**SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions.**

**SHA-3 Стандарт: Хеш-функции на основе перестановок и функции расширенного вывода (XOFs)**

**Аннотация**

Настоящий стандарт определяет семейство функций алгоритма безопасного хеширования (**Secure Hash Algorithm-3, SHA-3**) для двоичных данных. Каждая из функций SHA-3 основана на требованиях, установленных алгоритмом ***KECCAK***, который Национальным институтом стандартов и технологий США (**NIST – National Institute of Standards and Technology**) был выбран в качестве победителя конкурса **SHA-3 Cryptographic Hash Algorithm**. Это стандарт также определяет семейство математических перестановок ***KECCAK-p***, включающих перестановку, лежащую в основе *KECCAK*, для облегчения разработки дополнительных криптографических функций на основе перестановок.

Семейство SHA-3 состоит за четырёх криптографических хеш-функций: SHA3-224, SHA3-256, SHA3-384 и SHA3-512; и двух функций расширенного вывода (**XOFs**): SHAKE128 и SHAKE256.

Хеш-функции являются компонентами многих важных приложений информационной безопасности, в частности: 1) генерация и верификация цифровых подписей; 2) формирование ключа; 3) генерация псевдослучайных бит. Хеш-функции, определенные в настоящем стандарте, дополняют хеш-функцию SHA-1 и семейство хеш-функций SHA-2, определенных в стандарте **FIPS (Federal Information Processing Standards) 180-4, the Secure Hash Standard**.

Функции расширенного вывода отличаются от хеш-функций, но их можно использовать аналогичным образом и гибко адаптировать непосредственно к требованиям конкретных приложений с учетом дополнительных соображений безопасности.

**1. Наименование стандарта:** Стандарт SHA-3: Хеш-функции, основанные на перестановках, и функции расширенного вывода (**FIPS PUB 202**).

**2. Категория стандарта:** Стандарт компьютерной безопасности, криптография.

**3. Объяснение:** Настоящий стандарт (FIPS 202) определяет семейство функций алгоритма безопасного хеширования (SHA-3) для двоичных данных. Каждая из функций SHA-3 базируется на принципах алгоритма *KECCAK*, который NIST выбрало в качестве победителя конкурса SHA-3 Cryptographic Hash Algorithm Competition. Это стандарт также определяет семейство математических перестановок *KECCAK-p*, включающих перестановку, лежащую в основе *KECCAK*. Указанные перестановки могут служить основными компонентами дополнительных криптографических функций, которые могут быть определены в будущем.

Семейство SHA-3 состоит из четырёх криптографических хеш-функций и двух функций расширенного вывода (XOFs). Криптографические хеш-функции: SHA3-224, SHA3-256, SHA3-384 и SHA3-512; функции расширенного вывода: SHAKE128 и SHAKE256.

Данные на входе хеш-функций называются **сообщением** (**message**), a на выходе – **дайджестом** (**digest**) или «хешем» (**hash value**). Сообщение может иметь различную длину, длина дайджеста **фиксирована**. Криптографическая хеш-функция – это хеш-функция, которая обладает специальными свойствами, включающими в себя сопротивление коллизиям (**collision resistance**) и сопротивление прообразу (**preimage resistance**), которые важны для многих приложений в области информационной безопасности. К примеру, криптографическая хеш-функция увеличивает безопасность и эффективность схемы цифровой подписи в том случае, когда вместо сообщения цифровой подписью подписывается дайджест. В этом контексте сопротивление хеш-функции коллизиям обеспечивает уверенность в том, что оригинальное сообщение не могло быть изменено на другое сообщение с тем же значением хеш-функции и, следовательно, с той же подписью. Другое приложение криптографических хеш-функций включает в себя генерацию псевдослучайных чисел (**pseudorandom bit generation**), имитовставки (**message authentication codes**) и функции формирования ключа (**key derivation functions**).

Четыре хеш-функции SHA-3, определенные в настоящем стандарте, дополняют хеш-функции, которые определены в **FIPS 180-4**: семейство SHA-1 и SHA-2. Оба стандарта вместе обеспечивают устойчивость к будущим достижениям в области криптоанализа хеш-функций, поскольку они основаны на принципиально разных принципах проектирования. Помимо разнообразия дизайна, хеш-функции в данном стандарте обеспечивают некоторые дополнительную реализацию и характеристики производительности по сравнению с FIPS 180-4.

Длина выходных данных для XOFs может быть выбрана под требования конкретного приложения. Сами функции расширенного вывода могут быть адаптированы для использования в качестве хеш-функций с учётом дополнительных соображений безопасности, или использованы во множестве других приложений. Соответствующее использование XOFs будет определено в специальных публикациях NIST.

Перестановки *KECCAK-p* были разработаны для использования в качестве главных компонентов для множества криптографических функций, включая ключевые функции для аутентификации и/или шифрования (**keyed function for authentication and/or encryption**). Шесть функций SHA-3 можно рассматривать как режимы работы (**modes of operation, modes**) перестановки ***KECCAK-p*[1600, 24]**. В будущем дополнительные моды этой перестановки или другие перестановки *KECCAK-p* могут быть определены и одобрены в публикациях FIPS или специальных публикация NIST (**NIST Special Publications**).

**4. Утверждающий орган (Approving Authority):** Министр торговли (Secretary of Commerce).

**5. Подразделение техподдержки (Maintenance Agency):** Департамент торговли США (U.S.. Department of Commerce), Национальный институт стандартов и технологий (NIST), Лаборатория информационной технологии (**Information Technology Laboratory, ITL**).

**6. Применимость (Applicability):** Данный стандарт применим ко всем федеральным департаментам и агентствам (Federal departments and agencies) по защите конфиденциальной не секретной информации, которая подпадает под действие Раздела 10 United States Code Section 2315 (**10 USC 2315**) и не входит в систему национальной безопасности, что определено в Разделе **40 USC 11103(a)(1)**. Данный стандарт или федеральный стандарт обработки информации FIPS 180 должен быть реализован везде, где требуется безопасный алгоритм хеширования для федеральных приложений (Federal applications), в том числе в качестве компонента других криптографических алгоритмов и протоколов. Настоящий Стандарт может быть принят и использоваться неправительственными организациями (non-Federal Government organization).

**7. Спецификации:** Федеральный стандарт обработки информации (FIPS) 202, Стандарт SHA-3: хеш-функции на основе перестановок и функции расширенного вывода.

**8. Реализации:** Федеральные департаменты и агентства должны использовать реализации перестановок *KECCAK-p* только в режимах работы, одобренных FIPS или рекомендованных NIST, таких как SHA-3 функции, указанные в настоящем стандарте. SHA-3 функции могут быть реализованы в программном обеспечении (software), микропрограммном обеспечении (firmware), оборудовании (hardware) или любой их комбинации. Соответствующими настоящему Стандарту считаются только те реализации этих функций, которые проверены Программой валидации криптографических алгоритмов (**Cryptographic Algorithm Validation Program**). Информация о Программе валидации может быть получена по адресу <http://csrc.nist.gov/groups/STM/cavp/index.html>.

**9. График реализации:** Настоящий Стандарт вступает в силу немедленно. Приложения или расширения настоящего Стандарта, которые зависят от выпуска новых или пересмотренных Специальных публикаций NIST, вступают в силу после окончательной публикации поддерживающих Специальных публикаций.

**10. Патенты:** Реализации SHA-3 функций в настоящем Стандарте могут быть защищены иностранными патентами или патентами США.

**11. Экспортный контроль:** Определенные криптографические устройства и технические данные, относящиеся к ним, подлежат федеральному экспортному контролю. Экспорт криптографических модулей реализующих настоящий Стандарт и относящихся к ним технических данных, должен соответствовать федеральным нормам и иметь лицензию Бюро экспортного управления Министерства торговли США (Bureau of Export Administration of the U.S. Department of Commerce). Информация об экспортном регулировании доступна по адресу: <http://www.bis.doc.gov/index.htm>

**12. Квалификация:** Не смотря на то, что настоящий Стандарт определяет математические функции, являющиеся подходящими компонентами для приложений информационной безопасности, соответствие настоящему Стандарту не гарантирует безопасность конкретной реализации. Ответственный орган в каждом агентстве или департаменте должен гарантировать, что общая реализация обеспечивает приемлемый уровень безопасности. Настоящий Стандарт будет пересматриваться каждые пять лет в порядке оценки его адекватности.

**13. Процедура отказа от требования (Waiver Procedure):** Федеральный закон об управлении информационной безопасностью (**FISMA, Federal Information Security Management Act**) не допускает отказов от обязательных требований FIPS, установленных Министром торговли.

**14. Где можно получить копии стандарта:** Настоящая публикация доступна по адресу <http://csrc.nist.gov/publications/>. Другие публикации по компьютерной безопасности, выпущенные NIST, доступны на том же веб-сайте.

**Содержание**

**1. Введение**

**2. Глоссарий**

2.1 Термины и сокращения

2.2 Параметры алгоритма и другие переменные

2.3 Базовые операции и функции

2.4 Специальные функции

**3. *KECCAK-p* перестановки**

3.1 Состояние (State)

3.1.1 Части матрицы состояния

3.1.2 Конвертация строк в матрицы состояний

3.1.3 Конвертация матриц состояний в строки

3.1.4 Соглашение о маркировке матрицы состояний

3.2 Пошаговое отображение (Step Mapping)

3.2.1 Спецификация***θ***

3.2.2 Спецификация ***ρ***

3.2.3 Спецификация ***π***

3.2.4 Спецификация *χ*

3.2.5 Спецификация ***ι***

3.3 *KECCAK-p*[*b, nr*]

3.4 Сравнение с *KECCAK-f*

**4. Конструкция «губки» (Sponge Construction)**

**5. *KECCAK***

5.1 Спецификация***pad10\*1***

5.2 Спецификация *KECCAK[c]*

**6. SHA-3 Спецификации функции**

6.1 SHA-3 хеш-функции

6.2 SHA-3 функции расширенного вывода

6.3 Альтернативные определения SHA-3 функций расширенного вывода

**7. Соответствие стандарту (Conformance)**

**A. Безопасность**

A.1 Резюме

A.2 Дополнительные соображения о функциях расширенного вывода

**B. Примеры**

B.1 Функции преобразования

B.2 Шестнадцатеричная форма для заполняющих битов (padding bits)

**C. Объектные идентификаторы**

**D. Использованная литература**

**Рисунки**

Рисунок 1: Части матрицы состояния, организованные по размерности

Рисунок 2: ***x***,***y*** и ***z*** координаты для диаграмм пошагового отображения

Рисунок 3: Иллюстрация ***θ***, примененная к одному биту

Рисунок 4: Иллюстрация ***ρ*** для ***b = 200***

Рисунок 5: Иллюстрация ***π***, примененная к одному биту

Рисунок 6: Иллюстрация *χ*, примененная к одному ряду

Рисунок 7: Конструкция «губки»: ***Z=SPONGE*[*f, pad, r*](*N, d*)**

**Таблицы**

Таблица 1: Ширина *KECCAK-p* перестановки и связанные величины

Таблица 2: Смещение **ρ**

Таблица 3: Размеры входных блоков для *HMAC*

Таблица 4: Надежность (security strengths) SHA-1, SHA-2 и SHA-3 функций

Таблица 5: Иллюстрация ***h2b***

Таблица 6: Шестнадцатеричная форма SHA-3 заполнения для сообщений с

байтовым выравниванием

**ВВЕДЕНИЕ**

Настоящий Стандарт определяет новое семейство функций, которое дополняет семейство хеш-функций SHA-1 и SHA-2, определенных в FIPS 180-4 [1]. Это семейство, называемое SHA-3 (Secure Hash Algorithm-3), основано на алгоритме *KECCAK*[2] – победителе конкурса SHA-3 Cryptographic Hash Algorithm Competition[[1]](#footnote-2)[3]. Семейство SHA-3 состоит из четырёх криптографических хеш-функций и двух функций расширенного вывода. Эти шесть функций имеют общую структуру, описанную в [4], а именно, так называемую конструкцию «губки» (***sponge construction***). Функции с данной структурой называются функциями-«губками» (***sponge functions***).

