Описание хэш-функции SHA-3

**Оглавление**

[Общее описание хэш-функции 2](#__RefHeading___Toc1460_1562964618)

[Тестирование и описание характеристик реализации 4](#__RefHeading___Toc1462_1562964618)

[Описание тестового стенда 6](#__RefHeading___Toc1323_1566015580)

[Нагрузка на систему во время выполнения программы: 6](#__RefHeading___Toc1464_1562964618)

# Общее описание хэш-функции

Хеш-функции семейства *SHA-3* построены на основе конструкции криптографической губки, в которой данные сначала «впитываются» в губку, при котором исходное сообщении подвергается многораундовым перестановкам{\displaystyle f}, затем результат «отжимается» из губки. На этапе «впитывания» блоки сообщения суммируются по модулю 2 с подмножеством состояния, после чего всё состояние преобразуется с помощью функции перестановки. На этапе «отжимания» выходные блоки считываются из одного и того же подмножества состояния, изменённого функцией перестановок.

Особенности SHA-3:

* Конструкция Sponge. В отличие от конструкции Merkle–Damgård, используемой в SHA-2, SHA-3 «поглощает» входные биты, обрабатывает их, а затем «сжимает» выходные биты
* Настраиваемая безопасность. Структура SHA-3 позволяет варьировать уровни безопасности и производительности в зависимости от потребностей приложения
* Гибкость. SHA-3 предлагает несколько разных размеров дайджеста, а именно SHA3-224, SHA3-256, SHA3-384 и SHA3-512

Длина блока всегда остаётся фиксированной и равной 1600. Однако длина блоков, на которые разбиваются входные данные, никогда не составляет 1600, оно является одним из элементов множества: {1152, 1088, 832, 576}.

Количество раундов фиксировано и равно 24.

Подробно сосредоточимся на коллизиях семейства Keccak. Лучшие предыдущие практические атаки столкновений на семейство Keccak - это Keccak -224 и KECCAK -256, уменьшенные до 4 раундов, найденных Dinur l. В 2012 году и позже представлен в журнальной версии. После этого теоретические результаты улучшились до 5-раундового KECCAK -256. Однако на практике количество раундов остается на уровне 4. Чтобы продвинуть криптоанализ Keccak, команда разработчиков алгоритма предложила уменьшить количество раундов в Keccak challenge, т.е рассматривать хеш с размером 160 для атаки по поиску коллизий и хеш размером 80 для атаки прообраза с каждым из 4 размеров внутренних состояний (state size в 1 шаге дополнения) , уменьшенных до 12 раундов, т. е. полагается равным 0 в 1 шаге дополнения. Идеальные уровни безопасности обоих алгоритмов установлены на уровне 2^80 единичных вычислений для коллизий и прообразов соответственно. Здесь количество вычислений значительно ниже, чем у основных 4 экземпляров SHA-3, однако это количество остается вне досягаемости текущего вычислительного ресурса. Теоретические результаты были найдены Dinur l против KECCAK -256 со сложностями, с использованием обобщенных внутренних дифференциалов. Насколько известно, это остается единственным результатом атаки коллизиями против SHA-3, уменьшенным до 5 раундов на сегодняшний день.

В целом результаты по поиску коллизий и описанная атака прообраза показывают, что на сегодняшний день алгоритм SHA-3 является одним из самых безопасных и эффективных алгоритмов хеширования

# Тестирование и описание характеристик реализации

**Общее описание программной реализации**. В начале приведены константы, необходимые для различных функций, которые используются для хэширования. Основываясь на медленной реализации криптографического алгоритма Кузнечик, автор решил использовать не списки, а массивы библиотеки Numpy.

Функция ***bypass*** принимает на вход путь к файлу (директории), которую нужно хэшировать. Далее она рекурсивно проходит (вглубь) по каждой директории внутри и создаёт список, файлов, который затем хэшируется.

Функция ***get\_bitstring*** принимает на вход строку ASCII символов и трансформирует её в строку, состоящую из бит.

Функция ***get\_bitstring\_from\_file*** принимает на вход путь к файлу, который нужно хэшировать. Далее она считывает из него все байты и вызывает от считанных данных другую функцию.

Функция ***bytes\_to\_bitstring*** принимает на вход контейнер байт и переводит его в битовую строку, приписывая суффикс хэш-функции SHA-3.

Функция ***hex\_to\_array*** переводит заданное шестнадцатеричное число в массив библиотеки Nubpy.

Функция pad формирует дополнение сообщения до нужной длины.

Функции ***theta***, ***rho***, ***pi***, ***chi***, ***iota*** и ***keccak***  реализуют преобразование в соответствии с аналогично названными функциями из спецификации алгоритма SHA-3.

Функция **squeeze** выполняет преобразование массива бит в список байт.

Далее реализован аппарат взаимодействия с программой через консоль. Нужно в консоли зайти в папку, где хранится файл программы (или указать абсолютный путь к файлу программы), затем нужно указать «python3» и текст, удовлетворяющий шаблону:

[-m <message string> | -i <filename>] -o <output-bits (224, 256, 384, or 512)>

В результате имеем, что программа может выполнять хэширование файла, введённого с консоли сообщения и директории (рекурсивно).

Тесты приведены для хэширования в режиме SHA3-512, где размер выходного хэша составляет 512 бит.

Скорость хэширования *пустой строки* составляет: 0.013915300369262695 секунд

Скорость хэширования *одного блока* составляет: 0.0325932502746582 секунд

Скорость хэширования *1000 блоков* составляет: 9.027750253677368 секунд

Скорость (рассчётная) хэширования 1000000 блоков составляет: 150.5 минут

Скорость хэширования *1 Мб* данных: 110.07812762260437 секунд

Скорость (рассчётная) хэширования *1000 Мб* данных: 3,058 часов

Скорость (рассчётная) хэширования *1000000 Мб* данных: 30,58 часов

# Описание тестового стенда

1) Запуск программы производится в интегрированной среде разработки PyCharm;

2) Процессор аппарата, на котором происходит тестирование:

AMD Ryzen 9 7940HS

3) Оперативная память: LDDR5, 4800 МГц

4) Операционная система Linux (дистрибутив Ubuntu)

# Нагрузка на систему во время выполнения программы:

