

# Алгоритм хэширования Кессак и лавинный эффект.

Батарин Егор

## Аннотация

Статья посвящена обзору самому передовому на данный день алгоритму хеширования - Кессак. Рассмотрены история его создания, определение хэш-функции, основные свойства криптографических хэш-функций, общая структура алгоритма Кессак, реализация функции перестановок и описание проекта по исследованию лавинного эффекта на Кессак.

## 1 История создания алгоритма

В 2012 году сообщество NIST провело конкурс алгоритмов SHA-3, после того, как были проведены успешные атаки на предыдущие алгоритмы семейства SHA. На конкурс были приняты 51 алгоритмов, в результате естественного отбора которых в качестве нового стандарта SHA был принят алгоритм Кессак.

## 2 Что такое хэш-функция?

Хэш-функция - это отображение, которое ставит в соответствие строке произвольной длины строку фиксированной длины. Как правило, мощность отображаемого множества больше мощности множества значений хэш-функции, а значит она не может быть инъективной. Пара сообщений, нарушающих инъективность функции, называется коллизией.

Сферы применения хэш-функций различны:

- Построение уникального идентификатора для данного набора данных
- Для вычисления контрольных сумм и обнаружения ошибок
- При сохранении пароля в виде хэш-кода
- При выработке электронной подписи на хэш-код

Последние два применения взяты из криптографии. Не любые хэш-функции годятся для применения в этих случаях. Для этого нужно использовать специальные хэш-функции, называемые криптографическими, которые обеспечивают повышенную безопасность системы.

## 3 Свойства криптографических хэш-функций

Чтобы хэш-функция была годна для защиты информации, она должна обладать следующими свойствами:

-Сопrotивление поиску первого прообраза: для данного хэш-кода трудно найти исходное сообщение, которое было отображено данной хэш-функцией в данный хэш-код. Иными словами, для данной хэш-функции трудно построить функцию, обратную к ней.

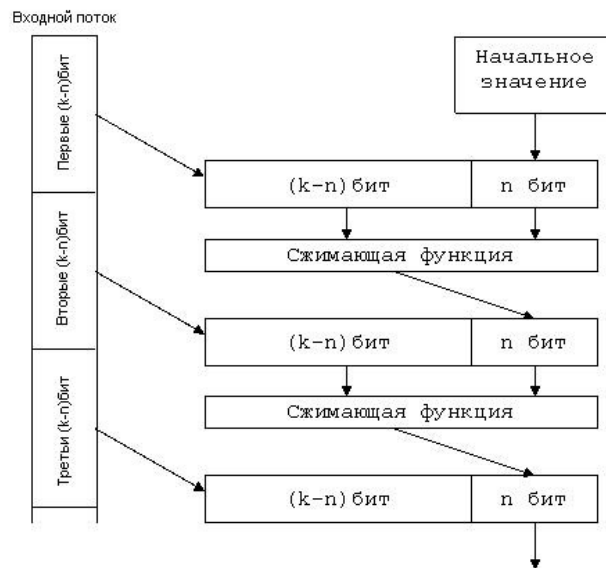
-Сопrotивление поиску второго прообраза: для данного сообщения трудно найти другое сообщение, отображаемое хэш-функцией в тот же хэш-код.

-Стойкость к коллизиям: нет эффективного полиномиального алгоритма, который находит коллизии.

-Лавинный эффект: малое изменение входных значений хэш-функции сильно меняет выходное значение.

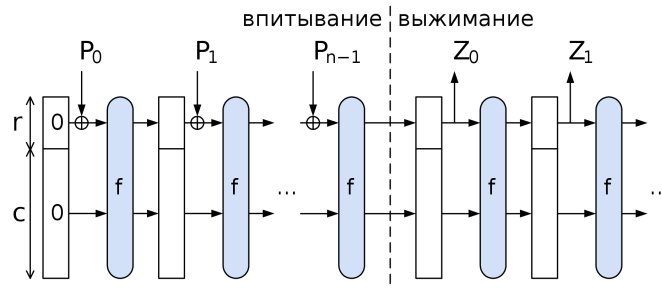
Как правило, криптографические хэш-функции строятся по итеративной последовательной схеме.

Такая схема позволяет достичь лавинного эффекта, поскольку каждый бит выходного потока зависит от всего входного потока данных. В основе распространенной схемы лежит разделение входного сообщения на множество блоков, каждый из которых обрабатывается сжимающей функцией. Затем полученное значение используется для последующих итераций. Для того, чтобы длина входного сообщения была кратна длине блока, его удлиняют дополнительными битами до нужной длины. Похожие идеи используются в алгоритме Кессак. Такая конструкция называется губкой.



## 4 Губка

Вычисление хэш-кода в конструкциях по типу губки происходит в несколько стадий.



- 1) Исходное сообщение  $M$  дополняется до строки  $P$  так, чтобы длина последней была кратна  $r$ . Для этого используется функция дополнения.
- 2) Строка  $P$  делится на строки  $P_0, \dots, P_{n-1}$ .
- 3) Происходит этап "впитывания":
  - 3.1) Каждый блок  $P_i$  дополняется нулями до строки длины  $b$  бит - верхние входы над вертикальными стрелочками на рисунке
  - 3.2) Генерируется начальная строка состояния  $S$  длины  $b = r + c$ , заполненная нулями - самый левый белый прямоугольник на рисунке.
  - 3.3) Она суммируется по модулю 2 вместе со строкой  $P_0$ , а результат обрабатывается функцией перестановок  $f$ .
  - 3.4) Полученная в результате действия  $f$  строка длины  $b$  суммируется по модулю 2 с  $P_1$  и все последующие операции повторяются итеративно.
- 4) Происходит этап "отжимания": на этом этапе суммирование по модулю 2 прекращается и к строке состояния применяется функция перестановок. К конечному результату  $Z$  каждый раз добавляются первые  $r$  бит строки состояния. Происходит это до того момента, пока длина  $Z$  будет не меньше  $d$ . Когда это достигнуто, лишняя часть обрезается и в конце концов получается строка  $Z$  длины  $d$ .