**Хеш-функция** – это функция над двоичными данными (т.е. битовыми строками), для которых длина выходных данных фиксирована[[2]](#footnote-3). Входные данные хеш-функции называются ***сообщением*** (***message***), а выходные данные – ***дайджестом*** (***digest***) или ***хешем*** (**hash value**). Этот дайджест часто служит сжатым представлением сообщения. Четыре SHA-3 хеш-функции называются SHA3-224, SHA3-256, SHA3-384 и SHA3-512; в каждом случае суффикс после тире указывает на фиксированную длину дайджеста, к примеру, SHA3-256 продуцирует 256 битный дайджест. Функции SHA-2, т.e. SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 и SHA-512/256, предлагают тот же набор длины дайджеста. Таким образом, SHA-3 хеш-функции могут быть применены как альтернатива функциям SHA-2 и наоборот.

**Функция расширенного вывода**(**XOFs**) – это функции над битовыми строками (также называемыми сообщениями), в которых выходные данные могут быть расширены на любую желаемую длину. Две SHA-3 функции расширенного вывода называются SHAKE128 и SHAKE256[[3]](#footnote-4). Суффиксы «128» и «256» указывают криптографическую стойкость (степень/уровень безопасности – **security strength**), которую обычно могут поддерживать эти две функции[[4]](#footnote-5), в отличии от суффиксов хеш-функций, которые указывают длину дайджеста. SHAKE128 и SHAKE256 не первые функции расширенного вывода, которые были стандартизированы NIST.

Шесть функций SHA-3 были разработаны для предоставления специальных свойств, таких как сопротивление коллизии (**resistance to collision**), сопротивление атакам нахождения первого и второго прообразов (**preimage and second preimage attacks**). Уровень сопротивления этим трём типам атак кратко изложен в Разделе A.1. Криптографические хеш-функции являются фундаментальными компонентами во многих приложения информационной безопасности, таких как генерация и проверка цифровой подписи (**digital signature generation and verification**), а также генерация псевдослучайных чисел (**pseudorandom bit generation**).

Длина дайджеста в одобренных FIPS хеш-функциях составляет 160, 224, 256, 384 и 512 бит. Когда в приложении требуется криптографическая хеш-функция с нестандартной длиной дайджеста, XOF является естественной альтернативой конструкциям, включающим множественные вызовы хеш-функции и/или усечение выходных бит. Однако, на XOFs распространяются дополнительные соображения безопасности, описанные в Разделе A.2.

Каждая из шести SHA-3 функций использует одну и ту же базовую перестановку в качестве основного компонента конструкции «губки». По сути SHA-3 функции являются режимами работы (модами) базовой перестановки (**modes of operation, modes**). В настоящем Стандарте перестановка определена как экземпляр семейства перестановок, называемого KECCAK-p, чтобы обеспечить гибкость изменения ее размера и параметров безопасности при разработке любых дополнительных модов в будущих документах.

Четыре SHA-3 хеш-функции немного отличаются от экземпляров KECCAK, предложенных для конкурса SHA-3 [3]. В частности, к сообщениям добавлен двубитный суффикс для того, чтобы отличить SHA-3 хеш-функцию от SHA-3 XOFs и облегчить разработку новых вариантов SHA-3 функций, которые могут быть предназначены для отдельных доменов приложений.

Две функции расширенного вывода SHA-3 определены таким образом, чтобы обеспечить разработку специальных вариантов (**dedicated variants**). Кроме того, функции расширенного вывода SHA-3 совместимы со схемой кодирования Sakura (**Sakura coding scheme**) [6] для дерева хешей (**tree hashing**) [7] для того, чтобы поддерживать разработку параллелизуемых вариантов (**parallelizable variants**) XOFs, определённых в отдельном документе.

Большая часть обозначений и терминологии в настоящем Стандарте согласуется со спецификацией KECCAK в [8].

**2. ГЛОССАРИЙ**

**2.1 Термины и Сокращения**

|  |  |
| --- | --- |
| бит (**bit**) | Двоичная цифра: 0 или 1. |
| байт (**byte**) | Последовательность из 8 бит |
| емкость (**capacity**) | В конструкции «губки»: ширина базовой функции минус скорость |
| столбец (**column**) | Для матрицы состояния: подмассив из пяти бит с константными *x* и *z* координатами |
| дайджест (**digest**) | Данные на выходе криптографической хеш-функции. Также называются хешем. |
| разделение доменов (**domain separation**) | Для функции: разделение данных на разные домены приложений для того, чтобы ни один вход не был назначен более чем одному домену. |
| функция расширенного вывода (**extendable-output function**, **XOF)** | Функция над битовой строкой, в которой выходные данные могут быть расширены на любую желаемую длину |
| **FIPS** | Федеральный стандарт обработки информации (**Federal Information Processing Standard**) |
| **FISMA** | Федеральный закон об управлении информационной безопасностью (**Federal Information Security Management Act**) |
| хеш-функция (**hash function**) | Функция над битовой строкой, в которой длина входных данных фиксирована. Выходные данные часто служат сжатым представлением входных данных |
| хеш (**hash value**) | Смотри «дайджест» |
| **HMAC** | Код аутентификации сообщений, использующий хеш-функцию с ключом (**Keyed-Hash Message Authentication Code**). |
| **KDF** | Функция формирования ключа (**Key derivation function**). |
| **KECCAK** | Семейство всех функций «губок» с перестановкой *KECCAK-f* в качестве базовой функции и мультискоростным заполнением (**multi-rate padding**) в качестве правила заполнения (**padding rule**). KECCAK изначально был указан в [8]. |
| полоса (**lane**) | Для матрицы состояния перестановки *KECCAK-p* с шириной *b*, подмассив из *b/25*бит с константными *x* и *y* координатами. |
| сообщение (**message**) | Битовая строка произвольной длины, поступающая на вход SHA-3 функции. |
| мультискоростное заполнение (**multi-rate padding**) | Правило заполнения *pad10\*1*, вывод которого равен 1, за которым следует (возможно, пустая) строка из 0, за которой следует 1 (100…..001). |
| **NIST** | Национальный институт стандартов и технологии (**National Institute of Standards and Technology**) |
| плоскость (**plane**) | Для матрицы состояния перестановки *KECCAK-p* c шириной *b*, подмассив из *b/*5 бит с константной *y* координатой. |
| скорость (**rate**) | В конструкции «губки» количество обработанных входных битов или выходных битов, сгенерированных при каждом вызове базовой функции. |
| раунд (**round**) | Последовательность пошаговых отображений, которая повторяется при вычислении перестановки *KECCAK-p* |
| константа раунда (**round constant**) | Для каждого раунда перестановки *KECCAK-p* значение полосы, которая определяется индексом раунда. Константа раунда является вторым входом в пошаговое отображение ***ι*** |
| индекс раунда (**round index**) | Целое значение индекса для раундов перестановки *KECCAK-p*. |
| ряд (**row**) | Для матрицы состояния, подмассив из пяти бит с константными *y* и *z* координатами. |
| **SHA-3** | Алгоритм безопасного хеша 3 (**Secure Hash Algorithm-3**). |
| **SHAKE** | **S**ecure **H**ash **A**lgorithm **KE**CCAK. |
| слой (**sheet**) | Для матрицы состояния перестановки *KECCAK-*p с длиной b подмассив из *b/*5 бит с константной *x* координатой. |
| срез (**slice**) | Для матрицы состояния, подмассив из 25 бит с константной *z* координатой |
| конструкция губки (**sponge construction**) | Метод, первоначально указанный в [4] для определения функции исходя из следующего: 1) базовая функция над битовой строкой фиксированной длины; 2) правило заполнения и 3) скорость. Входные и выходные данные результирующей функции – это битовые строки, которые могут быть сколь угодно длинными. |
| функция губки (**sponge function**) | Функция, которая определяется в соответствии с конструкцией «губки», возможно специализированная для фиксированной длины выходных данных |
| состояние (**state**) | Массив битов, который многократно обновляется в вычислительной процедуре. Для перестановки *KECCAK-p* состояние представляется либо как трёхмерный массив, либо как битовая строка. |
| матрица состояния (**state array)** | Для перестановки *KECCAK-p*, массив размера 5\*5\**w* бит, который представляет состояние. Индексы для x, y и z координат находятся в диапазонах [0, 4], [0, 4] и [0, *w*-1] соответственно. |
| пошаговое отображение (**step mapping**) | Один из пяти компонентов раунда перестановки KECCAK-p: **θ**, **ρ**, **π**, **χ** или **ι** |
| строка (**string**) | Для неотрицательных целых *m*, последовательность из *m* символов |
| ширина (**width)** | В конструкции губки, фиксированная длина входных и выходных данных базовой функции |
| **XOF** | ***Extendable-Output Function*** |
| **XOR** | Булева операция **исключающее ИЛИ**, обозначается знаком ⨁ (деление по модулю 2) |

**2.2 Параметры алгоритма и другие переменные**

|  |  |
| --- | --- |
| **A** | Матрица состояния |
| **A[*x*, *y*, *z*]** | Для матрицы состояния **A**, бит соответствующий тройке (x, y, z). |
| ***b*** | Ширина перестановки *KECCAK-p* в битах |
| ***c*** | Емкость функции «губки» |
| ***d*** | Длина дайджеста хеш-функции или требуемая длина выходных данных XOF в битах |
| ***f*** | Общая базовая функция для конструкции «губки» |
| ***ir*** | Индекс раунда для перестановки *KECCAK-p*. |
| ***J*** | Входная строка для RawSHAKE128 и RawSHAKE256 |
| ***l*** | Для перестановки KECCAK-p двоичный логарифм от размера полосы, т.е. log2(*w*). |
| ***Lane*(*i, j*)** | Для матрицы состояния **A** – строка всех битов полосы, чьи координаты *x* и *y* равны *i* и *j*. |
| ***M*** | Входная строка для SHA-3 хеш-функции или функции расширенного вывода. |
| ***N*** | Входная строка для конструкции «губки» SPONGE[*f*, pad, *r*] KECCAK[*c*]. |
| ***nr*** | Количество рандов для перестановки *KECCAK-p* |
| **pad** | Обобщенное правило заполнения для конструкции «губки». |
| ***Plane*(*j*)** | Для матрицы состояний **A** – строка всех битов плоскости, у которых координата *y* равна *j*. |
| ***r*** | Скорость функции «губки» |
| ***RC*** | Для раунда перестановки *KECCAK-p* – константа раунда |
| ***w*** | Размер полосы перестановки *KECCAK-p* в битах, т.е. *b/25*. |

**2.3 Базовые операции и функции**

|  |  |
| --- | --- |
| **0S** | Для положительного целого *s*, 0S это строка, которая состоит из *s* последовательных нулей. Если *s*=0, тогда это пустая строка |
| **len(*X*)** | Для битовой строки *X*, len(*X*) это длина *X* в битах. |
| ***X*[*i*]** | Для строки *X* и целого *i* такого, что 0 ≤ *i* < len(*X*), X[*i*] – бит *X* с индексом *i*. Битовые строки изображаются с индексами, возрастающими слева направо, так что *X*[0] появляется слева, затем *X*[1] и т.д. |
| **TruncS(*X*)** | Для положительного целого *s* и строки X TruncS(*X*) – это строка, состоящая из битов *X*[0]…*X*[*s*-1]. Например, если Trunc2(10100)=10. |
| ***X* ⨁ *Y*** | *Сложение по модулю 2* (**XOR**)  Для строк *X* и *Y* равной длины, *X*⨁*Y* – это строка, которая является результатом булевой операции «Исключающее ИЛИ» над *X* и *Y*. Например, 1100 ⨁ 1010 = 0110. |
| ***X || Y*** | Для строк *X* и *Y* *X* || *Y* – это конкатенация *X* и *Y*. Например, 11001 || 010 = 11001010. |
| ***m/n*** | Для целых *m* и *n*, *m*/*n* – это частное (**quotient**), т.е. *m* деленное на *n*. |
| ***m* mod *n*** | Для целых *m* и *n*, *m* mod *n* – это целое *r*, для которого  0 ≤ *r* < *n* и *m*–*r* кратно *n*. Например, 11 mod 5 = 1, и –11 mod 5 = 4 |
| **⌈*x*⌉** | Для действительного числа *x*, ⌈*x*⌉ – это наименьшее целое, которое не строго меньше *x*. Например, ⌈3.2⌉=4,  ⌈-3.2⌉ = -3, ⌈6⌉=6 |
| **log2(*x*)** | Для положительного действительного числа *x*, log2(*x*) это действительное число *y*, такое что 2*y*=*x*. |
| **min(*x*, *y*)** | Для действительных чисел *x* и *y*, min(*x*, *y*) является минимумом *x* и *y*. Например, min(9, 33)=9. |

**2.4 Определенные функции**

В настоящем Стандарте определены следующие функции более высокого уровня:

|  |  |
| --- | --- |
| **θ**, **ρ**, **π**, **χ**, **ι** | Пять пошаговых отображений, составляющие раунд. |
| **KECCAK[*c*]** | Экземпляр KECCAK с KECCAK-*f*[1600] в качестве базовой перестановки и емкости *c*. |
| **KECCAK-*f*[*b*]** | Семейство из семи перестановок, первоначально указанное в [8] как базовая функция для KECCAK. Набор значений ширины перестановок *b* равен {25, 50, 100, 200, 400, 800, 1600}. |
| **KECCAK-*p*[*b*, *nr*]** | Обобщение перестановок KECCAK, которое определено в настоящем Стандарте путём преобразования количества раундов *nr* во входной параметр. |
| **pad10\*1** | Правило мультискоростного (**multi-rate**) заполнения для KECCAK, первоначально указанное в [8]. |
| **RawSHAKE128** | Промежуточная функция в альтернативном определении SHAKE128 |
| **RawSHAKE256** | Промежуточная функция в альтернативном определении SHAK256 |
| ***rc*** | Функция, которая генерирует переменные биты для раундовых констант. |
| **Rnd** | Функция раунда перестановки KECCAK. |
| **SHA3-224** | Хеш-функция SHA-3, продуцирующая 224-битный дайджест |
| **SHA3-256** | Хеш-функция SHA-3, продуцирующая 256-битный дайджест |
| **SHA3-384** | Хеш-функция SHA-3, продуцирующая 384-битный дайджест |
| **SHA3-512** | Хеш-функция SHA-3, продуцирующая 512-битный дайджест |
| **SHAKE128** | Функция расширенного вывода SHA-3, в общем случае поддерживающая 128 бит криптографической стойкости (**security strength**), если выходные данные достаточно длинные; смотри Раздел A.1 |
| **SHAKE256** | Функция расширенного вывода SHA-3, в общем случае поддерживающая 256 бит криптографической стойкости (**security strength**), если выходные данные достаточно длинные; смотри Раздел A.1 |
| **SPONGE[*f*, pad, *r*]** | Функция «губка», в которой базовая функция – *f*, правило заполнения – pad и скорость – *r*. |

**3. *KECCAK-p* перестановки**

В данном разделе определены перестановки KECCAK-p с двумя параметрами: 1) фиксированная длина переставляемых строк, называемая шириной перестановки (**width of the permutation**); 2) количество итераций внутренней трансформации, называемое раундом (**round**). Ширина обозначается как *b*, а количество раундов – *nr*. Перестановка KECCAK-*p* с *nr* раундами и шириной *b* обозначается как KECCAK-*p*[*b*, *nr*]; перестановка определена для любого *b* из множества {25, 50, 100, 200, 400, 800, 1600} и любого положительного целого *nr*.

Раунд перестановки KECCAK-p, обозначенный как Rnd, состоит из последовательности пяти трансформаций, которые называются пошаговыми отображениями (**step mappings**). Перестановка определена в терминах циклично обновляемого массива значений для *b* бит, который называется состоянием (**state**). Состояние изначально устанавливается входными значениями перестановки.

Обозначения и терминология для состояния описаны в Разделе 3.1. Пошаговые отображения определены в Разделе 3.2. Перестановка KECCAK-p, включающая функцию раунда Rnd, определена в Разделе 3.3. Отношение перестановки KECCAK-p к перестановкам KECCAK-f, определенные для KECCAK в [8], описаны в Разделе 3.4.

**3.1 Состояние (State)**

Состояние для перестановки KECCAK-*p*[*b*, *nr*] состоит из *b* бит. Спецификации в настоящем Стандарте содержат 2 другие величины, связанные с *b*: *b*/25 и log2(*b*/25), обозначаемые как ***w*** и ***l*** соответственно. Семь возможных значений для этих переменных, определенных для перестановок KECCAK-p, приведены ниже в Таблице 1.

**Таблица 1:** Ширина перестановки KECCAK-p и соответствующие величины

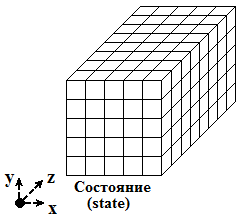
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *b* | 25 | 50 | 100 | 200 | 400 | 800 | 1600 |
| *w* | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
| *l* | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

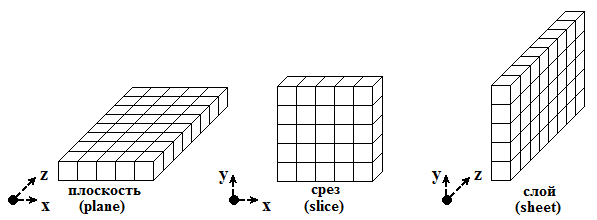
Удобно представлять входные и выходные состояния перестановки как строку бит длиной *b*, а входные и выходные состояния пошаговых отображений как массив бит размера 5\*5\**w*. Если обозначить строку, представляющую состояние, как *S*, то её биты индексируются от 0 до *b*–1, т.е.:

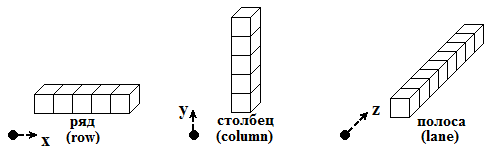
*S* = *S*[0] || *S*[1] || … || *S*[*b*-2] || *S*[*b*-1].

Если **A** обозначает массив бит размером 5\*5\**w*, представляющий состояние, то его индексы представляют собой целочисленные тройки (*x*, *y*, *z*), для которых 0≤*x*<5, 0≤*y*<5 и 0≤*z*<*w*. Бит, соответствующий (*x*, *y*, *z*), обозначается **A**[*x*, *y*, *z*]. Матрица состояния – это представление состояния в виде трехмерного массива, которые индексируется вышеуказанным образом.

**3.1.1 Части матрицы состояний**









**Рисунок 1:** Части матрицы состояния, организованные по измерениям [8].

Матрица состояния для перестановки KECCAK и его подмассивы меньшей размерности показаны выше на Рисунке 1 для случая *b* = 200 (*w* = 8 соответственно). Двумерные подмассивы называются ***слоями***, ***плоскостями*** и ***срезами*** (**sheets**, **planes** и **slices**); одномерные массивы называются ***рядами***, ***столбцами*** и ***полосами*** (**rows**, **columns** и **lanes**). Алгебраические определения для этих подмассивов даны в Глоссарии (в Разделе 2.1).

**3.1.2 Конвертация строк в матрицы состояний**

Пусть *S* обозначает строку из *b* бит, которая представляет состояние перестановки KECCAK-*p*[*b*, *nr*]. Соответствующая матрица состояния, обозначаемая **A**, определена следующим образом:

Для всех троек (*x*, *y*, *z*) таких, что 0≤ *x* <5, 0≤ *y* <5 и 0≤ *z* <*w*,

**A**[*x*, *y*, *z*] = *S*[*w*(5*y* + *x*) + *z*].

Например, если *b* = 1600 (*w* = 64), тогда:

**A**[0, 0, 0] = *S*[0] **A**[1, 0, 0] = *S*[64] **A**[4, 0, 0] = *S*[256]

**A**[0, 0, 1] = *S*[1] **A**[1, 0, 1] = *S*[65] **A**[4, 0, 1] = *S*[257]

**A**[0, 0, 2] = *S*[2] **A**[1, 0, 2] = *S*[66] **A**[4, 0, 2] = *S*[258]

⁞ ⁞ … ⁞

**A**[0, 0, 61] = *S*[61] **A**[1, 0, 61] = *S*[125] **A**[4, 0, 61] = *S*[317]

**A**[0, 0, 62] = *S*[62] **A**[1, 0, 62] = *S*[126] **A**[4, 0, 62] = *S*[318]

**A**[0, 0, 63] = *S*[63] **A**[1, 0, 63] = *S*[127] **A**[4, 0, 63] = *S*[319]

и

**A**[0, 1, 0] = *S*[320] **A**[1, 1, 0] = *S*[384] **A**[4, 1, 0] = *S*[576]

**A**[0, 1, 1] = *S*[321] **A**[1, 1, 1] = *S*[385] **A**[4, 1, 1] = *S*[577]

**A**[0, 1, 2] = *S*[322] **A**[1, 1, 2] = *S*[386] **A**[4, 1, 2] = *S*[578]

⁞ ⁞ ⁞

**A**[0, 1, 61] = *S*[381] **A**[1, 1, 61] = *S*[445] **A**[4, 1, 61] = *S*[637]

**A**[0, 1, 62] = *S*[382] **A**[1, 1, 62] = *S*[446] **A**[4, 1, 62] = *S*[638]

**A**[0, 1, 63] = *S*[383] **A**[1, 1, 63] = *S*[447] **A**[4, 1, 63] = *S*[639]

и

**A**[0, 2, 0] = *S*[640] **A**[1, 2, 0] = *S*[704] **A**[4, 2, 0] = *S*[896]

**A**[0, 2, 1] = *S*[641] **A**[1, 2, 1] = *S*[705] **A**[4, 2, 1] = *S*[897]

**A**[0, 2, 2] = *S*[642] **A**[1, 2, 2] = *S*[706] **A**[4, 2, 2] = *S*[898]

⁞ ⁞ ⁞

**A**[0, 2, 61] = *S*[701] **A**[1, 2, 61] = *S*[765] **A**[4, 2, 61] = *S*[957]

**A**[0, 2, 62] = *S*[702] **A**[1, 2, 62] = *S*[766] **A**[4, 2, 62] = *S*[958]

**A**[0, 2, 63] = *S*[703] **A**[1, 2, 63] = *S*[767] **A**[4, 2, 63] = *S*[959]

и т.д.

**3.1.3 Конвертация матриц состояний в строки**

Пусть **A** – матрица состояния. Соответствующее строковое представление, обозначаемое *S*, может быть получено из *полос* и *плоскостей* **A** следующим образом:

Для каждой пары целых (*i*, *j*) таких, что 0 ≤ *i* < 5 и 0 ≤ *j* < 5, определим строку *Lane*(*i*, *j*):

*Lane*(*i*, *j*) = A[*i*, *j*, 0] || **A**[*i*, *j*, 1] || **A**[*i*, *j*, 2] || … || **A**[*i*, *j*, *w*-2] || **A**[*i*, *j*, *w*-1].

Например, если *b* = 1600 (*w* = 64), то:

*Lane*(0, 0) = A[0, 0, 0] || **A**[0, 0, 1] || **A**[0, 0, 2] || … || **A**[0, 0, 62] || **A**[0, 0, 63]

*Lane*(1, 0) = A[1, 0, 0] || **A**[1, 0, 1] || **A**[1, 0, 2] || … || **A**[1, 0, 62] || **A**[1, 0, 63]

*Lane*(2, 0) = A[2, 0, 0] || **A**[2, 0, 1] || **A**[2, 0, 2] || … || **A**[2, 0, 62] || **A**[2, 0, 63]

и т.д.

Для каждого целого *j* такого, что 0 ≤ *j* < 5, определим строку *Plane*(*j*):

*Plane*(*j*) = *Lane*(0, *j*) || *Lane*(1, *j*) || *Lane*(2, *j*) || *Lane*(3, *j*) || *Lane*(4, *j*).

Тогда

S = *Plane*(0) || *Plane*(1) || *Plane*(2) || *Plane*(3) || *Plane*(4)

Например, если *b* = 1600 (*w* = 64), то:

*S* = **A**[0, 0, 0] || **A**[0, 0, 1] || **A**[0, 0, 2] || … || **A**[0, 0, 62] || **A**[0, 0, 63]

|| **A**[1, 0, 0] || **A**[1, 0, 1] || **A**[1, 0, 2] || … || **A**[1, 0, 62] || **A**[1, 0, 63]

|| **A**[2, 0, 0] || **A**[2, 0, 1] || **A**[2, 0, 2] || … || **A**[2, 0, 62] || **A**[2, 0, 63]

|| **A**[3, 0, 0] || **A**[3, 0, 1] || **A**[3, 0, 2] || … || **A**[3, 0, 62] || **A**[3, 0, 63]

|| **A**[4, 0, 0] || **A**[4, 0, 1] || **A**[4, 0, 2] || … || **A**[4, 0, 62] || **A**[4, 0, 63]

**A**[0, 1, 0] || **A**[0, 1, 1] || **A**[0, 1, 2] || … || **A**[0, 1, 62] || **A**[0, 1, 63]

|| **A**[1, 1, 0] || **A**[1, 1, 1] || **A**[1, 1, 2] || … || **A**[1, 1, 62] || **A**[1, 1, 63]

|| **A**[2, 1, 0] || **A**[2, 1, 1] || **A**[2, 1, 2] || … || **A**[2, 1, 62] || **A**[2, 1, 63]

|| **A**[3, 1, 0] || **A**[3, 1, 1] || **A**[3, 1, 2] || … || **A**[3, 1, 62] || **A**[3, 1, 63]

|| **A**[4, 1, 0] || **A**[4, 1, 1] || **A**[4, 1, 2] || … || **A**[4, 1, 62] || **A**[4, 1, 63]

⁞

**A**[0, 4, 0] || **A**[0, 4, 1] || **A**[0, 4, 2] || … || **A**[0, 4, 62] || **A**[0, 4, 63]

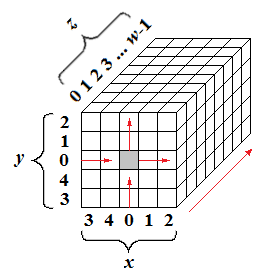
|| **A**[1, 4, 0] || **A**[1, 4, 1] || **A**[1, 4, 2] || … || **A**[1, 4, 62] || **A**[1, 4, 63]

|| **A**[2, 4, 0] || **A**[2, 4, 1] || **A**[2, 4, 2] || … || **A**[2, 4, 62] || **A**[2, 4, 63]

|| **A**[3, 4, 0] || **A**[3, 4, 1] || **A**[3, 4, 2] || … || **A**[3, 4, 62] || **A**[3, 4, 63]

|| **A**[4, 4, 0] || **A**[4, 4, 1] || **A**[4, 4, 2] || … || **A**[4, 4, 62] || **A**[4, 4, 63]

**3.1.4 Соглашение о маркировке матрицы состояния**



**Рисунок 2:** *x*, *y* и *z* координаты для диаграмм пошаговых отображений (красная стрелка – направление увеличение индекса)

На диаграммах состояния, сопровождающих спецификации пошаговых отображений, *полоса*, соответствующая координатам (*x*, *y*) = (0, 0), изображается в центре *среза*. Полная маркировка координат *x*, *y* и *z* для этих диаграмм показана выше на Рисунке 2.

**3.2 Пошаговое отображение (Step Mapping)**

Пять пошаговых отображений, составляющих раунд KECCAK-*p*[*b*,*nr*], обозначаются как **θ**, **ρ**, **π**, **χ** и **ι**. Спецификации для этих функций приведены в Разделах 3.2.1 – 3.2.5.

Алгоритм для каждого пошагового отображения на вход получает матрицу состояния, обозначаемую **A**, и возвращает изменённую матрицу состояния, обозначаемую **Aʹ**, на выходе. Размер состояния – это параметр, который опущен в нотации, поскольку *b* всегда определено при вызове пошагового отображения.

Отображение **ι** имеет второй вход: целое число, называемое *индексом раунда* (***round index***) и обозначаемое ***ir***, которое определено с помощью Алгоритма 7 для KECCAK-p (в Разделе 3.3). Прочие пошаговые отображения не зависят от индекса раунда.

**3.2.1 Спецификация *θ***

АЛГОРИТМ 1: θ(**A**)

Входные данные:

матрица состояния **A**.

Выходные данные:

матрица состояния **Aʹ**.

Шаги алгоритма:

1. Для всех пар (*x*, *z*) таких, что 0 ≤ *x* < 5 и 0 ≤ *z* < w, пусть

*C*[*x*, *z*] = **A**[*x*, 0, *z*] ⨁ **A**[*x*, 1, *z*] ⨁ **A**[*x*, 2, *z*] ⨁ **A**[*x*, 3, *z*] ⨁ **A**[*x*, 4, *z*].

2. Для всех пар (*x*, *z*) таких, что 0 ≤ *x* < 5 и 0 ≤ *z* < w, пусть

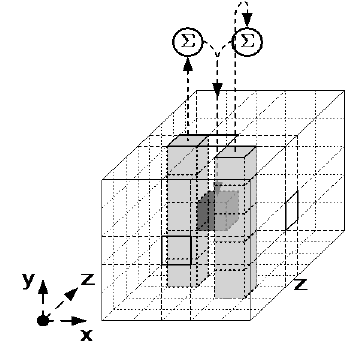
*D*[x, z] = *C*[(*x*–1) mod 5, *z*] ⨁ *C*[(*x*+1) mod 5, (*z*–1) mod *w*].

3. Для всех троек (*x*, *y*, *z*) таких, что 0 ≤ *x* < 5, 0 ≤ *y* < 5 и 0 ≤ *z* < w, пусть

**Aʹ**[*x*, *y*, *z*] = **A**[*x*, *y*, *z*] ⨁ D[*x*, *z*].

Эффект **θ** заключается в следующем: каждый бит в матрице состояния складывается по модулю 2 (операция XOR) с четностью двух соседних столбцов в матрице (*parities of two columns in the array*). В частности, для бита **A**[*x0*, *y0*, *z0*]: *x*-координата одного из столбцов равна (*x0*–1) mod 5 (*z*-координата та же), в то время как *x*-координата другого столбца (*x0*+1) mod 5, а *z*-координатаa – (*z0*—1) mod *w*.

Ниже на Рисунке 3 проиллюстрировано пошаговое отображение **θ**. Символ суммирования **Σ** указывает на четность (***parity***), т.е. XOR-сумму всех битов в столбце.



**Рисунок 3:** Иллюстрация **θ** для одного бита [8]

**3.2.2 Спецификация *ρ***

АЛГОРИТМ 2: ρ(**A**)

Входные данные:

матрица состояния **A**.

Выходные данные:

матрица состояния **Aʹ**.

Шаги алгоритма:

1. Для всех *z* таких, что 0 ≤ *z* < *w*, пусть

**Aʹ**[0, 0, *z*] = **A**[0, 0, *z*].

2. Пусть (*x*, *y*) = (1, 0).

3. Для *t* от 0 до 23:

a. для всех *z* (0 ≤ *z* < w): **Aʹ**[*x*, *y*, *z*] = **A**[*x*, *y*, (*z*–(t+1)(t+2)/2) mod *w*];

b. пусть (*x*, *y*) = (*y*, (2*x*+3*y*) mod 5).

4. Вернуть **Aʹ**.

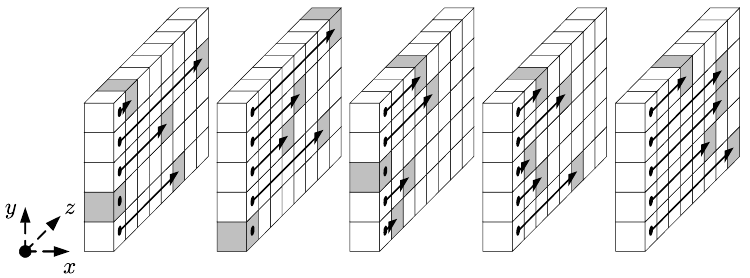
Эффект **ρ** заключается в ротации битов каждой *полосы* на величину, называемую *смещением* (***offset***), которая зависит от фиксированных *x* и *y* координат полосы. Эквивалентно, для каждого бита полосы, z-координата изменяется путём добавления смещения, деленного по модулю на длину полосы (***modulo the lane size***).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | ***x* = 3** | ***x* = 4** | ***x* = 0** | ***x* = 1** | ***x* = 2** |
| ***y* = 2** | 153 | 231 | 3 | 10 | 171 |
| ***y* = 1** | 55 | 276 | 36 | 300 | 6 |
| ***y* = 0** | 28 | 91 | 0 | 1 | 190 |
| ***y* = 4** | 120 | 78 | 210 | 66 | 253 |
| ***y* = 3** | 21 | 136 | 105 | 45 | 15 |

**Таблица 2:** Смещение **ρ** [8]

Смещение для каждой полосы, полученные в результате вычисления на шаге 3a в Алгоритме 2, перечислены выше в таблице 2.

Иллюстрация **ρ** для случая *w* = 8 приведена ниже на Рисунке 4. Соглашение об обозначении *x* и *y* координат на Рисунке 4 явно показано на Рисунке 2 и соответствует обозначению строк и столбцов в Таблице 2. Например, полоса **A**[0,0] изображена в середине среднего слоя, а полоса **A**[2,3] изображена внизу самого правого слоя.



**Рисунок 4:** Иллюстрация **ρ** для *b* = 200 [8]

Для каждой полосы на Рисунке 4 черная точка показывает бит с координатой *z* = 0, серый куб показывает позицию, на которую переместится этот бит после выполнения **ρ**. Другие биты в полосе циклически сдвигаются на ту же самую величину. Например, смещение полосы **A**[1, 0] равно 1, следовательно, последний бит, *z*-координата которого равна 7, сдвигается на переднюю позицию с *z*-координатой 0. Таким образом, смещения могут быть уменьшены по модулю на размер полосы (modulo *w*); например, полоса **A**[3,2], верхняя в самом левом слое, имеет смещение 153 mod 8 = 1 для приведённого примера.

**3.2.3 Спецификация *π***

АЛГОРИТМ 3: π(**A**)

Входные данные:

матрица состояния **A**.

Выходные данные:

матрица состояния **Aʹ**.

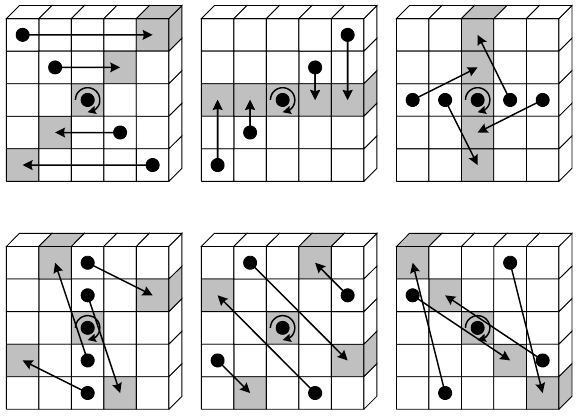
Шаги алгоритма:

1. Для всех троек (*x*, *y*, *z*) таких, что 0 ≤ *x* < 5, 0 ≤ *y* < 5 и 0 ≤ *z* < *w*, пусть

**Aʹ**[*x*, *y*, *z*] = **A**[(*x* + 3*y*) mod 5, *x*, *z*].

2. Вернуть **Aʹ**.

Эффект **π** заключается в изменении положения полос, как показано ниже для любого среза на Рисунке 5. Соглашение о наименовании координат представлено выше на Рисунке 2; например, бит с координатами *x* = *y* = 0 изображен в центре среза.



**Рисунок 5:** Иллюстрация применения **π** к отдельному срезу [8]

**3.2.4 Спецификация *χ***

АЛГОРИТМ 4: χ(**A**)

Входные данные:

матрица состояния **A**.

Выходные данные:

матрица состояния **Aʹ**.

Шаги алгоритма:

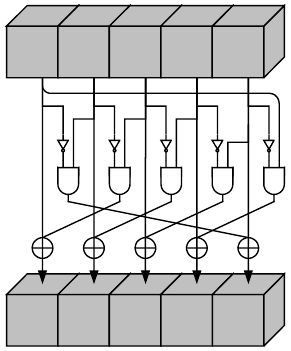
1. Для всех троек (*x*, *y*, *z*) таких, что 0 ≤ *x* < 5, 0 ≤ *y* < 5 и 0 ≤ *z* < *w*, пусть

**Aʹ**[*x*, *y*, *z*] = **A**[*x*, *y*, *z*] ⨁ ((**A**[(*x* + 1) mod 5, *y*, *z*] ⨁ 1) **∙** **A**[(*x* + 2) mod 5, *y*, *z*]).

2. Вернуть **Aʹ**.

Точка «**∙**» в правой части присваивания на шаге 1 указывает на целочисленное умножение (*integer multiplication*), которое в данном случае эквивалентно подразумеваемой булевой операции «И» (Boolean “AND” operation).

Результатом **χ** является сложение по модулю каждого бита с нелинейной функцией от двух других бит в этом же ряду (XOR each bit with a non-linear function of two other bits in its row), как показано ниже на Рисунке 6.



**Рисунок 6:** Иллюстрация применения **χ** к отдельному ряду [8]

**3.2.5 Спецификация *ι***

Отображение **ι** параметризуется индексом ранда ***ir***, значения которого определены на шаге 2 Алгоритма 7 для вычисления KECCAK-*p*[*b*, *nr*], в Разделе 3.3. Ниже в спецификации **ι** в Алгоритме 6 это параметр определяет *l* + 1 бит значения полосы, называется константой раунда и обозначается *RC*. Каждый из этих *l* + 1 бит генерируется функцией, которая основана на *регистре сдвига с линейной обратной связью* (**liner feedback shift register**). Эта функция, обозначенная как *rc*, определена в Алгоритме 5.

АЛГОРИТМ 5: *rc*(*t*)

Входные данные:

Целое *t*.

Выходные данные:

Бит *rc*(*t*).

Шаги алгоритма:

1. Если *t* mod 255 = 0, вернуть 1.

2. Пусть *R* = 1000 0000.

3. Для *i* от 1 до *t* mod 255, пусть

a. R = 0 || R;

b. R[0] = R[0] ⨁ R[8];

c. R[4] = R[4] ⨁ R[8];

d. R[5] = R[5] ⨁ R[8];

e. R[6] = R[6] ⨁ R[8];

f. R = Trunc8[R].

4. Вернуть *R*[0].

АЛГОРИТМ 6: **ι**(A, *ir*)

Входные данные:

Матрица состояния **A**;

Индекс раунда ***ir***.

Выходные данные:

матрица состояния **Aʹ**.

Шаги алгоритма:

1. Для всех троек (*x*, *y*, *z*) таких, что 0 ≤ *x* < 5, 0 ≤ *y* < 5 и 0 ≤ *z* < *w*, пусть

**Aʹ**[*x*, *y*, *z*] = **A**[*x*, *y*, *z*]

2. Пусть *RС* = 0*w*.

3. Для *j* от 0 до *l*, пусть *RC*[2*j* – 1] = *rc*(*j* + 7*ir*).

4. Для всех z таких, что 0 ≤ *z* < *w*, пусть **Aʹ**[0, 0, *z*] = **Aʹ**[0, 0, *z*] ⨁ *RC*[*z*]

5. Вернуть **Aʹ**.

Эффект **ι** заключается в изменении некоторых битов центральной полосы (*Lane*(0,0)) в зависимости от индекса раунда ***ir***. Другие 24 полосы не изменяются.

**3.3 *KECCAK-p*[*b, nr*]**

Даны матрица состояния **A** и индекс раунда ***ir***, *функция раунда* Rnd – это трансформация, являющая результатом последовательности пошаговых отображений **θ**, **ρ**, **π**, **χ** и **ι** в следующем порядке:

Rnd(**A**, *ir*) = **ι**(**χ**(**π**(**ρ**(**θ**(**A**), *ir*).

Перестановка KECCAK-*p*[*b*, *nr*], состоящая из *nr* итераций функции Rnd, определена в Алгоритме 7.

АЛГОРИТМ 7: KECCAK-*p*[*b*, *nr*](*S*)

Входные данные:

Строка ***S*** длины *b*;

Количество раундов ***nr***.

Выходные данные:

Строка ***S*ʹ** длины *b*.

Шаги алгоритма:

1. Конвертировать ***S*** в матрицу состояния **A**, как описано в Разделе 3.1.2.

2. Для *ir* от 12 + 2*l* – *nr* до 12 + 2*l* – 1 пусть **A** = Rnd(**A**, *ir*).

3. Конвертировать **A** в строку ***S*ʹ** длины b, как описано в Разделе 3.1.3.

4. Вернуть S**ʹ**.

**3.4 Сравнение с *KECCAK-f***

Семейство перестановок KECCAK-*f*, первоначально определенное в [8], представляет собой специализацию семейства KECCAK-*p* для случая *nr* = 12 + 2*l*:

KECCAK-*f*[b] = KECCAK-*p*[*b*, 12 + 2*l*].

Таким образом, перестановка KECCAK-*p*[1600, 24], лежащая в основе шести SHA-3 функций, эквивалентна KECCAK-*f*[1600].

Раунды в KECCAK-*f*[*b*] индексируются от 0 до 11 + 2l (12 + 2*l* – 1). В результате индексации на шаге 2 Алгоритма 7 раунды KECCAK-*p*[*b*, *nr*] совпадают с последними раундами KECCAK-*f*[*b*] и наоборот. Например, KECCAK-*p*[1600, 19] эквивалентна последним 19-ти раундам KECCAK-*f*[1600]. Точно так же KECCAK-*f*[1600] эквивалентна последним двадцати четырём раундам KECCAK-*p*[1600, 30]; в этом случае предыдущие раунды для KECCAK-*p*[1600, 30] индексируются целыми числами от –6 до –1.

