ОБОЗРЕНИЕ
ПРИКЛАДНОЙ И ПРОМЫШЛЕННОЙ
Том 24 МАТЕМАТИКИ Выпуск 3
2017

Ф. А. Дали, В. О. Миронкин (Москва, ТК 26, МИЭМ НИУ ВШЭ).
О некоторых моделях древовидного хэширования.

1. Введение Графическое представление, использующее связные ориентированные графы [2] является одним из наиболее наглядных способов описания режимов работы хэш-функций, допускающих распараллеливание вычислений. Представленные подобным образом режимы называются древовидными режимами, а соответствующие графы — деревьями хэширования.

Пользуясь терминологией теории графов [1, 2], введем ряд обозначений и определений. Так вершины дерева хэширования будем называть узлами. Узлы, не имеющие предков, будем называть листьями, а единственный узел, не имеющий потомков — корнем. Пусть при этом

- $\mathcal{P}$ — множество узлов дерева хэширования;
- $\mathcal{L}$ — множество листьев дерева хэширования;
- $\mathcal{I}$ — множество узлов дерева хэширования, не являющихся листьями;
- M_i — i-й слой дерева хэширования, $i \in \overline{1, k+1}$;
- $M_{i,j}$, — j-й узел i-го слоя, $j \in \overline{1, r_i}$.

О п р е д е л е н и е 1. Дерево хэширования называется l-арным, если каждый его узел из $\mathcal{I}$ имеет не более l предков.

О п р е д е л е н и е 2. Высотой k дерева хэширования называется максимальное расстояние от листьев к корню.

О п р е д е л е н и е 3. i-м слоем дерева хэширования называется множество из r_i узлов, лежащих на расстоянии $k - i + 1$ от корня.

Функция $h : V_m \to V_t$, $m > t$, где m — размер узлов, используемая для преобразования $\mathcal{I}$, называется внутренней функцией.

По способу функционирования древовидные режимы могут быть разделены на два класса:

- режимы последовательного преобразования слоев дерева хэширования;
- режимы параллельного преобразования слоев дерева хэширования.

Деревья хэширования из первого класса (MD6 [6], Skein [5] и Blake2 [4]) формируются за счет последовательного преобразования узлов первого слоя, состоящего из блоков исходного сообщения. Для второго класса деревья хэширования формируются до начала процесса вычисления хэш-кода на основе имеющего объема памяти. Для формирования таких деревьев блоки сообщения поступают сразу на все имеющиеся узлы и параллельно преобразуются [7].

Алгоритм 1 (последовательное преобразование слоев)

1. Исходное сообщение $M \in V_n$, $n \in \mathbb{N}$, разбивается последовательно на блоки длины m: $M = M_1||M_2||\dots||M_{\lceil \frac{n}{m} \rceil}$.

2. На вход узлов M_j, $j \in \overline{1, \min(T, \lceil \frac{n}{m} \rceil)}$, подаются блоки сообщения M_j. Полагаем $i = 1$.

3. Если $\lceil \frac{n}{m} \rceil > T$, то переходим на шаг 5, в противном случае переходим на шаг 7.

4. Если $\lceil \frac{n}{m-t} \rceil > T$, то переходим на шаг 6, в противном случае переходим на шаг 7.

Процедура впитывания

5. 1-й слой дерева хэширования формируется на основе вектора длины mT, взятого от исходного сообщения M. Полагаем $i = i + 1$, $n = n - mT$ и переходим на шаг 4.

6. 1-й слой дерева хэширования формируется как конкатенация вектора длины tT — результата преобразования i-го слоя на основе функции h, а именно конкатенации значений функции h от каждого узла, и вектора длины $(m-t)\,T$, взятого от оставшейся части сообщения M. Полагаем $i = i + 1$, $n = n - mT - (i-1)\,(m-t)\,T$ и переходим на шаг 4.

Процедура построения дерева

7. Пусть $k \in \mathbb{N}$ максимальное: $n_i = ml^k + q_i$, $0 \leqslant q_i < m\left(l^{k+1} - l^k\right)$. Тогда

- либо дополняем при необходимости i-й слой до длины кратной m вектором специального вида и формируем $(i+1)$-й слой на основе результата преобразования i-го слоя с помощью функции h.)
- либо формируем $(i+1)$-й слой на основе конкатенации вектора длины q_i, перенесенного с i-ого слоя, и результата преобразования i-го слоя (без учета вектора длины q_i) с помощью функции h.

8. Если длина слоя $n_i > m$, то полагаем $i = i + 1$, и переходим на шаг 7, в противном случае при необходимости дополняем i-й слой до длины m вектором специального вида и полагаем $H(M) = h(M_i)$.