## 5 Функция перестановок $f$

### 5.1 Общая структура

Как мы видели выше, строка состояния  $S$  претерпевает изменения под действием функции перестановок  $f$  в результате ”впитывания”и ”отжимания”. Авторы представляют эту строку как массив  $5 \times 5 \times 64$ , стало быть  $A[i][j][k]$  - это  $(5i + j) \times 64 + k$  бит строки  $S$ . В общей реализации SHA-3 вместо 64 стоит  $w = 2^l$ , при этом вычисления, проходящие во время определения результата функции перестановок, проходит в несколько раундов, число которых равно  $12 + 2l$  алгоритма Кессак положено  $l = 6$ , так что для него число раундов равно 24. В каждый из этих раундов проходит вычисление функций  $\theta, \rho, \pi, \chi, \iota$ . Опишем подробно сущности этих пяти шагов и договоримся на каждом шаге обозначать входной массив за  $A$ , а выходной за  $A'$ .  $\text{mod}$  означает сложение по модулю 2.  $i_r$  - номер раунда.

### 5.2 Шаг $\theta$

Для всех  $i$  и  $k$ , таких, что  $0 \leq i < 5, 0 \leq k < w$ , положим

$$C(i, k) = A[i, 0, k] \oplus A[i, 1, k] \oplus A[i, 2, k] \oplus A[i, 3, k] \oplus A[i, 4, k]$$

$$D(i, k) = C[(i - 1) \bmod 5, k] \oplus C[(i + 1) \bmod 5, (k - 1) \bmod w]$$

Для всех  $(i, j, k)$ , таких, что  $0 \leq i < 5, 0 \leq j < 5, 0 \leq k < w$ ,

$$A'[i, j, k] = A[i, j, k] \oplus D[i, k]$$

### 5.3 Шаг $\rho$

Для всех  $k$ , таких, что  $0 \leq k < w, A'[0, 0, k] = A[0, 0, k]$

Пусть в начале  $(i, j) = (1, 0)$ .

Для  $t$  от 0 до 23 выполнять:

1. Для всех  $k$ , таких, что  $0 \leq k < w, A'[i, j, k] = A[i, j, (k - (t + 1)(t + 2)/2) \bmod w]$
2.  $(i, j) = (j, (2i + 3j) \bmod 5)$

### 5.4 Шаг $\pi$

Для всех  $(i, j, k)$ , таких, что  $0 \leq i < 5, 0 \leq j < 5, 0 \leq k < w$

$$A'[i, j, k] = A[(i + 3j) \bmod 5, i, k]$$

### 5.5 Шаг $\chi$

Для всех  $(i, j, k)$ , таких, что  $0 \leq i < 5, 0 \leq j < 5$ ,

$$A'[i, j, k] = A[i, j, k] \oplus ((A[(i + 1) \bmod 5, j, k] \oplus 1) \cdot A[(i + 2) \bmod 5, j, k])$$

### 5.6 Шаг $\iota(A, i_r)$

Введем дополнительную функцию  $rc(t)$ . Она вычисляет значения следующим образом:

1. Если  $t \bmod 255 = 0$ , то возвращается 1
2. Пусть  $R = [100000000]$
3. Для  $i$  от 1 до  $t \bmod 255$  выполнять:
  - 3.1)  $R = 0 \parallel R$
  - 3.2)  $R[0] = R[0] \oplus R[8]$
  - 3.3)  $R[4] = R[4] \oplus R[8]$
  - 3.4)  $R[5] = R[5] \oplus R[8]$
  - 3.5)  $R[6] = R[6] \oplus R[8]$
  - 3.6)  $R = \text{Trunc}_8[R]$
- 4) Возвращается  $R[0]$

Теперь описываем сам алгоритм  $\iota(A, i_r)$ :

1. Для всех  $(i, j, k)$ , таких, что  $0 \leq i < 5, 0 \leq j < 5, 0 \leq k < w$   $A'[i, j, k] = A[i, j, k]$
2. Положим  $RC$  — массив длины  $w$ , заполненный нулями.
3. Для  $i$  от 0 до  $l$ :  $RC[2^i - 1] = rc(i + 7i_r)$
4. Для всех  $k$ , таких, что  $0 \leq k < w$ ,  $A'[0, 0, k] = A'[0, 0, k] \oplus RC[k]$

## 5.7 Алгоритм перестановок

Теперь можно писать структуру всего алгоритма перестановок:

1. Переводим строку  $S$  в массив  $A$
2. Для  $i_r$  от  $12 + 2l - n_r$  до  $12 + 2l - 1$  применяем  $A' = \iota(\chi(\pi(\rho(\theta(A))))), i_r)$
3. Переводим конечный массив  $A'$  в строку  $S'$

## 6 Исследование лавинного эффекта алгоритма Кессак в проекте

По определению, хэш-функция обладает свойством лавинного эффекта, если при изменении одного бита входной последовательности меняется в среднем половина битов выходной последовательности. Цель проекта - проверить, выполняется ли данный критерий для алгоритма Кессак.

Для этого в проекте генерируются пары строк, отличающихся на один бит - входные последовательности.

## 7 Список литературы

1. [FIPS PUB 202](#)
2. [SHA-3 Derived Functions](#)
3. [Статья в Википедии про SHA-3](#)
4. [Пост в Habr про Кессак](#)
5. [Статья в Википедии про функцию губки про хэш-функцию](#)
6. [Статья в Википедии про функцию губки](#)