в новом стандарте СТБ 34.101.77. Предлагаются приемы эффективной реализации алгоритмов. Проводится сравнение быстродействия алгоритмов Bash и Keccak (SHA-3).

Ключевые слова: алгоритм хэширования; шаговая функция хэширования; логические операции; циклический сдвиг; инструкции AVX2.

THE BASH HASHING ALGORITHMS: PERFOMANCE

S. Agievich, V. Semenov

Research Institute for Applied Problems of Mathematics and Informatics
Belarusian State University
Minsk, Belarus

We discuss performance of the Bash hashing algorithms which are introduced in the forthcoming standard STB 34.101.77. We propose techniques for efficient implementation of the Bash algorithms and compare them with the Keccak (SHA-3) algorithms.

Keywords: hashing algorithm; step hash function; bitwise logic operations; cyclic shift; AVX2 instructions.

АЛГОРИТМЫ Bash

Bash — это семейство алгоритмов хэширования, стандартизация которых завершается в Республике Беларусь. Определяющий алгоритмы стандарт СТБ 34.101.77 вводится в действие в октябре 2016 г.

Алгоритмы Bash различаются уровнями стойкости $l \in \{16, 32, 48, \cdots, 256\}$. Алгоритм уровня l обрабатывает данные блоками из $1536 - 4l$ битов и возвращает хэш-значение длины $2l$. Очередной блок данных записывается в начало хэш-состояния $S \in \{0,1\}^{1536}$, после чего состояние преобразуется с помощью биективной шаговой функции Bash-f. Первые $2l$ битов окончательного, после обработки всех блоков данных, состояния составляют итоговое хэш-значение.

Все операции в Bash-f выполняются над машинными словами длины 64. Состояние S разбивается на 24 таких слова: $S = S_0 \| S_1 \| \cdots \| S_{23}$. Слова записываются в матрицу 3×8 последовательно слева направо и сверху вниз. Тройки (S_v, S_{v+8}, S_{v+16}),

$v = 0, 1, \cdots, 7$, называются вертикальными плоскостями, а восьмерки $(S_{8i}, S_{8i+1}, \cdots, S_{8i+7})$, $i = 0, 1, 2$, — горизонтальными.

Действие Bash-f состоит в 24-кратном применении тактовых подстановок, которые представляют собой композиции описываемых ниже преобразований (подробнее см. в [1]).

Линейное перемешивание. К тройкам слов вертикальных плоскостей применяются преобразования $L3$, которые реализуются операциями $\oplus$ и $RotHi^d$: $\oplus$ – поразрядное сложение по модулю 2, $RotHi^d$ – циклический сдвиг на d позиций в сторону старших разрядов. Всего сдвиги в $L3$ выполняются 4 раза. Набор величин сдвигов $[m_1, m_2, n_1, n_2]$ уточняет действие $L3$. Вычисление $(W_0, W_1, W_2) = L3[m_1, m_2, n_1, n_2](w_0, w_1, w_2)$ проводится следующим образом:

$$W_0 \leftarrow w_0 \oplus w_1 \oplus w_2,$$

$$W_1 \leftarrow w_1 \oplus RotHi^{m_1}(w_0) \oplus RotHi^{n_1}(W_0),$$

$$W_2 \leftarrow w_2 \oplus RotHi^{m_2}(w_2) \oplus RotHi^{n_2}(w_1 \oplus RotHi^{n_1}(W_0)).$$

Можно обойтись 6 сложениями и 4 сдвигами, определив аргумент $RotHi^{n_2}$ при вычислении W_1.

Преобразование усложнения S3 реализуется операциями $\neg$ (поразрядное отрицание), $\wedge$ (поразрядное И), $\vee$ (поразрядное ИЛИ) и $\oplus$. Операции перечислены в порядке убывания приоритета. Тройка слов (W_0, W_1, W_2) каждой вертикальной плоскости преобразуется следующим образом:

$$(W_0, W_1, W_2) \leftarrow (W_0 \oplus W_1 \vee \neg W_2, W_1 \oplus W_0 \vee W_2, W_2 \oplus W_0 \wedge W_1).$$

Перестановка P описывает изменение порядка слов состояния: на место S_u записывается слово $S_{P(u)}$. Выбранная структура P задает циклический сдвиг горизонтальных плоскостей вверх с одновременными перестановками их слов:

$$P(u) = \begin{cases} \pi_0(u) + 8, & 0 \le u < 8, \\ \pi_1(u-8) + 16, & 8 \le u < 16, \\ \pi_2(u-16), & 16 \le u < 24. \end{cases}$$

Здесь π_i — перестановка слов i-й горизонтальной плоскости:

$$\pi_0(v) = (v + 2(v \bmod 2) + 7) \bmod 8,$$

$$\pi_1(v) = v + 1 - 2(v \bmod 2),$$

$$\pi_2(v) = (5v + 6) \bmod 8.$$

Сложение с тактовыми константами. На каждом такте к слову S_{23} добавляется тактовая константа $C \in \{0, 1\}^{64}$. Константы не повторяются, что делает тактовые преобразования разными (неоднородными).

КОЛИЧЕСТВО ОПЕРАЦИЙ

Алгоритмы Bash построены по той же схеме, что и стандартизированные недавно в США алгоритмы Keccak (SHA-3). Длина состояния Keccak составляет 1600 битов, данные обрабатываются блоками из $1600 - 4l$ битов. Аналогом Bash-f является шаговая функция Keccak-f[1600]. Эта функция также является композицией 24 так-

товых подстановок, и эти подстановки также построены с помощью операций $\oplus$, $RotHi^d$, $\neg$, $\wedge$ над 64-разрядными словами.

В табл. 1 указано количество операций, выполняемых на одном такте `Bash-f` и `Keccak-f[1600]`. Таблицу не следует трактовать так, что алгоритмы `Bash` безусловно опережают по быстродействию алгоритмы `Keccak`. Во-первых, в `Keccak` данные обрабатываются несколько бóльшими блоками . Во-вторых, для `Keccak-f[1600]` известны способы сокращения числа операций: через оптимизацию схемы вычислений или за счет дополнительных инструкций процессора. В-третьих, на быстродействие влияют неучтенные в таблице операции пересылки машинных слов, их загрузки в регистр процессора и выгрузки из регистра, организация конвейера команд, другие особенности целевых аппаратных архитектур.

Таблица 1

Трудоемкость одного такта шаговых функций

Операция	`Bash-f`	`Keccak-f[1600]`
$\oplus$	73	76
$\vee$	16	0
$\wedge$	8	25
$\neg$	8	25
$RotHi^d$	32	29
Всего	137	155

Далее мы обсудим тонкости программно-аппаратной реализации алгоритмов `Bash`.

ОПЕРАЦИЯ `ANDNOT`

На многих аппаратных платформах доступна инструкция `andnot`, которая реализует составную операцию $u \wedge \neg v$.

В `Keccak-f[1600]` операция $\neg$ идет в связке с $\wedge$, и такая связка тривиально меняется на $\wedge\neg$. При этом общее число операций на одном такте сокращается на 25, до 131.

В `Bash-f` операция $\neg$ используется только в преобразовании $S3$ и только в связке с $\vee$. Инструкция `ornot`, которая реализует составную операцию $u \vee \neg v$, является редкой, и прямое сокращение числа операций не всегда возможно. Однако $\vee\neg$ можно реализовать через $\wedge\neg$, если инвертировать определенные прообразы и образы $S3$. В результате число операций можно сократить на 8, до 129. Покажем, как это сделать.