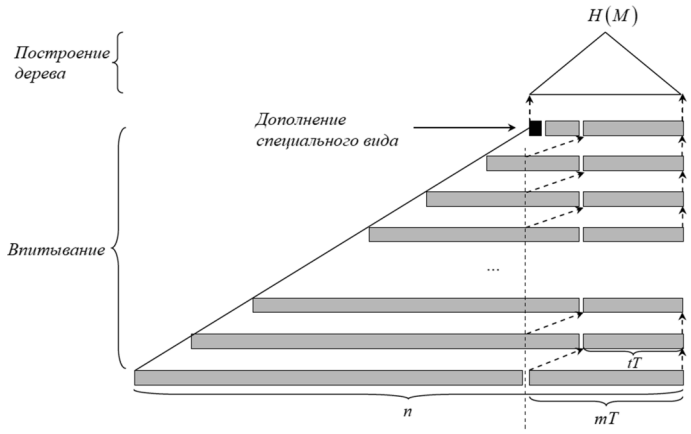

Рис. 1. Последовательное преобразование слоев

Схема работы алгоритма 1 представлена на рис. 1. Вектор, используемый для дополнения слоев на шаге 7 алгоритма 1, должен быть фиксированным и независимым от сообщения. Например, в MD6 используется нулевой вектор.

Однако, в ряде случаев дополнение нулевыми битами позволяет строить структурные коллизии, например, за счет дополнения сообщения нулевыми битами. Во избежание подобных ситуаций последний блок сообщения M следует принудительно дополнять, например, вектором вида $(1,0,\ldots,0)$.

Следует также отметить, что дополнение на последнем слое дерева хэширования в некоторых случаях дает возможность определения второго прообраза хэш-кода $H(M)$ с трудоемкостью меньшей трудоемкости полного перебора. При этом вероятность успеха существенно зависит от распределения на множестве образов внутренней функции h.

Формирование эквивалентных поддеревьев (деревьев, содержащих одинаковые значения в своих корнях) при последовательном формировании слоев также позволяют находить второй прообраз хэш-кода. Во избежание этого следует исключать совпадения узлов в слоях дерева хэширования [3].

Как было замечено выше второй класс рассматриваемых режимов существенно зависит от объема доступной памяти. Как правило, для корректного вычисления хэш-кода его должно хватать для хранения значений всех узлов полного дерева хэширования. Будем полагать, что каждый узел дерева ассоциирован с ячейкой памяти (или парой накопителей u_i, z_i), тогда имеет место следующий алгоритм.

Алгоритм 2 (параллельное преобразование слоев)

1. В начальный момент времени все накопители u_i, z_i, $i \in \mathcal{P}$, содержат пустые строки. Пусть $j = 0, u = 0$.

2. Для $i \in \mathcal{P}$, начиная с M_0 послойно в накопители u_i записываются блоки длины m исходного сообщения M.

3. Для $i \in \mathcal{P}$ в накопители z_i узлов P_i записывается значения $z_i = h(u_i)$.

4. Если $n > m\left(\frac{l^{k+1}-1}{l-1}+1\right)$, то переходим на шаг 6.

5. Если $n \leqslant m$, то дополняем сообщение вектором специального вида до длины m и полагаем $H(M) = z_0$, и алгоритм заканчивает свою работу, в противном случае вычисляем d:

$$\frac{l^d-1}{l-1} \leqslant \left\lceil \frac{n-m}{m} \right\rceil \leqslant \frac{l^{d+1}-1}{l-1},$$

и переходим на шаг 10.

Процедура впитывания

6. $j = j+1$. В накопитель u_0 записывается $(m-2t)$ бит оставшейся части сообщения M, затем для $i \in \mathcal{I}$, $i > 0$, в накопители u_i — $(m-tl)$ бит, и для $i \in \mathcal{L}$ в накопители u_i — m бит.

7. В накопитель z_0 записывается значение $z_0 = h(u_0||z_0||z_1)$. Для $i \in \mathcal{I}$, $i > 0$, в накопители z_i записываются значения $z_i = h\left(u_i||z_{il+1}||...||z_{(i+1)l}\right)$.

8. Если $n - j\left(m - 2t + \frac{l^k-1}{l-1}(m-tl) + ml^k\right) > 0$, то переходим на шаг 6.

9. Если $n - j\left(m - 2t + \frac{l^k-1}{l-1}(m-tl)\right) - (j-1)ml^k > 0$, тогда полагаем $d = k$, в противном случае полагаем $d = k-1$, и переходим на шаг 10.

Процедура усечения дерева

10. Если $u < d$, то в накопитель z_0 записывается значение $z_0 = h(u_0||z_0||z_1)$, в накопители z_i, $i > 0$, записываются значения $z_i = h\left(u_i||z_{il+1}||...||z_{(i+1)l}\right)$, полагаем $u = u+1$ и переходим на шаг 10, в противном случае полагаем $H(M) = z_0$, и алгоритм заканчивает свою работу.

На рис. 2 и 3 схематично представлены основные этапы работы алгоритма 2.

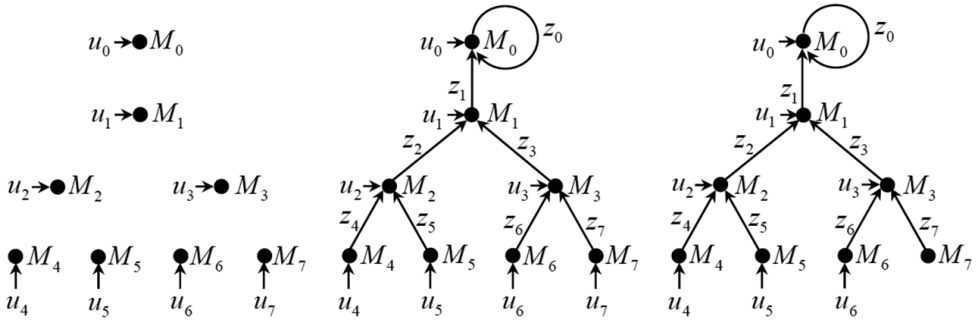

Рис. 2. Процедура впитывания

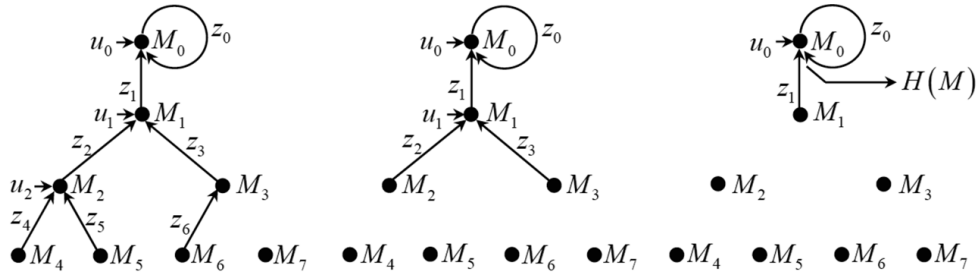

Рис. 3. Процедура усечения дерева

Из рис. 2 видно, что процедура усечения дерева хэширования начинается в случае, когда на узлы перестают поступать блоки сообщения M. При этом после каждого очередного шага алгоритма 2 узлы дерева хэширования последовательно от листьев к корню перестают вносить изменения в узлы верхних слоев (рис. 3). В итоге единственный оставшийся узел будет содержать значение хэш-кода.

Теорема. *Трудоемкость алгоритма* 1 *не превосходит трудоемкости алгоритма* 2.

Таким образом, с точки зрения эффективности для заданного объема памяти алгоритм 1 является более предпочтительным.

СПИСОК ЛИТЕРАТУРЫ

1. *Колчин В. Ф.* Случайные графы (2-е изд.). М.: ФИЗМАТЛИТ, 2004.
2. *Сачков В. Н.* Вероятностные методы в комбинаторном анализе. М.: Наука, 1978.
3. *Дали Ф. А.*, *Миронкин В. О.* О вероятностных характеристиках одного класса моделей древовидного хэширования. — Обозрение прикл. и промышл. матем. 2016, т. 23, в. 4, с. 345–347.
4. *Aumasson J.-P.*, *Neves S.*, *Wilcox-O'Hearn Z.*, *Winnerlein C.* BLAKE2: simpler, smaller, fast as MD5, 2013, https://blake2.net.
5. *Ferguson N.*, *Lucks S.*, *Schneier B.*, *Whiting D.*, *Bellare M.*, *Kohno T.*, *Callas J.* The Skein Hash Function Family, Version 1.3., October, 2010.
6. *Rivest R. L.*, *Agre B.*, *Bailey D. V.*, *Crutchfield C.*, *Dodis Y.*, *Fleming K. E.*, *Khan A.*, *Krishnamurthy J.*, *Lin Y.*, *Reyzin L.*, *Shen E.*, *Sukha J.*, *Sutherland D.*, *Tromer E.*, *Yin Y. L.* The MD6 hash function. A proposal to NIST for SHA-3, http://csrc.nist.gov — Massachusetts Institute of Technology, Cambridge, MA, October, 2008.
7. *Sarkar P.*, *Schellenberg P. J.* A parallelizable design principle for cryptographic hash functions, Cryptology ePrint Archive, Report 2002/031, 2002, http://eprint.iacr.org.