**Федеральное Государственное Бюджетное**

**Образовательное Учреждение**

**Высшего Профессионального Образования**

**Национальный Исследовательский Университет «МЭИ»**

Институт Автоматики и Вычислительной Техники

Кафедра Прикладной Математики

**«Программная реализация функции хеширования SHA-3»**

**Курсовой проект**

по учебной дисциплине

«Защита данных»

Выполнил:

студент группы А-05-14

**Зубов Михаил Игоревич**

Преподаватель:

кандидат технических наук

доцент **Хорев Павел Борисович**

**Москва 2017**

Оглавление

[ВВЕДЕНИЕ 3](#_Toc501596439)

[1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 4](#_Toc501596440)

[1.1. Обзор стандарта SHA-3 4](#_Toc501596441)

[1.2. Алгоритм SHA-3 4](#_Toc501596442)

[1.2.1. Общие сведения. 5](#_Toc501596443)

[1.2.2. Состояние. 5](#_Toc501596444)

[1.2.3. Алгоритмы пошаговых отображений. 7](#_Toc501596445)

[1.2.4. Алгоритмы функций верхнего уровня 11](#_Toc501596446)

[1.3. Конструкция HMAC 13](#_Toc501596447)

[1.3.1. Алгоритм HMAC 13](#_Toc501596448)

[2. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 14](#_Toc501596449)

[2.1. Графический интерфейс. 14](#_Toc501596450)

[2.2. Реализация алгоритмов стандарта SHA-3. 19](#_Toc501596451)

[2.3. Реализация конструкции HMAC. 20](#_Toc501596452)

[2.4. Обработка парольной фразы. 20](#_Toc501596453)

[3. ТЕСТИРОВАНИЕ РАЗРАБОТАННОЙ ПРОГРАММЫ 21](#_Toc501596454)

[ЗАКЛЮЧЕНИЕ 27](#_Toc501596455)

[СПИСОК ИСТОЧНИКОВ 28](#_Toc501596456)

[Приложение 1. 29](#_Toc501596457)

[Файл Main.cs 29](#_Toc501596458)

[Файл KeySet.cs 38](#_Toc501596459)

[Файл KeyInput.cs 39](#_Toc501596460)

[Файл SHA3.cs 41](#_Toc501596461)

# ВВЕДЕНИЕ

Прогресс не стоит на месте. Вместе с общим технологическим развитием возрастают и вычислительные мощности, что в свою очередь накладывает все более жёсткие требования к криптостойкости алгоритмов хеширования и заставляет разрабатывать более совершенные средства защиты информации, в частности алгоритмы хеширования.

**Цель** работы – разработка приложения для хеширования текстов и файлов по относительно новому алгоритму SHA-3 с возможностью использования парольной фразы, функцией сохранения и проверки контрольного хеш-значения, регулировкой сложности парольной фразы.

Были поставлены и решены следующие **задачи**:

* проектирование и реализация графического интерфейса;
* проектирование логики взаимодействия компонентов программы;
* реализация основного алгоритма хеширования;
* реализация механизма HMAC;
* реализация механизма хеширования файлов;
* тестирование и отладка программы.

В **первой** главе будет приведена краткая справка о keccak, его основные преимущества и недостатки, описание алгоритмов функций SHA-3, описание конструкции HMAC.

Во **второй** главе будут представлены результаты проектирования программы и графического интерфейса, описание возможностей программы, описание созданных при реализации стандарта методов.

В **третьей** главе будут приведены результаты тестирования программы на предмет корректной работы алгоритмов хеширования, интерфейса и логики программы.

## ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

### Обзор стандарта SHA-3

2 ноября 2007 года NIST анонсировал конкурс на разработку нового стандарта хеширования, чтобы в случае необходимости была возможность быстро заменить SHA-2. 2 октября 2012 года алгоритм Keccak стал победителем конкурса, в ходе которого получил некоторые изменения (было увеличено количество раундов функции перестановки и изменён алгоритм дополнения входной строки), и был стандартизован как алгоритм SHA-3[[3](#Источник3)]. 5 августа 2015 года алгоритм утвержден и опубликован в качестве стандарта FIPS 202[[4](#Источник4)][[5](#Источник5)].

Алгоритм хеширования Keccak был разработан Гвидо Бертони, Йоаном Дайменом, Жилем Ван Аше из STMicroelectronics и Микаэлем Питерсом из NXP Semiconductors. Он является логическим продолжением более ранних хеш-функций PANAMA и RadioGatún[[1](#Источник1)] и работает на основе криптографической губки (sponge)[[2](#Источник2)].

Криптографическая губка представляет из себя функцию перестановок строки состояния фиксированной длины, правило заполнения этой строки входной строкой произвольной длины и правило получения из неё выходной строки требуемой длины[[2](#Источник2)].

Ключевыми отличиями криптографической губки от структуры Меркла — Дамгарда, на которой построен SHA-2, являются:

* использование на каждом шаге фазы «впитывания» сложения по модулю два строки состояния и очередного блока входного сообщения, вместо их слияния;
* использование функции перестановки, вместо функции сжатия;
* возможность генерации выходной строки с любой задаваемой длиной за счёт составления её из частей цепочки строк состояний, генерируемой после полного «впитывания» входной строки[[2](#Источник2)][10].

Недостатком SHA-3 является в среднем вдвое меньшая скорость работы программной реализации в сравнении с SHA-2[6][7].

Преимуществами же являются гораздо более быстрая работа аппаратной реализации[8][9] и большая криптографическая стойкость[[5](#Источник5)][7] по сравнения с SHA-2.

### Алгоритм SHA-3

Алгоритм состоит из пяти функций перестановок составляющих раунд преобразований над строкой состояния, функции проводящей заданное количество раундов над состоянием, функции губки и функции дополнения входной строки.

#### Общие сведения.

KECCAK-*p*[*b*, *nr*] -перестановки, используемые в стандарте, характеризуется двумя параметрами: 1) *b* - фиксированной длиной поданного на вход сообщения, называемой шириной перестановок, и 2) *nr* - количеством итерации внутренних преобразований, называемых раунд (*Rnd*). В стандарте представлены перестановки для всех *b*, входящих в множество {25, 50, 100, 200, 400, 800, 1600} и всех положительных целых значениях *nr*.

Каждый раунд перестановок включает в себя последовательность из пяти функций перестановок, называемых пошаговые отображения (*step mappings*). Перестановки заключаются в операциях над постоянно обновляемым по ходу работы алгоритма массивом размером *b* бит, называемым состояние (*state*).

В стандарте существуют ещё две важные величины, зависимые от ширины *b*: *w* – величина равная отношению *b* к 25 (*b*/25) и *l* – величина равная логарифму по основанию два от упомянутого отношения (). Семь возможных значений этих величин, которые определены стандартом приведены в таблице 1.1.

Таблица 1.1. Значения величин b, w, l.

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

Так как в стандарте используется только количество раундов *nr* = 12 + 2*l*, переопределим KECCAK-*f* [*b*] = KECCAK-*p*[*b*, 12 + 2*l*].

#### Состояние.

Удобно представлять входное и выходное состояние в виде битовой строки: , где – состояние каждого бита строки.

А внутреннее состояние в виде кубического массива размерностью 5 на 5 на *w*. – состояние каждого бита, где , , и . Части массива состояний для *w* = 8 представлены на рисунке 1.1.

Чтобы перевести строковое представление в массив, используется формула: .

Чтобы произвести обратное преобразование используется следующее преобразование:

для каждой пары : ,

,

для каждого *j*:

,

.

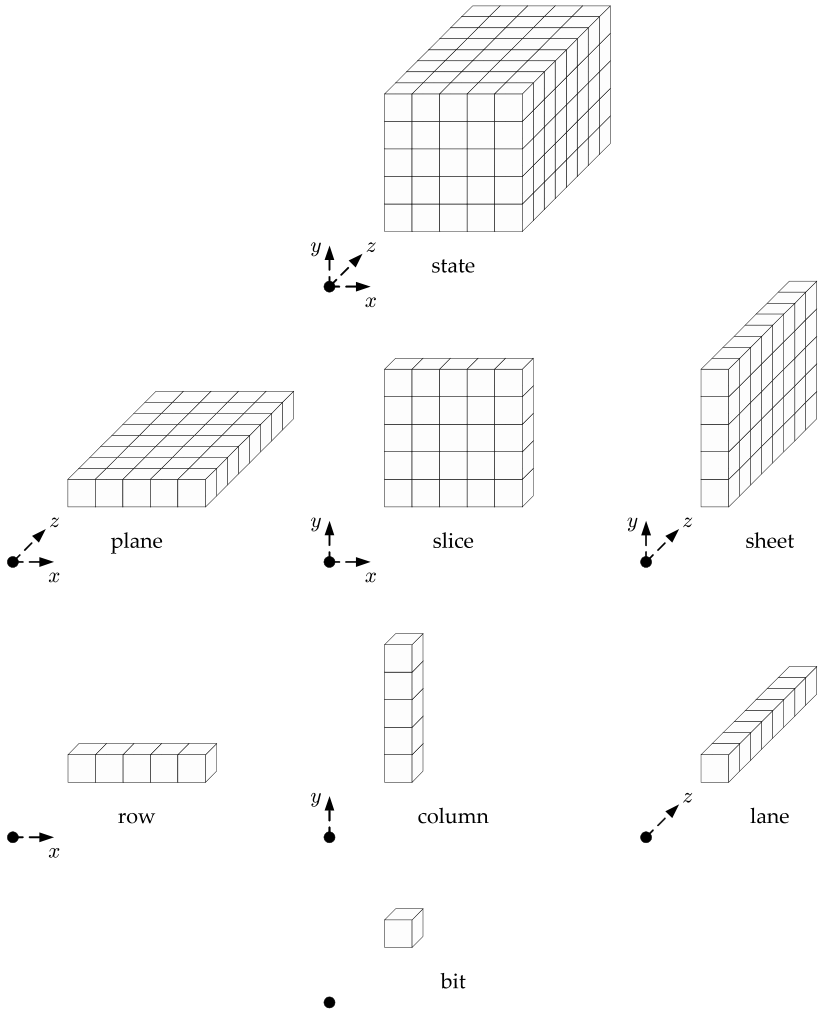


Рисунок.1.1. Части массива состояний для w = 8.

На диаграммах состояния, которые сопровождают спецификации пошаговых отображений, линия, соответствующая координатам (x, y) = (0, 0), изображена в центре срезов. Полное положение координат x, y и z для этих диаграмм показана на рисунке 1.2.

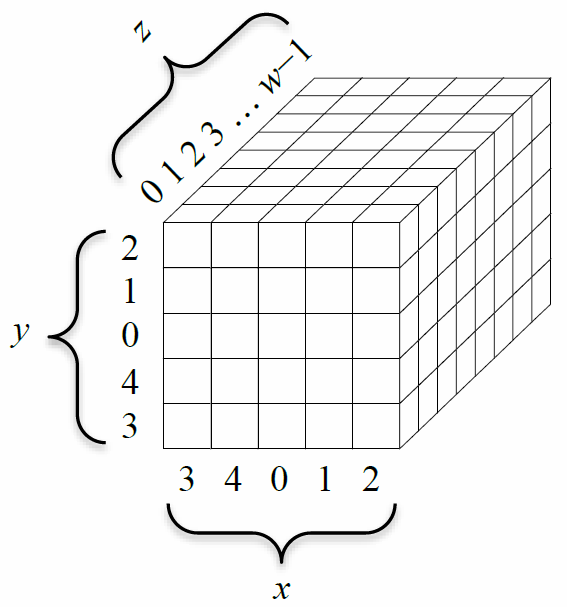
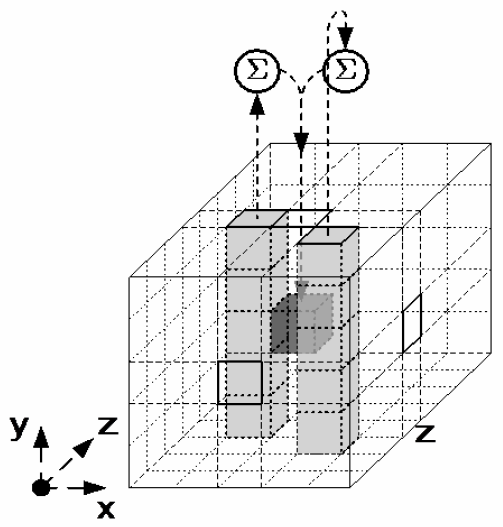


Рисунок.1.3. Результат работы функции θ(А).

Рисунок.1.2. Положение координат.

#### Алгоритмы пошаговых отображений.

Пять пошаговых отображений, составляющих раунд KECCAK-*p*[*b*, *nr*] обозначаются θ («тета»), ρ («ро»), π («пи»), χ («хи»), ι («йота»). Каждое из них в качестве входного параметра получает массив состояния А, и возвращает обновлённый массив состояний A'. Отображение ι имеет второй входной параметр – номер раунда *ir*.

Алгоритм 1: *θ*(А)

*Входные данные*: массив состояния А.

*Выходные данные*: обновленный массив состояния A'.

Шаг 1. Для каждой пары (x, z), и :

.

Шаг 2. Для каждой пары (x, z), и :

.

Шаг 3. Для каждой комбинации (x, y, z), , и :

.

В результате работы данной функции (Рис.1.3), с помощью оператора сложения по модулю 2 (XOR) каждый бит матрицы состояния складывается с суммами, полученными в результате той же операции, но над двумя столбцами, при чем, если координаты элемента , то сложение ведётся со столбцами, и .

Алгоритм 2: ρ(A)

Входные данные: массив состояния А.

Выходные данные: обновленный массив состояния A'.

Шаг 1. Для каждого , : .

Шаг 2. Принять .

Шаг 3. Для каждого , :

a) Для каждого , :

;

b) Принять .

Результатом вызова данной функции будет циклическая смена битов в каждой линии на величину, называемую смещением (offset), которое фиксировано зависит от каждой координаты , и, соответственно, может быть вычислено заранее. Или, что эквивалентно, каждый бит в линии смещается по координате на величину смещения по модулю (Рис.1.4).

Значения для каждой пары , приведены в таблице 1.2.

Таблица 1.2. Значения для каждой пары .

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | ***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 |

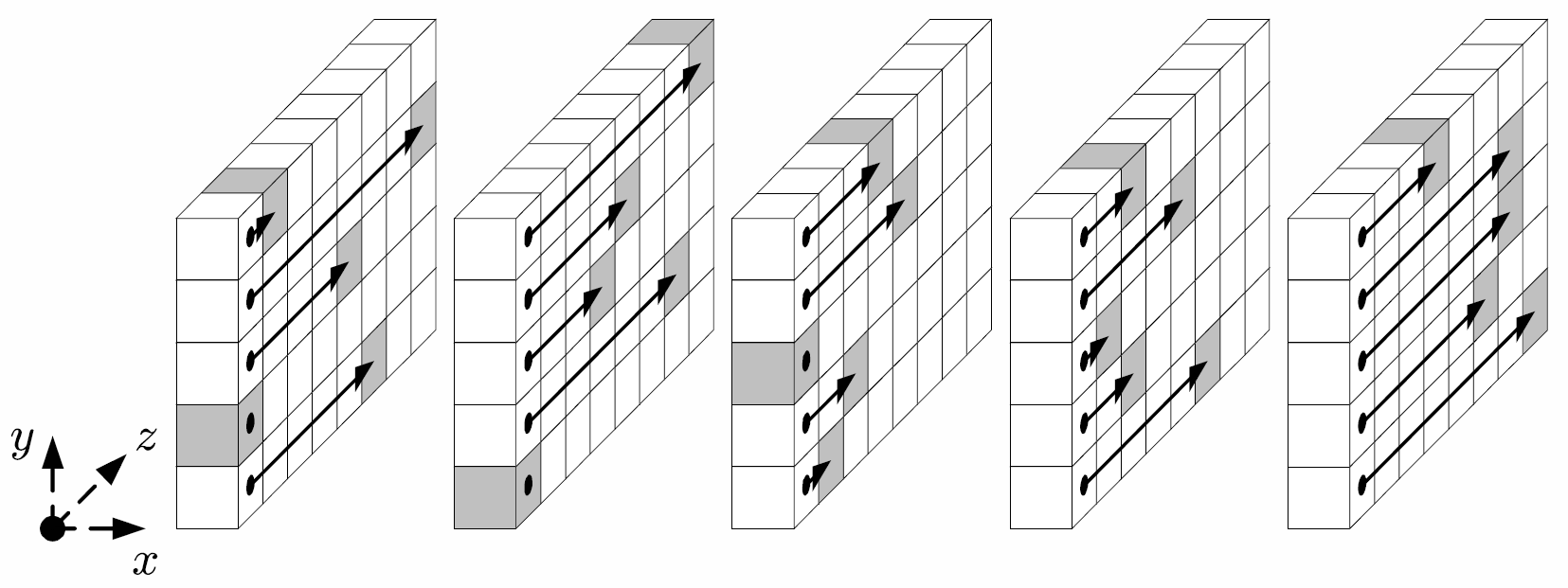


Рисунок 1.4. Результат работы функции ρ(А).

Алгоритм 3: π(А)

*Входные данные*: массив состояния А.

*Выходные данные*: Обновленный массив состояния A'.

Шаг 1. Для каждой комбинации (x, y, z), , и :

.

Результатом данной функции будут перестановки линий между собой, как показано на рисунке 1.5.

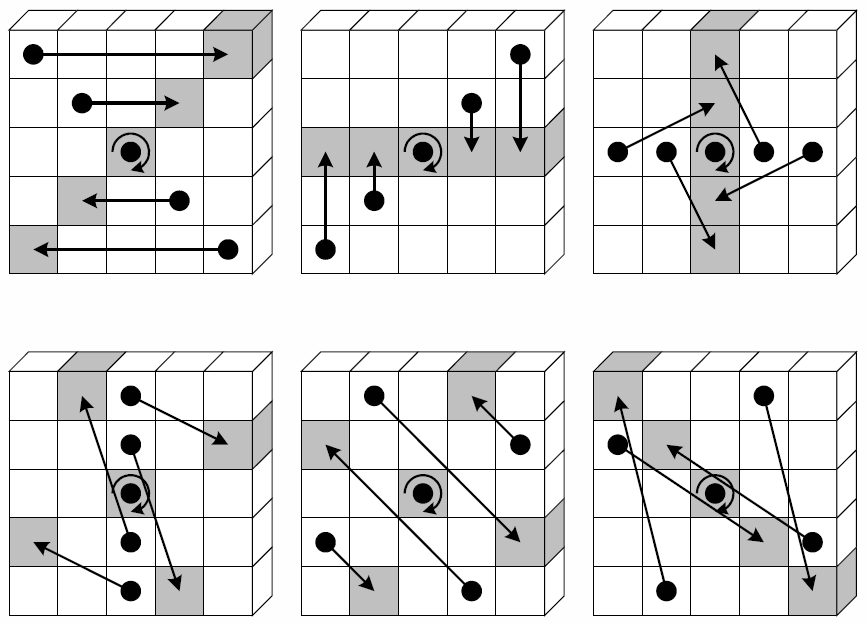


Рисунок 1.5. Результат работы функции π(А).

Алгоритм 4: χ(А)

*Входные данные*: массив состояния А.

*Выходные данные*: Обновленный массив состояния A'.

Шаг 1. Для каждой комбинации (x, y, z), , и :

Точка означает в данном случае эквивалент логической операции «И».

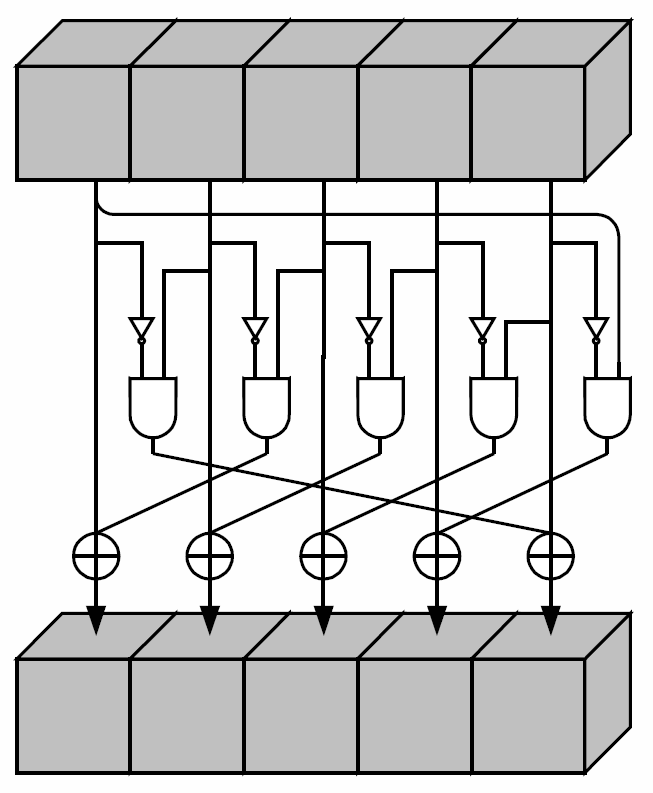
Результатом работы функции будет сложение по модулю два каждого бита с нелинейной функцией из двух других битов в ряду. Иллюстрация работы алгоритма представлена на рисунке 1.6.

Рисунок 1.6. Результат работы функции χ(А).

Алгоритм 5: *rc*(t)

Этот алгоритм является вспомогательным и генерирует круговую константу (RC), которые потребуется в алгоритме ι(А).

*Входные данные*: целое число t.

*Выходные данные*: битовое значение RC.

Шаг 1. Если , вернуть 1.

Шаг 2. Принять R=10000000.

Шаг 3. Для каждого , :

1. ;
2. ;
3. ;
4. ;
5. ;
6. .

Шаг 4. Вернуть .

Операция означает взятие первых символов строки .

Алгоритм 6: ι(А)

*Входные данные*: массив состояния А, номер раунда .

*Выходные данные*: Обновленный массив состояния A'.

Шаг 1. Для каждой комбинации (x, y, z), , и :

.

Шаг 2. Заполнить массив нулями: .

Шаг 3. Для каждого , :

.

Шаг 4. Для каждого , :

.

Результатом работы данной функции является модификация центральной линии (x=0, y=0) согласно формуле, зависящей от номера прохода. Остальные 24 линии участия в вычислениях не принимают и никак не меняются.

#### Алгоритмы функций верхнего уровня

Функция

.

Алгоритм 7: KECCAK-p[](S)

*Входные данные*: строка S, длины , количество раундов алгоритма .

*Выходные данные*: строка S', длины .

Шаг 1. Конвертировать строку S в массив состояния, как описано в п.1.1.2.

Шаг 2. Для каждого , : .

Шаг 3. Конвертировать массив состояния обратно в строку, как описано в п.1.1.2.

Алгоритм 8: pad10\*1()

*Входные данные*: размер обрабатываемой строки в битах – x, размер входного массива данных – m.

*Выходные данные*: строка P такая, что сумма , кратна x.

Шаг 1. Принять .

Шаг 2. Вернуть .

Знак «звездочка» в названии функции показывает, что «0» бит может быть опущен или добавлен необходимое количество раз, чтобы создать выходную строку желаемой длины.

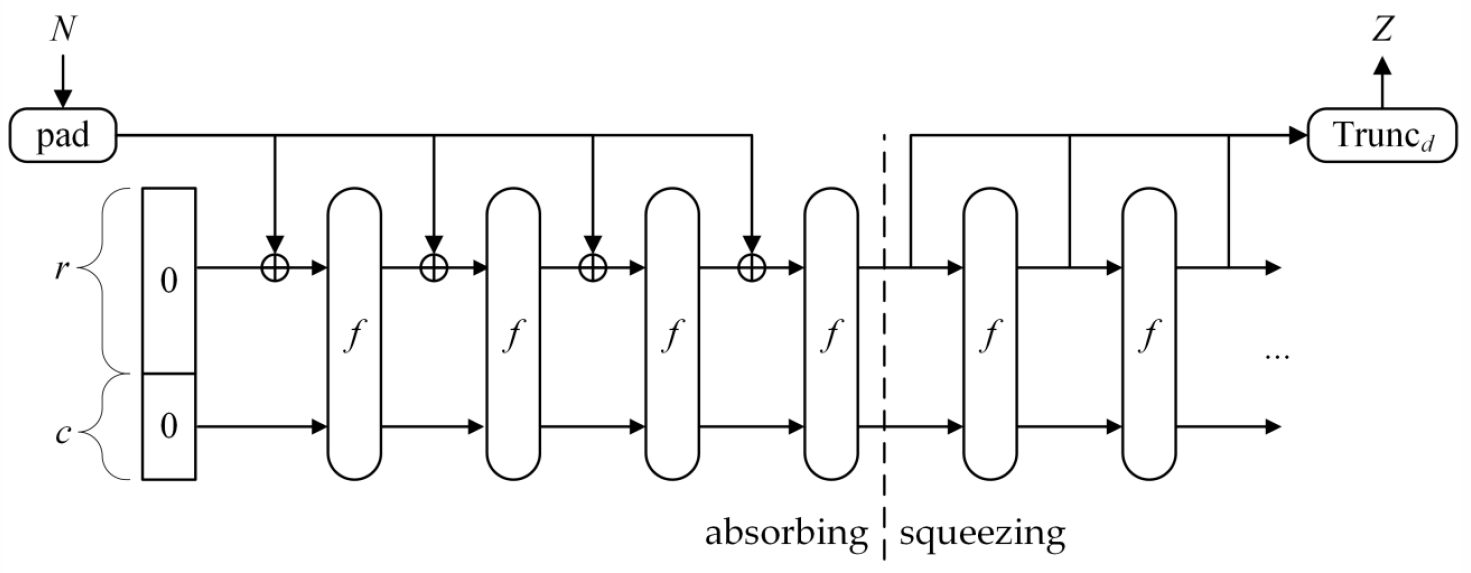


Рисунок 1.7. Работа функции SPONGE.

Алгоритм 9: SPONGE[f, pad, r](N,d)

*Входные данные*: битовая строка N, неотрицательная целая длина выходной строки d.

*Выходные данные*: строка S длины d .

Шаг 1. Расширить входное сообщение до достижения кратности размеру блока:

.

Шаг 2. Вычислить количество блоков во входном сообщении: .

Шаг 3. Вычислить вспомогательную переменную: .

Шаг 4. Представить строку P последовательностью подстрок длины r:

.

Шаг 5. Заполнить строку нулями: .

Шаг 6. Фаза «впитывания» (absorbing). Для каждого , :

.

Шаги 7 – 10 составляют фазу «выжимки» (squeezing).

Шаг 7. Создать пустую строку Z.

Шаг 8. Добавить к строке Z первые r элементов строки S: .

Шаг 9. Если , вернуть первые d символов строки Z: ,

иначе – продолжить.

Шаг 10. Принять и вернуться на шаг 8.

Работа функции проиллюстрирована на рисунке 1.7.

Функция KECCAK[c]

В основном стандарте используется только b равное 1600 битам. Переопределим:

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

Функции SHA-3

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).

Для каждой вычислительной процедуры, указанной в этом стандарте, соответствующая реализация может заменить заданный набор шагов любым математически эквивалентным набором шагов.[[5](#Источник5)][14]

### Конструкция HMAC

Основные достоинства предлагаемой конструкции:

* не требует модификаций существующих хеш-функций;
* сохраняет исходную производительность хеш-функции без существенного замедления работы;

Определение HMAC требует криптографической хеш-функции, которую мы обозначим H, и секретный ключ K. Предполагается, что функция H хеширует данные путем итеративного применения базовой функции к блокам данных. Обозначим длину такого блока в байтах буквой B, а L — длину выходного хеш-значения функции H в байтах. Ключ аутентификации K может иметь любой размер вплоть до размера блока хэш-функции B. Ключи размером больше B, будут сначала хешироваться с помощью H, а затем хеш длиной L будет использоваться, как ключ для HMAC. Рекомендуется использовать ключи K размером не менее L байтов (выходной размер хэш-функции). [12]

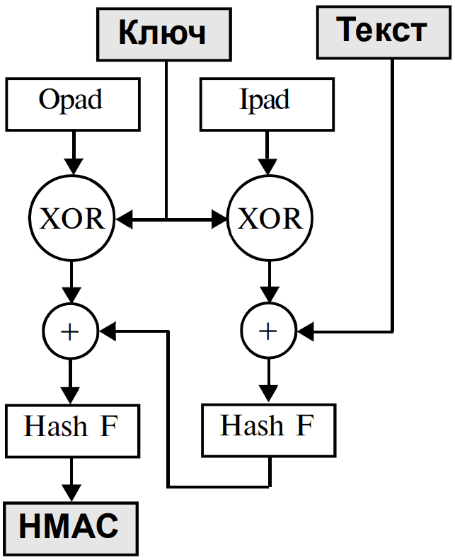
Определим две константных строки ipad и opad:

Рисунок 1.8. Схема работы HMAC.