На первом такте вместо $S3$ выполним преобразование

$$S3_0\colon (\overline{W}_0, W_1, W_2) \leftarrow (W_0 \oplus W_2 \wedge \neg W_1, W_1 \oplus W_0 \vee W_2, W_2 \oplus W_0 \wedge W_1).$$

Черта над словом означает, что оно вычислено в инвертированном виде: вместо слова W_0, которое было бы получено применением $S3$, вычислено слово $\neg W_0$. После перестановки P и линейного преобразования $L3$ инверсия W_0 приведет к инверсии всех прообразов $S3$ на втором такте. На этом такте вместо $S3$ выполним преобразование

$$S3_1\colon (W_0, W_1, W_2) \leftarrow (\overline{W}_0 \oplus \overline{W}_1 \wedge \neg\overline{W}_2, \overline{W}_1 \oplus \overline{W}_0 \wedge \overline{W}_2, \overline{W}_2 \oplus \overline{W}_0 \vee \overline{W}_1).$$

На выходе $S3_1$ мы возвращаемся к обычным, без инвертирования, словам. Поэтому чередование «$S3_0$ на нечетных тактах, $S3_1$ на четных тактах» можно продолжить и дальше.

ПЕРЕСТАНОВКА *P*

Если слова состояния располагаются в отдельных регистрах процессора или в отдельных ячейках памяти, то преобразование P можно реализовать перенумерацией регистров или ячеек. Но при использовании SIMD-инструкций, оперирующих сразу несколькими словами, может потребоваться явная перестановка слов внутри длинного регистра или в смежных ячейках памяти. Опишем способ снижения затрат на реализацию P.

Напомним, что P задает сдвиг горизонтальных плоскостей состояния вверх с одновременными перестановками их слов. Упростим P, заменяя, по-возможности, перестановки слов в горизонтальных плоскостях на перенумерацию вертикальных плоскостей и учитывая перенумерацию на следующем такте. При упрощении необходимо сохранять расположение слов в каждой из вертикальных плоскостей — к словам плоскостей применяются преобразования $L3$ и $S3$. Изберем следующий принцип упрощения: расположение слов в средней горизонтальной плоскости (которая становится верхней) сохраняется.

На первом такте вместо P используется перестановка

$$P_0: \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 \end{pmatrix} \mapsto \begin{pmatrix} 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 22 & 19 & 16 & 21 & 18 & 23 & 20 & 17 \\ 1 & 0 & 3 & 2 & 5 & 4 & 7 & 6 \end{pmatrix},$$

а на втором такте — перестановка

$$P_1: \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 \end{pmatrix} \mapsto \begin{pmatrix} 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 18 & 23 & 20 & 17 & 22 & 19 & 16 & 21 \\ 5 & 4 & 7 & 6 & 1 & 0 & 3 & 2 \end{pmatrix}.$$

Композиция двух экземпляров P совпадает с композицией P_0 и P_1. Поэтому на 3-м и следующих нечетных тактах вместо P снова можно использовать перестановку P_0, а на четных тактах — перестановку P_1.

СДВИГИ НА 32-РАЗРЯДНЫХ ПЛАТФОРМАХ

При переносе `Bash-f` на 32-разрядную аппаратную платформу циклический сдвиг 64-разрядного слова становится громоздким — его приходится реализовывать через большое число операций с короткими словами.

Сдвиг можно упростить, если использовать прием «чередование битов» (bit interleaving), предложенный в [5]. 64-разрядное слово W загружается в два 32-х разрядных регистра следующим образом: биты с четными номерами загружаются в регистр R_0, а биты с нечетными номерами – в регистр R_1 (биты W нумеруются от младших к старшим, начиная с 0). При таком представлении W циклический сдвиг $RotHi^{2d}(W)$ соответствует циклическим сдвигам R_0 и R_1 на d позиций. А циклический сдвиг $RotHi^{2d+1}(W)$ соответствует сдвигу R_0 на $d+1$ позицию, сдвигу R_1 на d позиций и последующему обмену содержимым регистров.

Чередование битов можно поддерживать на протяжении нескольких вызовов `Bash-f`. Требуется только корректно загружать слова входных данных и выгружать выходные слова состояния.