**4. Конструкция «губки» (Sponge Construction)**

Конструкция «губки» [4] является основой (framework) для определения функций над двоичными данными с произвольной длинной выходных данных. В конструкции используются следующие три компонента:

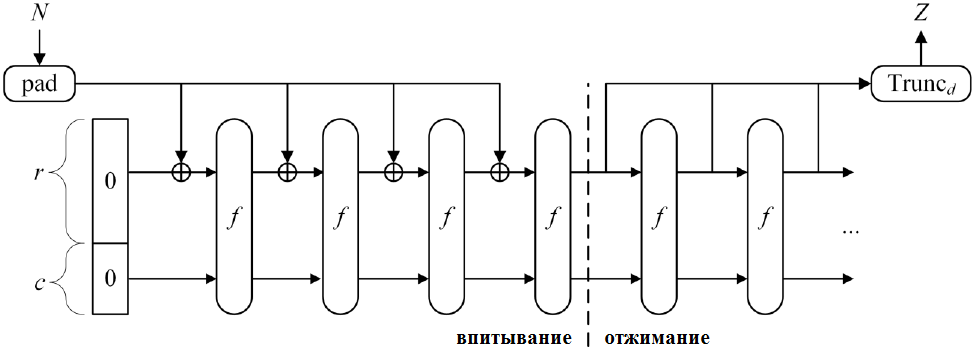
• базовая функция для строк фиксированной длины, обозначаемая ***f*** ;

• параметр, называемый скоростью и обозначаемый ***r*** ;

• правило заполнения, обозначаемое **pad**.

Функция, которую производит конструкция из указанных компонентов, называется ***функцией-«губкой»*** (***sponge function***) и обозначается как SPONGE[*f*,pad,*r*]. Функция-губка принимает два входных параметра: битовую строку (*N)* и битовую длину выходной строки (*d*) – SPONGE[*f*,pad,*r*](*N*, *d*). Аналогия с губкой состоит в том, что произвольное количество входных бит как бы «впитывается» (*absorbed*) в состояние функции, после чего произвольное количество выходных бит как бы «отжимается» (*squeezed*) из её состояния.

Конструкция «губки» проиллюстрирована на Рисунке 7 ниже (адаптировано из [4]).



**Рисунок 7:** Конструкция «губки»: *Z*=SPONGE[*f*, pad, *r*](*N*, *d*) [4]

Функция ***f*** отображает строки одной фиксированной длины *b* в строки той же длины. Также как и Разделе 3, *b* называется *шириной* *f*. SHA-3 функции, определенные в Разделе 6, являются примерами конструкции «губки», в которых базовая функция *f* обратима, т.е. является перестановкой, хотя конструкция губки этого не требует.

Скорость *r* – это положительное целое число, строго меньшее ширины *b*. *Емкость с* – это положительное целое определяемое как *b – r*. Таким образом,

*r + c = b*.

Правило заполнения pad представляет собой функцию, которая производит заполнение, т.е. строку соответствующей длины для добавления к другой строке. В общем случае, если задано положительное целое *x* и неотрицательное целое *m*, выходное значение pad(*x, m*) представляет собой строку, для которой *m*+len(pad(*x*,*m*)) положительно кратно *x* (*positive multiple of x*). В конструкции «губки» *x = r* и *m* = len(*N*), так что дополненная входная строка может быть разбита на последовательность *r*-битных строк. Алгоритм 9 в Разделе 5.1 определяет правило заполнения для функций KECCAK и, следовательно, для функций SHA-3.

Учитывая эти три компонента *f*, pad и *r*, как описано выше, функция SPONGE[*f*, pad, *r*] на (*N*, *d*) задается Алгоритмом 8. Ширина *b* определяется выбором *f*.

АЛГОРИТМ 8: SPONGE[*f*, pad, *r*](*N*, *d*)

Входные данные:

Строка ***N***;

Неотрицательное целое *d*.

Выходные данные:

Строка ***Z*** такая, что len(*Z*)=*d*.

Шаги алгоритма:

1. Пусть *P* = *N* || pad(*r*, len(*N*)).

2. Пусть *n* = len(*P*)/*r*.

3. Пусть *c* = *b* – *r*.

4. Пусть *P0*, … , *Pn-1* – уникальная последовательность строк длины *r* такая, что *P* = *P0* || … || *Pn-1*.

5. Пусть *S* = 0*b*.

6. Для *i* от 0 до *n*–1, пусть *S* = *f*(*S* ⨁ (*Pi* || 0*c*)).

7. Пусть *Z* – пустая строка.

8. Пусть *Z* = *Z* || Trunc*r*(*S*).

9. Если *d* ≤ |*Z*|, тогда вернуть Truncd(*Z*); иначе продолжаем.

10. Пусть *S = f*(*S*), и продолжаем с Шага 8.

Обратите внимание, что входное *d* определяет количество бит, возвращаемое Алгоритмом 8, но не влияет на их значения. В принципе, выходные данные могут рассматриваться как бесконечная строка, вычисление которой на практике останавливается после получения желаемого количества выходных бит.

**5. *KECCAK***

KECCAK – это семейство функций-«губок», первоначально определенное в [8]. Правило заполнения для KECCAK, называемое *мультискоростным заполнением* (***multi-rate padding***), определено в Разделе 5.1. Параметры и базовые перестановки для KECCAK описаны в Разделе 5.2. Меньшее семейство функций KECCAK, KECCAK[*c*], явно определено. Этого будет достаточно для определения функций SHA-3 в Разделе 6.

**5.1 Спецификация *pad10\*1***

АЛГОРИТМ 9: pad10\*1(*x*, *m*)

Входные данные:

Положительное целое ***x***;

Неотрицательное целое ***m***.

Выходные данные:

Строка ***P*** такая, что m + len(*P*) является положительным кратным *x*.

Шаги алгоритма:

1. Пусть *j* = (–*m* – 2) mod *x*.

2. Вернуть *P* = 1 || 0*j* || 1.

Таким образом, знак «\*» в обозначении “pad10\*1” указывает на то, что бит «0» либо опускается, либо повторяется по мере необходимости для получения выходной строки желаемой длины.

**5.2 Спецификация *KECCAK[c]***

KECCAK – это семейство функций «губок» с перестановкой KECCAK-*p*[*b*, 12+2*l*] (определенной в Разделе 3.3) в качестве базовой функции и правилом pad10\*1 (определенном в Разделе 5.1) в качестве правила заполнения. Семейство параметризуется любым выбором скорости *r* и емкости *c* такими, что *r* + *c* находится в диапазоне {25, 50, 100, 200, 400, 800, 1600}, т.е. берётся одно из семи значений *b* в Таблице 1.

Для случая *b* = 1600, семейство KECCAK обозначается как KECCAK[*c*]. В этом случае *r* определяется выбором *c*. В частности:

KECCAK[c] = SPONGE[KECCAK-*p*[1600, 24], pad10\*1, 1600 – *c*].

Таким образом, учитывая входную битовую строку *N* и длину выходных данных *d*:

KECCAK[c](*N*, *d*) = SPONGE[KECCAK-*p*[1600, 24], pad10\*1, 1600 – *c*](*N*, *d*).

**6. SHA-3 Спецификации функции**

В Разделе 6.1 определены четыре хеш-функции SHA-3, а в Разделе 6.2 – две функции SHA-3 XOF. В Разделе 6.3 альтернативное определение каждой функции SHA-3 XOF дается в терминах промежуточной функции.

**6.1 SHA-3 хеш-функции**

Для данного сообщения *M* четыре хеш-функции SHA-3 определяются из функции KECCAK[*c*], указанной в Разделе 5.2, путем добавления двухбитового суффикса к *M* и указания длины выходных данных следующим образом:

SHA3-224(M) = KECCAK[448](M || 01, 224);

SHA3-256(M) = KECCAK[512](M || 01, 256);

SHA3-384(M) = KECCAK[768](M || 01, 384);

SHA3-512(M) = KECCAK[1024](M || 01, 512).

В каждом случае ёмкость соответствует удвоенной длине дайджеста, т.е. *c*=2*d*, а результирующий ввод *N* для KECCAK[*c*] является сообщением с добавленным суффиксом, т.е. *N* = *M* || 01. Суффикс поддерживает разделение доменов (**domain separation**); т.е. он позволяет различать входные данные для KECCAK[*c*], возникающие из хеш функций SHA-3, и входные данные, возникающие из функций XOF SHA-3 (определенны в Разделе 6.2), а также другие домены, которые могут быть определены в будущем.

**6.2 SHA-3 функции расширенного вывода**

Для данного сообщения *M* две функции расширенного вывода SHA-3 XOFs, SHAKE128 и SHAKE256, определены из функции KECCAK[*c*] (указанной в Разделе 5.2) путем добавления четырёхбитного суффикса к *M* для любой длины *d* выходных данных:

SHAKE128(*M*, *d*) = KECCAK[256](*M* || 1111, *d*),

SHAKE256(*M*, *d*) = KECCAK[512](*M* || 1111, *d*).

Цели добавления дополнительных бит 1111 обсуждаются в Разделе 6.3.

**6.3 Альтернативные определения SHA-3 функций расширенного вывода**

Две дополнительные функции «губки», называемые RawSHAKE128 и RawSHAKE256, определены как следующие экземпляры KECCAK[*c*] (где входная строка обозначена *J*, а длина выходной строки обозначена *d*):

RawSHAKE128(*J*, *d*) = KECCAK[256](*J* || 11, *d*),

RawSHAKE256(*J*, *d*) = KECCAK[512](*J* || 11, *d*).

Эти функции позволяют дать альтернативное определение SHAKE128 и SHAKE256 в качестве эквивалента определению из Раздела 6.2. В частности:

SHAKE128(*M*, *d*) = RawSHAKE128(*M* || 11, *d*),

SHAKE256(*M*, *d*) = RawSHAKE256(*M* || 11, *d*).

Следующий вход *N* для KECCAK[*c*] является результатом заполнения этих входов *M* и *J*:

*N* = *J* || **11** = *M* || *11* || **11**.

Суффикс, выделенный жирным шрифтом (**11**), поддерживает разделение доменов: он отличает вход для KECCAK[*c*], возникающий из RawSHAKE128 и RawSHAKE256 от входа, возникающего из хеш-функций SHA-3 (определенных в Разделе 6.1), а также от других доменов, которые могут быть определены в будущем.

Суффикс, выделенный курсивом (*11*), обеспечивает совместимость RawSHAKE128 и RawSHAKE256 со схемой шифрования «Сакура» [6]. Эта схема облегчит будущую разработку вариантов хеширования дерева [7], в которых для более эффективного вычисления и обновления дайджеста длинных сообщений может быть применена параллельная обработка.

Следует обратить внимание на то, что при выполнении KECCAK[*c*] к *N* присоединяются дополнительные биты, как указано в правиле мультискоростного заполнения.

**7. Соответствие стандарту (Conformance)**

Реализация перестановки KECCAK-*p*[1600, 24] и шести модов SHA-3 этой перестановки – SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128 и SHAKE256 – может быть протестирована на соответствие настоящему Стандарту под эгидой (*under the auspices*) Программы проверки криптографических алгоритмов (***Cryptographic Algorithm Validation Program***) [9].

SHA3-224, SHA3-256, SHA3-384 and SHA3-512 являются утвержденными криптографическими хеш-функциями. Одно из утвержденных применений криптографических хеш-функций происходит в коде аутентификации сообщения с хеш-ключом (**HMAC** – ***Keyed-Hash Message Authentication Code***). Размер входного блока в байтах (обозначенный *B*) в спецификации HMAC [10] для SHA-3 хеш-функций[[5]](#footnote-6) приведен ниже в Таблице 3:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Хеш-функция | SHA3-224 | SHA3-256 | SHA3-384 | SHA3-512 |
| Размер блока (в байтах) | 144 | 136 | 104 | 72 |

**Таблица 3:** Размер входного блока для HMAC

SHAKE128 и SHAKE256 являются утвержденными XOFs, разрешенное использование которых будет определено в специальных публикациях NIST. Хотя некоторые из этих применений могут пересекаться с использованием утвержденных хеш-функций, функции XOFs *НЕ УТВЕРЖДЕНЫ* в качестве хеш-функций, из-за свойства, которое обсуждается в Разделе A.2.

Перестановка KECCAK-*p*[1600, 24] одобрена для использования в контексте утвержденных режимов работы, таких как функции SHA-3. Точно так же другие промежуточные функции, определенные в настоящем Стандарте – например, KECCAK[*c*], RawSHAKE128, RawSHAKE256 – утверждаются в контексте одобренного режима работы базовой перестановки KECCAK-*p*.

Перестановка KECCAK-*p*[1600, 24] может быть одобрена для других целей. Другие перестановки KECCAK-*p* также могут быть одобрены, если какие-либо режимы работы для них разработаны и утверждены в рамках публикаций FIPS или специальных публикаций NIST.

Функции SHA-3 определены для сообщений любой длины в битах, включая пустые строки. Соответствующая реализация функции SHA-3 может ограничивать набор поддерживаемых битовых длин для сообщений. Точно так же соответствующая реализация функций SHA-3 XOFs может ограничивать набор поддерживаемых значений длины вывода. В обоих случаях любые подобные ограничения могут повлиять на взаимодействие с другими реализациями.

Для каждой вычислительной процедуры, указанной в настоящем Стандарте, соответствующая реализация может заменить заданный набор шагов любым математически эквивалентным набором шагов. Другими словами, разрешены различные процедуры, которые производят корректный вывод для каждого ввода.

**A. Безопасность**

Подробный анализ свойств безопасности KECCAK в [8] относится к семейству хеш-функций SHA-3 и функций расширенного вывода. Семейство SHA-3 также наследует свойства безопасности от конструкции «губки»; эти свойства подробно проанализированы в [4].

Приложения хеш-функций часто требуют устойчивости к коллизиям, атаке прообраза и/или атаке второго прообраза; эти свойства обобщены для семейства хеш-функций SHA-3 и XOF в Разделе A.1. XOFs отличается от хеш-функций генерацией тесно связанных выходных данных (*in the generation of closely related outputs*); это важное соображение безопасности обсуждаются в Разделе A.2.

**A.1 Резюме**

На момент публикации настоящего стандарта степень безопасности функций SHA-3 представлены в Таблице 4. Функции SHA-1 и SHA-2 включены для сравнения, частично дублируя обсуждение в [11], которое будет обновляться по мере необходимости с учетом последней информации о безопасности.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Функции** | **Размер вывода** | **Степень безопасности (в битах)** | | |
| **Коллизия** | **Прообраз** | **2-й Прообраз** |
| SHA-1 | 160 | < 80 | 160 | 160 – *L*(*M*) |
| SHA-224 | 224 | 112 | 224 | *min*(224, 256 – *L*(*M*)) |
| SHA-512/224 | 224 | 112 | 224 | 224 |
| SHA-256 | 256 | 128 | 256 | 256 – *L*(*M*) |
| SHA-512/256 | 256 | 128 | 256 | 256 |
| SHA-384 | 384 | 192 | 384 | 384 |
| SHA-512 | 512 | 256 | 512 | 512 – *L*(*M*) |
| SHA3-224 | 224 | 112 | 224 | 224 |
| SHA3-256 | 256 | 128 | 256 | 256 |
| SHA3-384 | 384 | 192 | 384 | 384 |
| SHA3-512 | 512 | 256 | 512 | 512 |
| SHAKE128 | *d* | *min*(*d*/2, 128) | ≥ *min*(*d*, 128) | *min*(*d*, 128) |
| SHAKE256 | *d* | *min*(*d*/2, 256) | ≥ *min*(*d*, 256) | *min*(*d*, 256) |

**Таблица 4:** Степень безопасности SHA-1, SHA-2 и SHA-3 функций.

Степень безопасности против атак нахождения второго прообраза для сообщения *M* – функция *L*(*M*) определена как ⌈log2(len(*M*)/*B*)⌉, где *B* – это размер блока функции в битах, т.е. *B* = 512 для SHA-1, SHA-224, SHA-256 и *B* = 1024 для SHA-512.

Четыре функции SHA-3 являются альтернативами функциям SHA-2. Они разработаны для обеспечения сопротивления коллизиям, атаке прообраза и второго прообраза, которое эквивалентно или превышает сопротивление, предоставляемое соответствующими функциями SHA-2. Функции SHA-3 также разработаны для сопротивления другим атакам, таким как ***атака удлинением сообщения*** (***length-extension attacks***), которым будет противостоять случайная функция той же выходной длины, в целом обеспечивающая ту же степень безопасности, что и случайная функция, вплоть до выходной длины.

Две функции SHA-3 XOFs разработаны для сопротивления коллизии, атаке прообраза, второго прообраза и другим атакам, которым может противостоять случайная функция запрашиваемой выходной длины, вплоть до степени безопасности 128 бит для SHAKE128 и 256 бит для SHAKE256. Случайная функция, длина выходных данных которой *d*, не может обеспечить более чем *d*/2 бит безопасности против коллизионной атаки и *d* бит безопасности против атаки прообраза и второго прообраза. Следовательно, когда *d* достаточно мало, SHAKE128 и SHAKE256 будут предоставлять менее чем 128 и 256 бит безопасности соответственно (как указано в Таблице 4). Например, если *d* = 224, тогда SHAKE128 и SHAKE256 предоставляют 112 бит стойкости к коллизиям; однако, они предоставляют различную степень стойкости к атакам прообраза: 128 бит для SHAKE128 и 224 бит для SHAKE256.

Если *d* > *r* + *c*/2, тогда SHAKE128 и SHAKE256 обеспечивают более чем 128 и 256 бит стойкости к атаке прообраза соответственно; более того, если *d* > 1600, вероятного прообраза не существует.

**A.2 Дополнительные соображения о функциях расширенного вывода**

Функции XOF – это новый мощный тип ***криптографического примитива***, обеспечивающий гибкость для создания выходных данных любой желаемой длины. Технически, возможно использовать функцию XOF как хеш-функцию, выбрав фиксированную длину вывода. Однако, XOFs имеют возможность для генерации связанных выходных данных – свойство, которое разработчики приложений/протоколов/систем безопасности могут не ожидать от хеш-функций. Это свойство важно учитывать при разработке приложений XOFs.

По замыслу длина вывода для XOF не влияет на биты, которые он производит, а это означает, что длина вывода не обязательно является входом для функции. Концептуально, выходные данные могут быть бесконечной строкой, и приложения/протоколы/системы, вызывающие функцию, просто вычисляют желаемое количество начальных бит этой строки. Следовательно, когда для общего сообщения выбираются две разных длины вывода, эти два вывода тесно связаны: более длинный вывод является расширением более короткого вывода. Например, для заданных положительных целых *d* и *e*, и любого сообщения *M*: Trunc*d*(SHAKE128(*M*, *d*+*e*) идентично SHAKE128(*M*, *d*). Это же свойство применимо и для SHAKE256.

Нельзя ожидать, что две различные функции SHA-3 когда-либо проявят это свойство на практике. Например, для случайно выбранного сообщения *M*, SHA3-256(*M*) почти наверняка не будет расширением SHA3-224(M) или SHAKE128(*M*,224), несмотря на то, что эти три функции имеют почти идентичную структуру. То же утверждение относится к утвержденным ранее хеш-функциям, включая усеченные версии (truncated versions) SHA-512 в FIPS 180-4 (например, SHA-512/256).

Однако, существующие механизмы для конструирования функций с произвольной длиной выходных данных – путём конкатенации и/или усечения дайджеста хеш-функции – обычно проявляют это свойство.

Возможность существования тесно связанных выходных данных может повлиять на безопасность приложения/протокола/системы, в которых вызывается XOF. Например, наивным и не утвержденным (*naive and non-approved*) способом для двух сторон договориться о получении 112-битного ключа Triple DES из сообщения, обозначенного как *keymaterial*, будет вычисление SHAKE128(*keymaterial*, *keylength*), где *keylength* – длина ключа равная 112. Однако, если злоумышленник сможет заставить одну из сторон использовать другое значение *keylength*, скажем 168 бит, при том же самом значении *keymaterial*, то обе стороны получат следующие ключи:

SHAKE128(*keymaterial*, 112) = **fg**

SHAKE128(*keymaterial*, 168) = **fgh**,

где выделенные жирным шрифтом буквы дайджеста представляют 56-ти битную строку, например, части ключа Triple DES. Из-за структуры Triple DES эти ключи уязвимы для атак.

На практике использование функции XOF в качестве ***функции формирования ключа*** (***KDF, key derivation function***) может исключить возможность связанных выходных данных за счет включения длины и/или типа полученного ключа в сообщение на входе KDF. В этом случае разногласие или непонимание между двумя пользователями KDF относительно типа или длины получаемого ключа почти наверняка не приведет к связанным результатам.

Там, где расширенные дайджесты проблематичны, более общим решением является разделение доменов, с помощью которого можно создавать разные экземпляры XOF и настраивать их для разных целей, независимо от количества выходных бит. Все функции SHA-3 предназначены для создания вариантов для новых разделяемых доменов, которые NIST может разработать в будущем.

**B. Примеры**

Примеры пяти пошаговых отображений и шести функций SHA-3 доступны на странице примеров **NIST's Computer Security Resource Center** по адресу <http://csrc.nist.gov/groups/ST/toolkit/examples.html>.

Битовая строка для этих примеров представлена в шестнадцатеричном виде, т.е. последовательности из шестнадцати шестнадцатеричных цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E и F, где A – 10, B – 11 и т.д.