Для вычисления значения HMAC строки S выполняется операция:

На рисунке 1.8 показана схема работы HMAC. [13]

#### Алгоритм HMAC

Шаг 1. Если , то .

Шаг 2. Принять .

Шаг 3. Если , то

Шаг 4. Для каждого , : ,

.

Шаг 5. Для каждого , : , .

Шаг 6. Вернуть .

## ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

### Графический интерфейс.

Программа запускается по умолчанию в текстовом режиме (рис.2.1).

При помощи кнопок на главном окне можно: вычислить хеш текста, вычислить хеш текста с использованием парольной фразы, сравнить хеш-значение текста и хеш-значение в нижнем текстовом поле (результат сравнения отобразится над полем с хеш-значением (рис.2.2, 2.3)), сравнить с использованием парольной фразы, ввести парольную фразу.

С помощью поля выбора можно включить режим считывания 16-ричных строк из текстового поля.

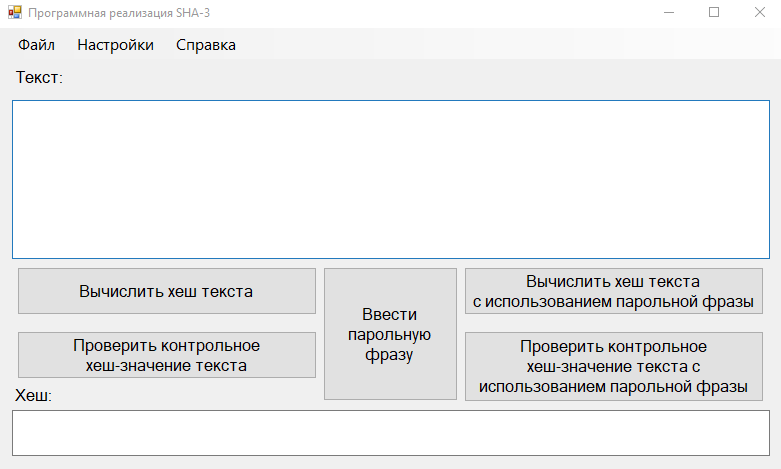
Если парольная фраза не задана и производится попытка вычисления с использованием парольной фразы, то выведется сообщение о необходимости сначала ввести парольную фразу (рис.2.4), после чего откроется окно ввода парольной фразы (рис.2.5). Если парольная фраза не удовлетворяет ограничениям или подтверждение не совпадают, будут выведены соответствующие сообщения.

Рисунок 2.1. Текстовый режим работы.

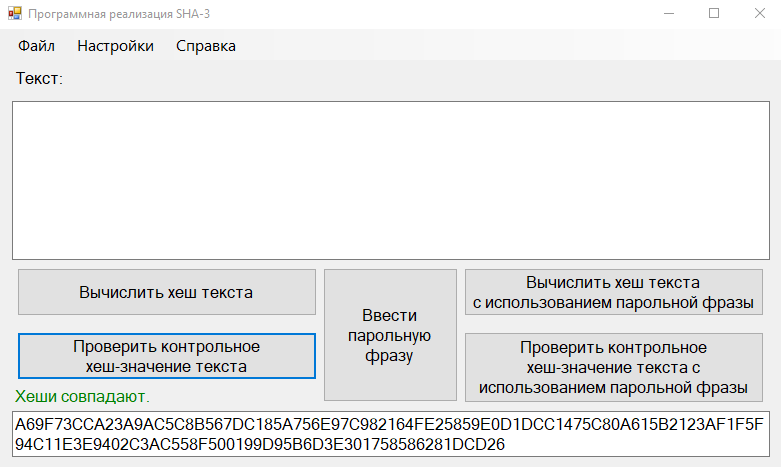
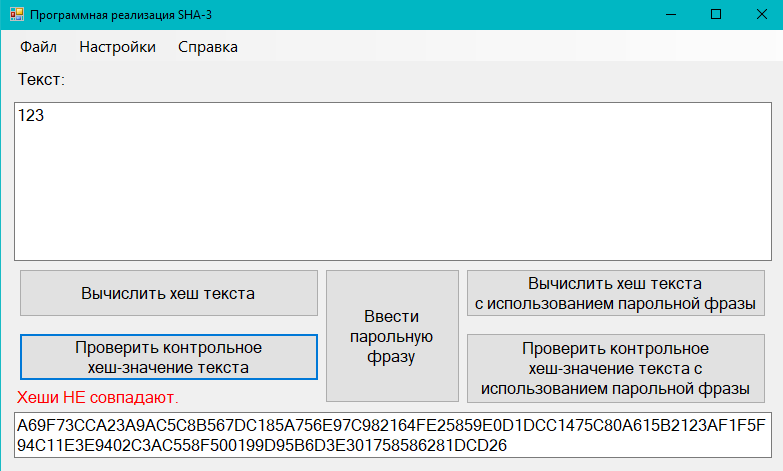
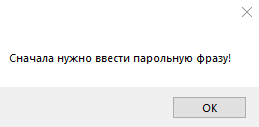


Рисунок 2.3. Отрицательный результат сравнения.

Рисунок 2.2. Положительный результат сравнения.

Рисунок 2.4. Предупреждение.

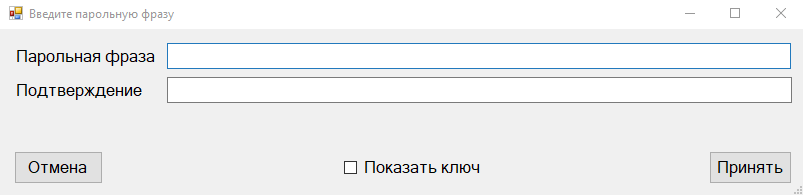
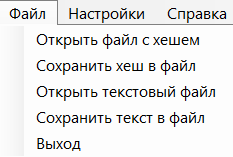
В пункте меню «Файл» доступны возможности: открыть текст или хеш из файла, сохранить текст или хеш в файл, выход из программы (рис.2.6).

Рисунок 2.5. Окно ввода парольной фразы.

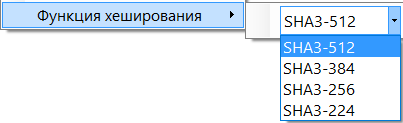
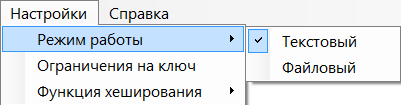
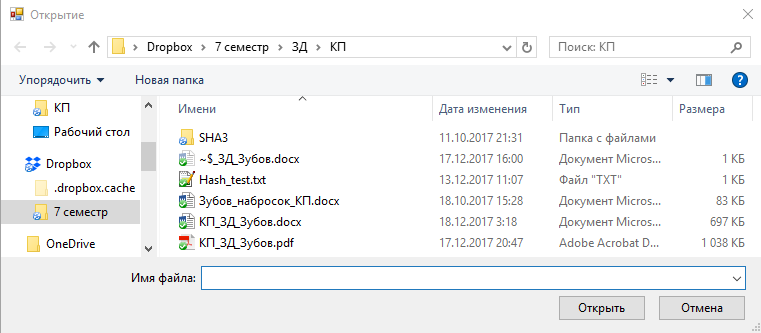
 Пути для работы с файлами запрашиваются при помощи проводника windows (рис.2.7).

Рисунок 2.7. Запрос пути для работы с файлом.

Рисунок 2.8. меню «Настройки».

Рисунок 2.6. меню «Файл».

В пункте меню «Настройки» (рис.2.8) доступны возможности: перейти в файловый режим работы, настроит ограничения, накладываемые на парольную фразу, выбрать используемую функцию хеширования (рис.2.9). По умолчанию используется функция SHA3-512.

Рисунок 2.9. Выбор используемой функции хеширования.

В окне ограничений парольной фразы есть возможность задать минимальную и максимальную длину, потребовать наличия цифр, спецсимволов, строчных и заглавных букв (рис.2.10). После закрытия программы настройки сохраняются в файл «settings».

По умолчанию никакие ограничения не стоят.

Рисунок 2.10. Окно настройки ограничений парольной фразы.

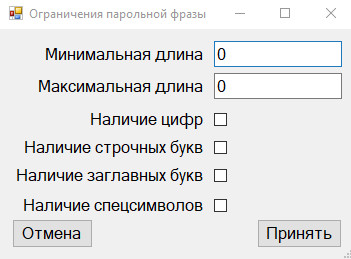
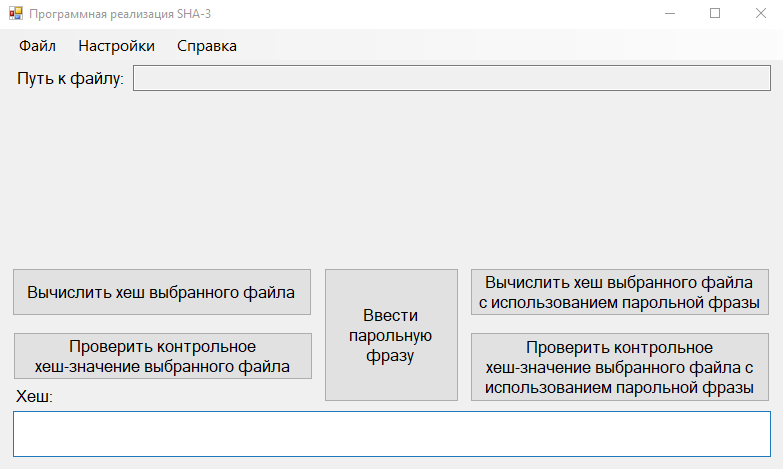
При переходе в файловый режим кнопки меняются на аналогичные кнопкам в текстовом режиме, но предназначенные для работы с файлами, поле для текста скрывается и становится видимым поле выбора пути к файлу, хеш которого требуется находить (рис.2.11).