ЭКСПЕРИМЕНТЫ

Эталонная реализация алгоритмов `Bash` написана на языке Си. Реализация представлена в основной ветке криптографической библиотеки Bee2 [2]. В реализации циклы и тактовые константы развернуты, используется 3 слова дополнительной памяти.

Анализ ассемблерного кода реализации, полученного с помощью компилятора MSVC, выявил частые обращения к оперативной памяти на тактах `Bash-f`, что, видимо, связано с переиспользованием (повторной инициализацией) временных переменных. Добавление в реализацию преобразования $S3$ новых временных переменных позволило компилятору эффективнее использовать регистры процессора, сократило время работы на 30 % и количество обращений к памяти более чем вдвое, с 2944 до 1304.

Реализация `Bash` с использованием инструкций AVX2 представлена в экспериментальной ветке Bee2 [3]. Структура `Bash-f` позволяет загрузить все состояние в 6 регистров AVX2 и выполнять операции одновременно в 4 вертикальных плоскостях состояния. Помимо 6 регистров, представляющих состояние, дополнительно используется не менее 8 регистров для хранения промежуточных результатов вычислений. Реализация, полученная с помощью компилятора MSVC, использует 16 регистров.

При переходе на архитектуру AVX2 быстрые циклические сдвиги и перестановка P усложняются. Циклические сдвиги реализованы через инструкции правых и левых покомпонентных сдвигов с последующим сложением. Для реализации P используется прием, описанный выше. AVX2 содержит инструкции для перемешивания и выборки слов внутри 256-разрядного регистра, что позволяет эффективно реализовать перестановки P_0 и P_1. Заметим, что левая и правая половины третьей горизонтальной плоскости перестановок P_0 и P_1 перемешиваются независимо, что упрощает реализацию. Кроме того, в AVX2 есть инструкция `andnot`, поэтому можно отказаться от явного инвертирования, задействовав операцию $\wedge\neg$ по описанной выше схеме.

Производительность двух реализаций `Bash` была измерена в системе eBACS [4], результаты замеров приведены в табл. 2.

Таблица 2

Производительность (тактов процессора на байт данных)

Реализация хэш-функции	Длина хэш-значения		
	256	384	512
Конфигурация: amd64; 2012 Intel Xeon E3-1275 V2			
Bash (эталонная)	11,79	15,47	23,43
Keccak (эталонная)	10,19	13,29	19,20
Конфигурация: amd64; 2013 Intel Xeon E3-1275 V3			
Bash (AVX2)	6,55	8,60	13,05
Keccak (AVX2)	8,93	11,69	16,56

Отметим, что новая архитектура AVX-512 добавляет еще больше возможностей для оптимизации: можно использовать 512-разрядные регистры, тернарную логику (инструкции, которые поразрядно применяют произвольную булеву функцию к тройке регистров), инструкции перемешивания слов в регистре.

БИБЛИОГРАФИЧЕСКИЕ ССЫЛКИ

1. Agievich S., Marchuk V., Maslau A., Semenov V. Bash-f: another LRX sponge function // Pre-proceedings of the 5th Workshop on Current Trends in Cryptology. CTCrypt-2016, Yaroslavl, Russia, June 6–8, 2016, Jaroslanl, 2016. P. 184–205. URL : http://eprint.iacr.org/2016/587.
2. Bee2: A cryptographic library (the master branch). URL: https://github.org/agievich/-bee2 (last access 30.07.2016.).
3. Bee2: A cryptographic library (the bash-avx2 branch). URL: https://github.org/bcrypto/-bee2/tree/bash-avx2 (last access 30.07.2016.).
4. eBACS: ECRYPT Benchmarking of Cryptographic Systems. ed.: D. Bernstein, T. Lange URL: https://bench.cr.yp.to/ (last access 30.07.2016).
5. Bertoni G., Daemen J., Peeters M., Van Assche G., Van Keer R. Keccak implementation overview. Version 3.2, 2012. URL: http://keccak.noekeon.org/.