Соглашение об интерпретации шестнадцатеричных строк в виде битовых строк для входных и выходных данных примеров SHA-3 отличается от соглашения для других функций на странице примеров. Функции конвертации между шестнадцатеричными строками и битовыми строками SHA-3 определены в Разделе B.1. Для сообщений с выравниванием по байтам шестнадцатеричные формы заполнения для SHA-3 функций описаны в Разделе B.2. Специфические шестнадцатеричные строки и этих разделах выделены шрифтом Courier New, которому предшествует маркер 0x.

**B.1 Функции преобразования**

Функции преобразования из шестнадцатеричных строк в строки SHA-3, которые их представляют, обозначаются ***h2b*** и определены в Алгоритме 10. Функции определены для шестнадцатеричных строк с четным количеством цифр.

АЛГОРИТМ 10: ***h2b***(*H*, *n*)

Входные данные:

Шестнадцатеричная строка *H*, состоящая из 2*m* цифр для некоторого положительного целого ***m***;

Положительное целое ***n***, такое что *n* ≤ 8*m*.

Выходные данные:

Битовая строка ***S*** такая, что len(*S*)=*n*.

Шаги алгоритма:

1. Для каждого целого i такого, что 0 ≤ *i* < 2*m*-1, пусть *Hi* – i-ая 16-теричная цифра в H: *H* = *H0* *H1* *H2* *H3*… *H2m-2* *H2m-1*.

2. Для каждого целого i такого, что 0 ≤ *i* < *m*.

a. Пусть *hi* = 16 • *H2i* + *H2i+1*.

b. Пусть *bi0 bi1 bi2 bi3 bi4 bi5 bi6 bi7* – уникальная последовательно бит таких, что:

*hi* = *bi7* • 27 + *bi6* • 26 + *bi5* • 25 + *bi4* • 24 + *bi3* • 23 + *bi2* • 22 + *bi1* • 21 + *bi0* • 20.

3. Для каждой пары целых (*i*, *j*) таких, что 0 ≤ *i* < *m* и 0 ≤ *j* < 8, пусть

*T*[8*i* + *j*] = *bi j*.

4. Вернуть *S* = Trunc*n*(*T*).

На Шаге 1 индексы определены как шестнадцатеричные цифры. На Шаге 2a каждая пара шестнадцатеричных цифр преобразуется в целое число (по основанию 10) от 0 до 255, которое представляет пара по основанию 16. На Шаге 2b каждое целое число из Шага 2a преобразуется в его двоичное представление как байта. На Шаге 3 байты составляются в отдельную строку. На Шаге 4 результат усекается до желаемого количества бит.

Например, если *H* = 0xA32E и *n* = 14, то промежуточное значение, определенное в Алгоритме 10 на Шаге 1 и 2 продемонстрировано в Таблице 5 ниже:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *H0* = A | | | | *H1* = 3 | | | | *H2* = 2 | | | | *H3* = E | | | |
| *h0* = 16 • 10 + 3 = 163 =  1 • 27 + 0 • 26 + 1 • 25 + 0 • 24 + 0 • 23 +  0 • 22 + 1 • 21 + 1 • 20 | | | | | | | | *h1* = 16 • 2+ 14 = 46 =  0 • 27 + 0 • 26 + 1 • 25 + 0 • 24 + 1 • 23 +  1 • 22 + 1 • 21 + 0 • 20 | | | | | | | |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| *b07* | *b06* | *b05* | *b04* | *b03* | *b02* | *b01* | *b00* | *b17* | *b16* | *b15* | *b14* | *b13* | *b12* | *b11* | *b10* |

**Таблица 5:** Иллюстрация ***h2b***.

Таким образом,

*T* = *b*00*b*01*b*02*b*03*b*04*b*05*b*06*b*07*b*10*b*11*b*12*b*13*b*14*b*15*b*16*b*17 = 1100 0101 0111 0100,

и, как описано в Шагах 3 и 4:

*S* = Trunc14(*T*) = 1100 0101 0111 01

Если длина *n* выходной строки *S* не определена явно в качестве входных данных, тогда ***h2b***(*H*) определяется как ***h2b***(*H*, 8*m*), т.е. берётся максимально возможное значение *n*.

Функция конвертации из битовых строк SHA-3 в шестнадцатеричные строки, представляющие их, обозначается ***b2h***, и определена в Алгоритме 11.

АЛГОРИТМ 11: ***b2h***(*S*)

Входные данные:

Битовая строка ***S***, состоящая из ***n*** бит для положительного целого ***n***.

Выходные данные:

Шестнадцатеричная строка *H*, состоящая из 2⌈*n*/8⌉ цифр.

Шаги алгоритма:

1. Пусть *n* = len(*S*).

2. Пусть *T* = *S* || 0- n mod 8 и *m* = ⌈*n*/8⌉.

3. Для каждой пары целых (*i*, *j*) таких, что 0 ≤ *i* < *m* и 0 ≤ *j* < 8, пусть

*bi j* = *T*[8*i* + *j*].

4. Для каждого целого i такого, что 0 ≤ *i* < *m*.

a. Пусть:

*hi* = *bi7* • 27 + *bi6* • 26 + *bi5* • 25 + *bi4* • 24 + *bi3* • 23 + *bi2* • 22 + *bi1* • 21 + *bi0* • 20.

b. Пусть H2i и H2i+1 – шестнадцатеричные цифры такие, что

*hi* = 16 • *H2i* + *H2i+1*

5. Вернуть *H* = *H0* *H1* *H2* *H3*… *H2m-2* *H2m-1*.

Формальная функция переупорядочивания битов, указанная в [12] – для представления KECCAK на конкурс SHA-3 – дает эквивалентные преобразования, когда сообщение выровнено по байтам, т.е. когда *n* кратно 8.

**B.2 Шестнадцатеричная форма для заполняющих битов (padding bits)**

Для функций SHA-3 к сообщению *M* добавляется двух- или четырехбитный суффикс для создания входной строки *N* для KECCAK[*c*], а дополнительные биты присоединяются как часть правила мультискоростного заполнения.

Для большинства приложений сообщение выравнивается по байтам, т.е. len(*M*) = 8*m* для неотрицательного целого *m*. В этом случае общее количество байт, добавляемых к сообщению (обозначаемое ***q***), определяется через *m* и скорость *r* следующим образом:

*q* = (*r*/8) – (*m* mod (*r*/8)).

Значение *q* определяет шестнадцатеричный вид этих байт, в данном случае в соответствии с функциями преобразования, указанными в Разделе B.1. Получаемые в результате дополненные сообщения сведены в Таблицу 6:

|  |  |  |
| --- | --- | --- |
| Тип функции SHA-3 | Количество байт заполнения | Дополненное сообщение |
| Hash | *q* = 1 | *M* || 0x86  🡪 0110 0001 |
| Hash | *q* = 2 | *M* || 0x0680  🡪 0110 0000 0000 0001 |
| Hash | *q* > 2 | *M* || 0x06 || 0x00… || 0x80  🡪 0110 0000 … … 000 0001 |
| XOF | *q* = 1 | *M* || 0x9F  🡪 1111 1001 |
| XOF | *q* = 2 | *M* || 0x1F80  🡪 1111 1000 0000 0001 |
| XOF | *q* > 2 | *M* || 0x1F || 0x00… || 0x80  🡪 1111 1000 … … 0000 0001 |

**Таблица 6:** Шестнадцатеричная форма SHA-3 заполнения для сообщений с

байтовым выравниванием

В Таблице 6 обозначение «0x00…» указывает на строку, которая состоит из *q*–2 нулевых байт.

Символ «🡪» показывает направление байт (от младшего к старшему), жёлтым цветом выделен битовый суффикс, добавляемый к сообщению (10 – для функций SHA-3, 1111 – для XOFs), голубым цветом выделены биты заполнения.

**C. Объектные идентификаторы**

Объектные идентификаторы (OIDs) для SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128 и SHAKE256 опубликованы в <http://csrc.nist.gov/groups/ST/crypto_apps_infra/csor/algorithms.html>.

**D. Использованная литература**

[1] Federal Information Processing Standards Publication 180-4, *Secure Hash Standard (SHS)*, Information Technology Laboratory, National Institute of Standards and Technology, March 2012, <http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>.

[2] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, *The KECCAK SHA-3 submission, Version 3*, January 2011,

<http://keccak.noekeon.org/Keccak-submission-3.pdf>.

[3] The SHA-3 Cryptographic Has Algorithm Competition, November 2007 – October 2012, <http://csrc.nist.gov/groups/ST/hash/sha-3/index.html>.

[4] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, *Cryptographic sponge functions*, January 2011, <http://sponge.noekeon.org/CSF-0.1.pdf>.

[5] Ethan Heilman to [hash-forum@nist.gov](mailto:hash-forum@nist.gov), October 5, 2012, Hash Forum,

<http://csrc.nist.gov/groups/ST/hash/email_list.html>.

[6] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, *SAKURA: a flexible coding for tree hashing*, <http://keccak.noekeon.org/Sakura.pdf>.

[7] R. C. Merkle, *A digital signature based on a conventional encryption function*, Advances in Cryptology – CRYPTO ’87, A Conference on the Theory Applications of Cryptographic Techniques, Santa Barbara, California, USA, 1987, 369-378.

[8] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, *The KECCAK reference, Version 3.0*, January 2011, <http://keccak.noekeon.org/Keccak-reference-3.0.pdf>.

[9] NIST Cryptographic Algorithm Validation Program (CAVP),

<http://csrc.nist.gov/groups/STM/cavp/index.html>.

[10] Federal Information Processing Standards Publication 198-1, *The Keyed-Hash Message Authentication Code (HMAC)*, Information Technology Laboratory, National Institute of Standards and Technology, July 2008, <http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf>.

[11] NIST Special Publication 800-107 Revision 1: *Recommendation for Using Approved Hash Algorithm*, August 2012, <http://csrc.nist.gov/publications/nistpubs/800-107-rev1/sp800-107-rev1.pdf>.

[12] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer, *KECCAK implementation overview*, January 2011,

<http://keccak.noekeon.org/Keccak-implementation-3.0.pdf>.

1. Точнее, конкурс требовал четыре хеш-функции, а KECCAK – это более широкое семейство функций. [↑](#footnote-ref-2)
2. Для многих хеш-функций, есть ограничение (довольно большое) на длину входных данных. [↑](#footnote-ref-3)
3. Имя «SHAKE» было предложено в [5] как комбинация терминов «**S**ecure **H**ash **A**lgorithm» и «**KE**CCAK». [↑](#footnote-ref-4)
4. За исключением случая, когда длина выходных данных достаточна мала; смотри обсуждение в секции A.1 [↑](#footnote-ref-5)
5. В общем, размер входного блока (в битах) функции «губки»– это её скорость. [↑](#footnote-ref-6)