Рисунок 2.11. Файловый режим работы.

При возникновении ошибки при работе с файлами будет выведено сообщение с подробностями об ошибке (рис.2.12).

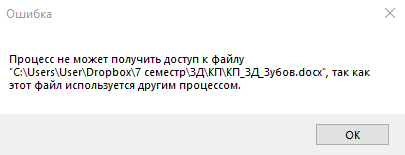
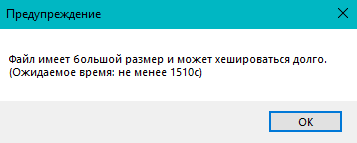
При выборе файла большого размера будет выдано предупреждение о том, что вычисление хеша может быть долгим, с примерной оценкой требующегося времени (рис.2.13).

Рисунок 2.13. Предупреждение.

Рисунок 2.12. Сообщение об ошибке.



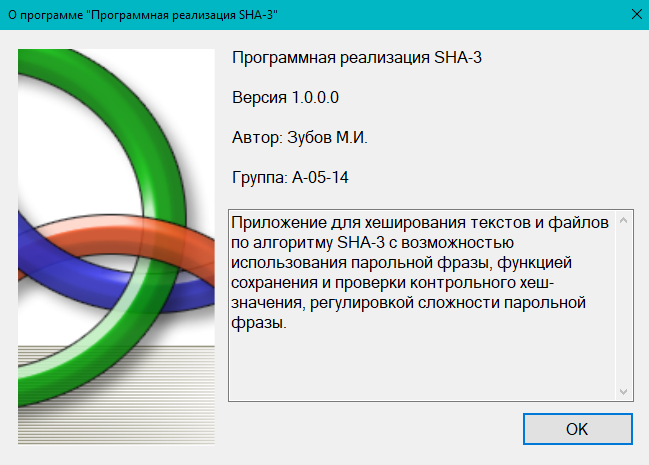
В пункте меню «справка» (рис.2.14) можно посмотреть сведения о программе (рис.2.15).

Рисунок 2.15. Сведения о программе.

Рисунок 2.14. Меню «справка».

### Реализация алгоритмов стандарта SHA-3.

Алгоритмы стандарта реализованы в виде отдельного класса, имеющего публичные методы вызовов внутренних функций для защиты от случайных изменений состояний класса извне.

Так как длина линии в матричном представлении состояния в основном стандарте составляет 64 бита, для его хранения был выбран массив 5 на 5 с 8 байтовыми беззнаковыми целыми значениями (UInt64), что позволило производить эффективные вычисления на современных процессорах. Таким образом дальнейшие преобразования производятся над линиями, а не над отдельными битами состояния.

Все вспомогательные значения, используемые в пошаговых отображениях, вычислены заранее, так как зависят только от размера строки состояния и количества раундов, которые в основном стандарте являются константами.

Алгоритм губки был доработан таким образом, чтобы не требовалось знать длину входящего сообщения, что позволило обрабатывать файловые потоки, а также избавило от некоторых вычислений. Это было достигнуто благодаря применению функции дополнения не ко всему сообщению до фазы впитывания, а только к последнему, неполному блоку, что эквивалентно.

Методы реализованы в виде отдельного класса «SHA3».

Приватные методы:

Keccak – основная функция класса. Принимает на вход размер выходной строки, байтовый массив с входным сообщением или файловый поток, байтовый массив с ключом (или null, если ключ не используется). Возвращает байтовый массив соответствующей длины.

Last\_block\_proc – функция обработки последнего блока сообщения. Дополняет последний блок сообщения до размера обрабатываемого блока. Вызывается только из функции Keccak. Принимает на вход двумерный 8-ми байтовый (UInt64) массив состояния, байтовый массив с размером меньше или равным размеру обрабатываемого блока (при кратности длины сообщения размеру обрабатываемого блока будет иметь нулевой размер, так как дополнение сообщения происходит и в этом случае), содержащий последнюю часть входного сообщения, размер обрабатываемого блока. Возвращает обновлённый массив состояния.

State\_Change – складывает по модулю 2 блок сообщения и массив состояния, после чего перемешивает состояние в соответствии с алгоритмом. Принимает на вход двумерный 8-ми байтовый (UInt64) массив состояния, байтовый массив размера обрабатываемого блока, содержащий часть входного сообщения, размер обрабатываемого блока. Возвращает обновлённый массив состояния.

Squeezing – выделяет из массива состояния хеш-значение требуемой длины. Вызывается только из функции Keccak. Принимает на вход двумерный 8-ми байтовый (UInt64) массив состояния, размер обрабатываемого блока. Возвращает байтовый массив соответствующей длины.

Round – функция, выполняющая перестановки в массиве состояний. Принимает на вход двумерный 8-ми байтовый (UInt64) массив состояния и одну из констант. Возвращает обновлённый массив состояния.

Rotate – вспомогательная функция перестановок, используемая в функции Round. Принимает на вход линию (один элемент массива состояний), одну из констант и длину линии. Возвращает обновлённую линию.

SubArray – функция выделяющая подмассив.

StringToByte – функция переводящая строку в массив байт.

ByteToHex – функция переводящая массив байт в 16-ричную строку.

HexToByte – функция переводящая 16-ричную строку в массив байт.

Публичные методы:

SHA3\_224, SHA3\_256, SHA3\_384, SHA3\_512 – принимают строку сообщения, возвращают 16-ричную строку соответствующей длины.

SHA3\_224\_HEX, SHA3\_256\_HEX, SHA3\_384\_HEX, SHA3\_512\_HEX– принимают 16-ричную строку сообщения, возвращают 16-ричную строку соответствующей длины.

### Реализация конструкции HMAC.

Конструкция встроена в класс, производящий хеширование.

Приватные методы: Key\_transform – принимает на вход строку с ключом и размер обрабатываемого блока, возвращает массив из двух байтовых строк: ipad и opad.

Публичные методы: SHA3\_HMAC\_224, SHA3\_HMAC\_256, SHA3\_HMAC\_384, SHA3\_HMAC\_512 – принимают строку сообщения и ключа, возвращают 16-ричную строку соответствующей длины. SHA3\_HMAC\_224\_HEX, SHA3\_HMAC\_256\_HEX, SHA3\_HMAC\_384\_HEX, SHA3\_HMAC\_512\_HEX– принимают 16-ричную строку сообщения и строку ключа, возвращают 16-ричную строку соответствующей длины.

### Обработка парольной фразы.

От введенной парольной фразы находится хеш-значение SHA3-512 и сохраняется в качестве ключа, который далее используется при хешировании и проверке целостности данных с использованием конструкции HMAC.

## ТЕСТИРОВАНИЕ РАЗРАБОТАННОЙ ПРОГРАММЫ

Сравним хеш-значения, выдаваемые программой с тестами, подготовленными разработчиками стандарта.[11]

Sec – размер выходной строки.

Len – длина входного сообщения.

Msg – входное сообщение в 16-ричном формате.

MD – хеш-значение, предоставленное разработчиками.

hash – хеш-значение, полученное с помощью программы.

Таблица 3.1. Тест 1.

|  |  |
| --- | --- |
| Sec | 224 |
| Len | 2312 |
| Msg |  |
| MD | aab23c9e7fb9d7dacefdfd0b1ae85ab1374abff7c4e3f7556ecae412 |
| hash | aab23c9e7fb9d7dacefdfd0b1ae85ab1374abff7c4e3f7556ecae412 |

Таблица 3.2. Тест 2.

|  |  |
| --- | --- |
| Sec | 256 |
| Len | 2184 |
| Msg |  |
| MD | cb5648a1d61c6c5bdacd96f81c9591debc3950dcf658145b8d996570ba881a05 |
| hash | cb5648a1d61c6c5bdacd96f81c9591debc3950dcf658145b8d996570ba881a05 |

Таблица 3.3. Тест 3.

|  |  |
| --- | --- |
| Sec | 384 |
| Len | 1672 |
| Msg | 5fe35923b4e0af7dd24971812a58425519850a506dfa9b0d254795be785786c319a2567cbaa5e35bcf8fe83d943e23fa5169b73adc1fcf8b607084b15e6a013df147e46256e4e803ab75c110f77848136be7d806e8b2f868c16c3a90c14463407038cb7d9285079ef162c6a45cedf9c9f066375c969b5fcbcda37f02aacff4f31cded3767570885426bebd9eca877e44674e9ae2f0c24cdd0e7e1aaf1ff2fe7f80a1c4f5078eb34cd4f06fa94a2d1eab5806ca43fd0f06c60b63d5402b95c70c21ea65a151c5cfaf8262a46be3c722264b |
| MD | 3054d249f916a6039b2a9c3ebec1418791a0608a170e6d36486035e5f92635eaba98072a85373cb54e2ae3f982ce132b |
| hash | 3054d249f916a6039b2a9c3ebec1418791a0608a170e6d36486035e5f92635eaba98072a85373cb54e2ae3f982ce132b |

Таблица 3.4. Тест 4.

|  |  |
| --- | --- |
| Sec | 512 |
| Len | 1160 |
| Msg | 664ef2e3a7059daf1c58caf52008c5227e85cdcb83b4c59457f02c508d4f4f69f826bd82c0cffc5cb6a97af6e561c6f96970005285e58f21ef6511d26e709889a7e513c434c90a3cf7448f0caeec7114c747b2a0758a3b4503a7cf0c69873ed31d94dbef2b7b2f168830ef7da3322c3d3e10cafb7c2c33c83bbf4c46a31da90cff3bfd4ccc6ed4b310758491eeba603a76 |
| MD | e5825ff1a3c070d5a52fbbe711854a440554295ffb7a7969a17908d10163bfbe8f1d52a676e8a0137b56a11cdf0ffbb456bc899fc727d14bd8882232549d914e |
| hash | e5825ff1a3c070d5a52fbbe711854a440554295ffb7a7969a17908d10163bfbe8f1d52a676e8a0137b56a11cdf0ffbb456bc899fc727d14bd8882232549d914e |

Таблица 3.5. Тест 5.

|  |  |
| --- | --- |
| Sec | 512 |
| Len | 0 |
| Msg |  |
| MD | a69f73cca23a9ac5c8b567dc185a756e97c982164fe25859e0d1dcc1475c80a615b2123af1f5f94c11e3e9402c3ac558f500199d95b6d3e301758586281dcd26 |
| hash | a69f73cca23a9ac5c8b567dc185a756e97c982164fe25859e0d1dcc1475c80a615b2123af1f5f94c11e3e9402c3ac558f500199d95b6d3e301758586281dcd26 |

Проверим работу функции проверки хеш-значения.

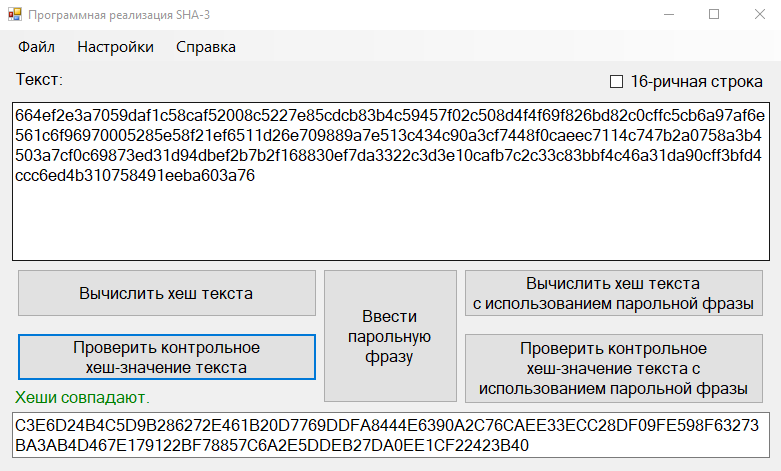


Рисунок 3.1. Тестирование функции проверки хеш-значения.

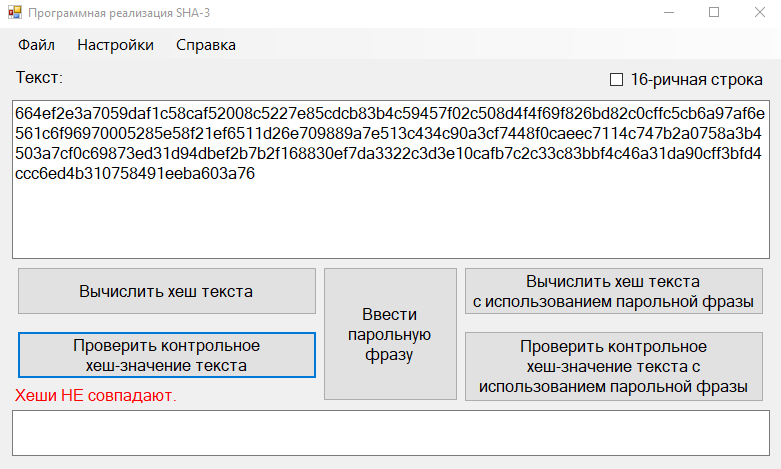


Рисунок 3.2. Тестирование функции проверки хеш-значения.

Проверим работу конструкции HMAC с помощью сторонней реализации алгоритма.[16]

Сгенерируем вручную ключ из парольной фразы для проверки алгоритма.

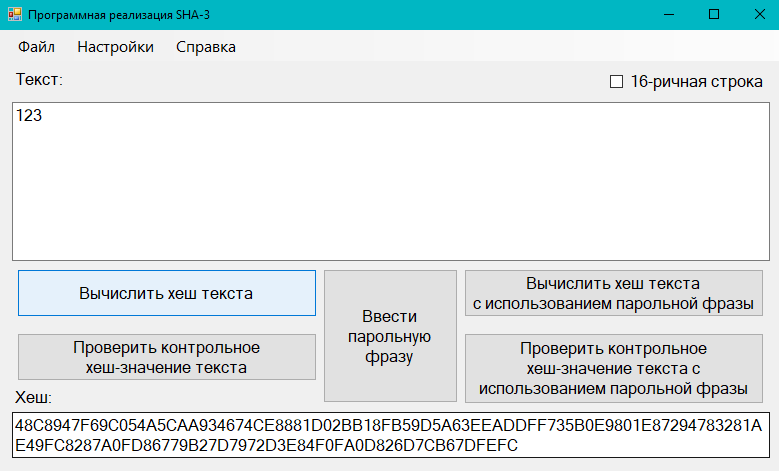


Рисунок 3.3. Ручная генерация ключа.

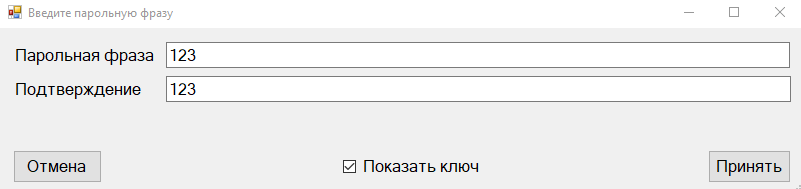


Рисунок 3.4. Ввод парольной фразы.

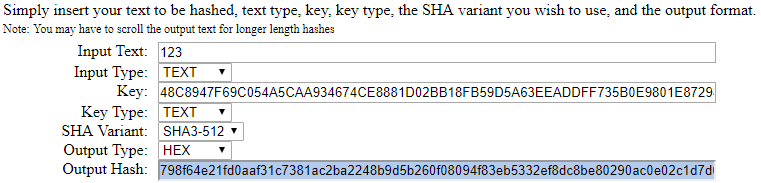


Рисунок 7.5. Генерация хеша с использованием ключа с помощью сторонней реализации.

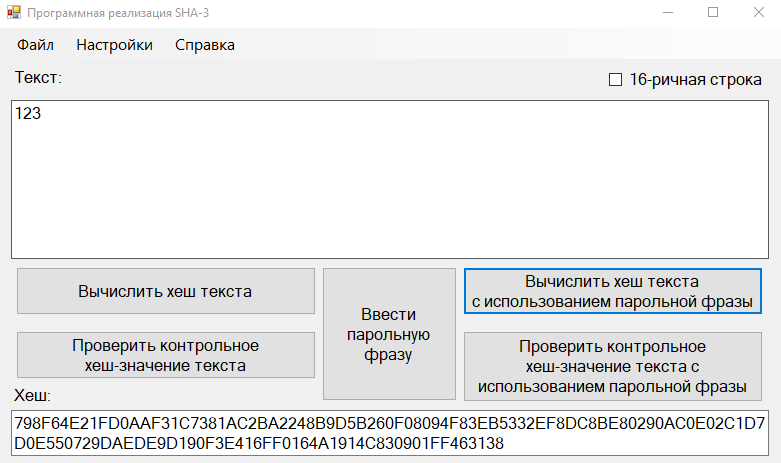


Рисунок 3.6. Генерация хеша с использованием парольной фразы.

Проверим работу нахождения хеш-значения файла с помощью сторонней реализации алгоритма.[17]

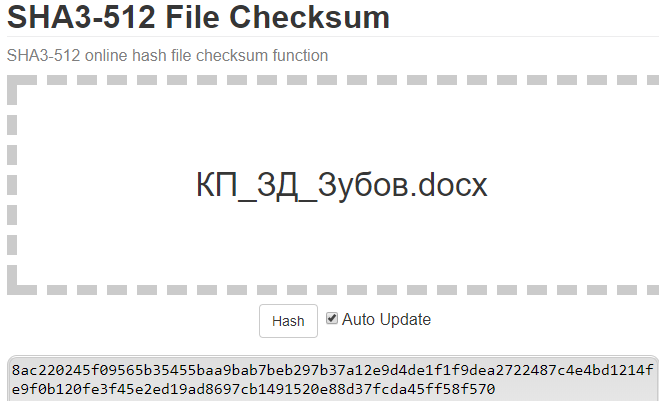


Рисунок 3.7. Генерация хеша файла с помощью сторонней реализации.

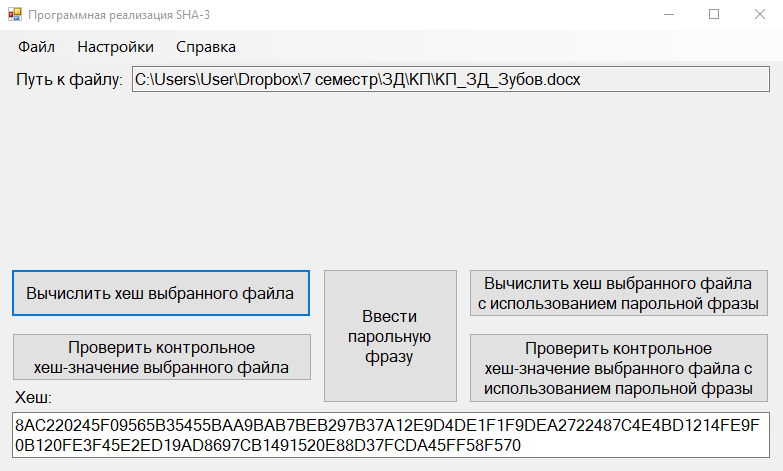


Рисунок 3.8. Генерация хеша файла.

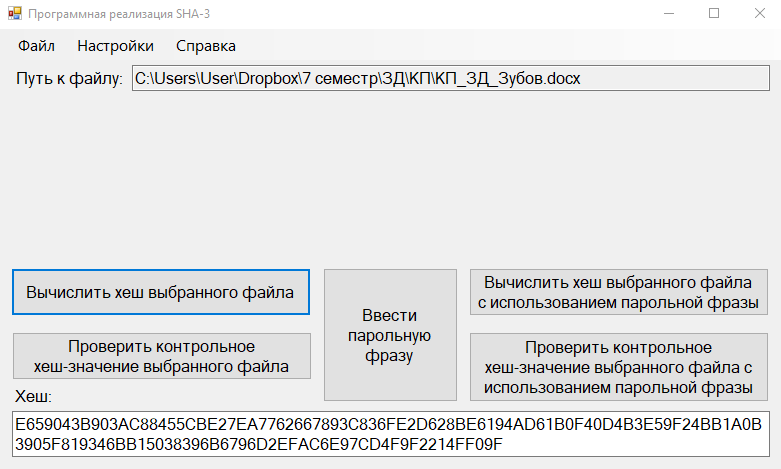
После изменения файла хеш-значение изменилось. 

Рисунок 3.9. Генерация хеша файла после изменения файла.

Проверим работу ограничений парольной фразы.

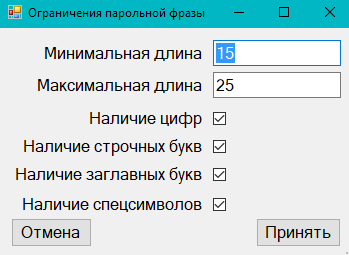


Рисунок 3.10. Ввод ограничений парольной фразы.

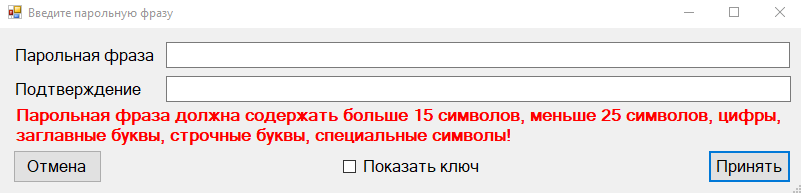


Рисунок 3.11. Проверка работы ограничений парольной фразы.

## ЗАКЛЮЧЕНИЕ

По итогам этой работы было разработано программное обеспечение, реализующее стандарт SHA-3 с возможностями потокового хеширования произвольных файлов и текстовых сообщений, использования ключа с регулируемой минимальной длиной и сложностью, сохранения в файле полученного хеш-значения, в качестве цифровой подписи файлов и сообщений.

Были получены навыки по реализации криптографических стандартов, составлению пояснительной записки, оптимизации алгоритмов.

# СПИСОК ИСТОЧНИКОВ

1. Guido Bertoni, Joan Daemen, Michaёl Peeters and Gilles Van Assche.

The Road from Panama to Keccak via RadioGatún.

https://keccak.team/files/TheRoadFromPanamaToKeccak.pdf

1. Guido Bertoni, Joan Daemen, Michaёl Peeters and Gilles Van Assche.

Cryptographic sponge functions.

https://keccak.team/sponge\_duplex.html

1. SHA-3 Project

https://csrc.nist.gov/projects/hash-functions/sha-3-project

1. NIST Releases SHA-3 Cryptographic Hash Standard

https://www.nist.gov/news-events/news/2015/08/nist-releases-sha-3-cryptographic-hash-standard

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

http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

1. Is SHA-3 slow?

https://keccak.team/2017/is\_sha3\_slow.html

1. Third-Round Report of the SHA-3 Cryptographic Hash Algorithm Competition

http://nvlpubs.nist.gov/nistpubs/ir/2012/NIST.IR.7896.pdf

1. Comparing Hardware Performance of Round 3 SHA-3 Candidates using Multiple Hardware Architectures in Xilinx and Altera FPGAs

http://www.ecrypt.eu.org/hash2011/proceedings/hash2011\_07.pdf

1. SHA-3 Hardware Project

http://www.rcis.aist.go.jp/special/SASEBO/SHA3-en.html

1. Merkle-Damgard Revisited : how to Construct a Hash Function

http://www.crypto-uni.lu/jscoron/publications/merkle.pdf

1. Cryptographic Algorithm Validation Program

https://csrc.nist.gov/Projects/Cryptographic-Algorithm-Validation-Program/Secure-Hashing#sha3vsha3vss

1. HMAC: Keyed-Hashing for Message Authentication

http://www.protocols.ru/WP/wp-content/uploads/2017/01/rfc2104.pdf

1. Фрадков С.А. Алгоритм HMAC - цифровая подпись в реальном масштабе времени для протокола IPv6. Компьютерная инженерия и техническая диагностика, 1999, №2, с.82-84.
2. Королев Д.С. Модуль вычисления хеш-функции по международному стандарту SHA-3 для языка PHP. Выпускная квалификационная работа специалиста, 2017.
3. Keccak, новый стандарт хеширования данных https://habrahabr.ru/post/159073/
4. jsSHA. https://caligatio.github.io/jsSHA/
5. SHA3-512 File Checksum

<https://emn178.github.io/online-tools/sha3_512_checksum.html>

# Приложение 1.

**ЛИСТНИНГ ПРОГРАММЫ**

## Файл Main.cs

using System;

using System.Drawing;

using System.Windows.Forms;

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

namespace SHA3

{

[Serializable]

public struct KeyRestrict

{

UInt16 min\_length;

UInt16 max\_length;

bool numbers;

bool lowercase;

bool capitals;

bool special\_chars;

public UInt16 Min\_length { get => min\_length; set => min\_length = value; }

public UInt16 Max\_length { get => max\_length; set => max\_length = value; }

public bool Numbers { get => numbers; set => numbers = value; }

public bool Lowercase { get => lowercase; set => lowercase = value; }

public bool Capitals { get => capitals; set => capitals = value; }

public bool Special\_chars { get => special\_chars; set => special\_chars = value; }

public KeyRestrict(UInt16 minimal\_length, UInt16 maximal\_length, bool contain\_numbers, bool contain\_lowercase, bool contain\_capitals, bool contain\_special\_chars)

{

min\_length = minimal\_length;

max\_length = maximal\_length;

numbers = contain\_numbers;

lowercase = contain\_lowercase;

capitals = contain\_capitals;

special\_chars = contain\_special\_chars;

}

}

public partial class Main : Form

{

public static string key;

public static KeyRestrict keyRestrict = new KeyRestrict(0, 0, false, false, false, false);

private string savePath;

private string choosePath;

private const string settingsPath = ".\\settings";

private static string hashFunction;

public Main()

{

InitializeComponent();

FileMod(false);

toolStripComboBox\_Hash\_Func\_Choose.DropDownStyle = System.Windows.Forms.ComboBoxStyle.DropDownList;

toolStripComboBox\_Hash\_Func\_Choose.SelectedItem = toolStripComboBox\_Hash\_Func\_Choose.Items[0];

if (File.Exists(settingsPath))

try

{

using (FileStream fs = new FileStream(settingsPath, FileMode.OpenOrCreate, FileAccess.Read))

{

BinaryFormatter formatter = new BinaryFormatter();

keyRestrict = (KeyRestrict)formatter.Deserialize(fs);

}

}

catch (Exception)

{

MessageBox.Show("Файл настроек повреждён\nнастройки будут сброшены.", "Ошибка");

}

}

private void ExitToolStripMenuItem\_Click(object sender, EventArgs e)

{

Close();

}

private void OpenHashFileToolStripMenuItem\_Click(object sender, EventArgs e)

{

OpenFileDialog OPF = new OpenFileDialog();

if (OPF.ShowDialog() == DialogResult.OK)

{

Hash\_textBox.Text = File.ReadAllText(OPF.FileName);

}

}

private void SaveAsHashFileToolStripMenuItem\_Click(object sender, EventArgs e)

{

SaveFileDialog SF = new SaveFileDialog();

if (SF.ShowDialog() == DialogResult.OK)

{

savePath = SF.FileName;

File.WriteAllText(savePath, Hash\_textBox.Text);

}

}

private void сохранитьТекстВФайлToolStripMenuItem\_Click(object sender, EventArgs e)

{

SaveFileDialog SF = new SaveFileDialog();

if (SF.ShowDialog() == DialogResult.OK)

{

File.WriteAllText(SF.FileName, Text\_textBox.Text);

}

}

private void открытьТекстИзФайлаToolStripMenuItem\_Click(object sender, EventArgs e)

{

OpenFileDialog OPF = new OpenFileDialog();

if (OPF.ShowDialog() == DialogResult.OK)

{

Text\_textBox.Text = File.ReadAllText(OPF.FileName);

}

}

private void TextModeToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (FileModeToolStripMenuItem.Checked)

{

FileMod(false);

}

}

private void FileModeToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (TextModeToolStripMenuItem.Checked)

{

FileMod(true);

}

}

private void KeyConstraintsToolStripMenuItem\_Click(object sender, EventArgs e)

{

KeySet keySet = new KeySet();

keySet.ShowDialog();

}

private void AboutToolStripMenuItem\_Click(object sender, EventArgs e)

{

AboutBox aboutBox = new AboutBox();

aboutBox.ShowDialog();

}

private void Choose\_File\_Path\_Click(object sender, EventArgs e)

{

OpenFileDialog OPF = new OpenFileDialog();

if (OPF.ShowDialog() == DialogResult.OK)

{

choosePath = OPF.FileName;

textBox\_Choose\_File\_Path.Text = choosePath;

FileInfo fileInfo = new FileInfo(choosePath);

double len\_in\_MiBs = fileInfo.Length/1024/1024;

if (len\_in\_MiBs > 10)

MessageBox.Show("Файл имеет большой размер и может хешироваться долго.\n(Ожидаемое время: не менее " + (Math.Truncate(len\_in\_MiBs \* 0.41)).ToString() + "с)", "Предупреждение");

}

}

private void toolStripComboBox\_Hash\_Func\_Choose\_SelectedIndexChanged(object sender, EventArgs e)

{

hashFunction = toolStripComboBox\_Hash\_Func\_Choose.SelectedItem.ToString();

}

private void button\_key\_input\_Click(object sender, EventArgs e)

{

KeyInput keyInput = new KeyInput();

keyInput.ShowDialog();

}

private void Calc\_Text\_Hash\_button\_Click(object sender, EventArgs e)

{

Hash\_textBox.Text = Get\_SHA3\_Hash(Text\_textBox.Text);

result\_label\_reset();

}

private void Calc\_Text\_Hash\_with\_key\_button\_Click(object sender, EventArgs e)

{

if (key\_not\_empty\_Check())

{

Hash\_textBox.Text = Get\_HMAC\_SHA3\_Hash(Text\_textBox.Text, key);

result\_label\_reset();

}

}

private void Check\_Text\_Hash\_button\_Click(object sender, EventArgs e)

{

Hash\_compare(Get\_SHA3\_Hash(Text\_textBox.Text));

}

private void button\_Check\_Text\_Hash\_with\_key\_Click(object sender, EventArgs e)

{

if (key\_not\_empty\_Check())

{

Hash\_compare(Get\_HMAC\_SHA3\_Hash(Text\_textBox.Text, key));

}

}

private void Calc\_File\_Hash\_button\_Click(object sender, EventArgs e)

{

if (file\_exist\_check())

{

try

{

using (FileStream fs = File.OpenRead(choosePath))

{

Hash\_textBox.Text = Get\_SHA3\_Hash(fs);

}

}

catch (Exception exc)

{

MessageBox.Show(exc.Message, "Ошибка");

}

result\_label\_reset();

}

}

private void Calc\_File\_Hash\_with\_key\_button\_Click(object sender, EventArgs e)

{

if (key\_not\_empty\_Check() && file\_exist\_check())

{

try

{

using (FileStream fs = File.OpenRead(choosePath))

{

Hash\_textBox.Text = Get\_HMAC\_SHA3\_Hash(fs, key);

}

}

catch (Exception exc)

{

MessageBox.Show(exc.Message, "Ошибка");

}

result\_label\_reset();

}

}

private void Check\_File\_Hash\_button\_Click(object sender, EventArgs e)

{

if (file\_exist\_check())

{

try

{

using (FileStream fs = File.OpenRead(choosePath))

{

Hash\_compare(Get\_SHA3\_Hash(fs));

}

}

catch (Exception exc)

{

MessageBox.Show(exc.Message, "Ошибка");

}

}

}

private void button\_Check\_File\_Hash\_with\_key\_Click(object sender, EventArgs e)

{

if (key\_not\_empty\_Check() && file\_exist\_check())

{

try

{

using (FileStream fs = File.OpenRead(choosePath))

{

Hash\_compare(Get\_HMAC\_SHA3\_Hash(fs, key));

}

}

catch (Exception exc)

{

MessageBox.Show(exc.Message, "Ошибка");

}

}

}

private bool file\_exist\_check()

{

if (File.Exists(choosePath))

{

return true;

}

else

{

MessageBox.Show("Файл не существует!");

return false;

}

}

private bool key\_not\_empty\_Check()

{

if (!String.IsNullOrEmpty(key))

{

return true;

}

else

{

MessageBox.Show("Сначала нужно ввести парольную фразу!");

KeyInput keyInput = new KeyInput();

keyInput.ShowDialog();

return false;

}

}

private void Hash\_compare(string hash)

{

if (Hash\_textBox.Text == hash)

{

label\_Result.ForeColor = Color.Green;

label\_Result.Text = "Хеши совпадают.";

}

else

{

label\_Result.ForeColor = Color.Red;

label\_Result.Text = "Хеши НЕ совпадают.";

}

}

private void result\_label\_reset()

{

label\_Result.ForeColor = Color.Black;

label\_Result.Text = "Хеш:";

}

private void FileMod(bool on)

{

//File mode

FileModeToolStripMenuItem.Checked = on;

textBox\_Choose\_File\_Path.Visible = on;

Calc\_File\_Hash\_button.Visible = on;

Calc\_File\_Hash\_with\_key\_button.Visible = on;

Check\_File\_Hash\_button.Visible = on;

button\_Check\_File\_Hash\_with\_key.Visible = on;

if (on) Choose\_File\_label.Text = "Путь к файлу:";

else Choose\_File\_label.Text = "Текст:";

//Text mode

TextModeToolStripMenuItem.Checked = !on;

checkBox\_HEX.Visible = !on;

Text\_textBox.Visible = !on;

Calc\_Text\_Hash\_button.Visible = !on;

Calc\_Text\_Hash\_with\_key\_button.Visible = !on;

Check\_Text\_Hash\_button.Visible = !on;

button\_Check\_Text\_Hash\_with\_key.Visible = !on;

}

public static string Get\_SHA3\_Hash(string Text)

{

if (!checkBox\_HEX.Checked)

switch (hashFunction)

{

case "SHA3-512":

return SHA3.SHA3\_512(Text);

case "SHA3-384":

return SHA3.SHA3\_384(Text);

case "SHA3-256":

return SHA3.SHA3\_256(Text);

case "SHA3-224":

return SHA3.SHA3\_224(Text);

}

else

switch (hashFunction)

{

case "SHA3-512":

return SHA3.SHA3\_512\_HEX(Text);

case "SHA3-384":

return SHA3.SHA3\_384\_HEX(Text);

case "SHA3-256":

return SHA3.SHA3\_256\_HEX(Text);

case "SHA3-224":

return SHA3.SHA3\_224\_HEX(Text);

}

return "Error in switching.";

}

public static string Get\_HMAC\_SHA3\_Hash(string Text, string Key)

{

if (!checkBox\_HEX.Checked)

switch (hashFunction)

{

case "SHA3-512":

return SHA3.SHA3\_HMAC\_512(Text, Key);

case "SHA3-384":

return SHA3.SHA3\_HMAC\_384(Text, Key);

case "SHA3-256":

return SHA3.SHA3\_HMAC\_256(Text, Key);

case "SHA3-224":

return SHA3.SHA3\_HMAC\_224(Text, Key);

}

else

switch (hashFunction)

{

case "SHA3-512":

return SHA3.SHA3\_HMAC\_512\_HEX(Text, Key);

case "SHA3-384":

return SHA3.SHA3\_HMAC\_384\_HEX(Text, Key);

case "SHA3-256":

return SHA3.SHA3\_HMAC\_256\_HEX(Text, Key);

case "SHA3-224":

return SHA3.SHA3\_HMAC\_224\_HEX(Text, Key);

}

return "Error in switching.";

}

public static string Get\_SHA3\_Hash(FileStream fs)

{

switch (hashFunction)

{

case "SHA3-512":

return SHA3.SHA3\_512(fs);

case "SHA3-384":

return SHA3.SHA3\_384(fs);

case "SHA3-256":

return SHA3.SHA3\_256(fs);

case "SHA3-224":

return SHA3.SHA3\_224(fs);

}

return "Error in switching.";

}

public static string Get\_HMAC\_SHA3\_Hash(FileStream fs, string Key)

{

switch (hashFunction)

{

case "SHA3-512":

return SHA3.SHA3\_HMAC\_512(fs, Key);

case "SHA3-384":

return SHA3.SHA3\_HMAC\_384(fs, Key);

case "SHA3-256":

return SHA3.SHA3\_HMAC\_256(fs, Key);

case "SHA3-224":

return SHA3.SHA3\_HMAC\_224(fs, Key);

}

return "Error in switching.";

}

private void Main\_FormClosing(object sender, FormClosingEventArgs e)

{

try

{

if (File.Exists(settingsPath))

File.SetAttributes(settingsPath, FileAttributes.Normal);

using (FileStream fs = new FileStream(settingsPath, FileMode.OpenOrCreate, FileAccess.Write))

{

BinaryFormatter formatter = new BinaryFormatter();

formatter.Serialize(fs, keyRestrict);

}

}

catch (Exception exc)

{

MessageBox.Show("Не удалось сохранить настройки.\n" + exc.Message, "Ошибка");

}

}

}

}

## Файл KeySet.cs

using System;

using System.Windows.Forms;

namespace SHA3

{

public partial class KeySet : Form

{

UInt16 min, max;

public KeySet()

{

InitializeComponent();

textBox\_min\_length.Text = Main.keyRestrict.Min\_length.ToString();

textBox\_max\_length.Text = Main.keyRestrict.Max\_length.ToString();

checkBox\_capitals.Checked = Main.keyRestrict.Capitals;

checkBox\_lowercase.Checked = Main.keyRestrict.Lowercase;

checkBox\_Numbers.Checked = Main.keyRestrict.Numbers;

checkBox\_spec\_chars.Checked = Main.keyRestrict.Special\_chars;

}

private void button2\_Click(object sender, EventArgs e)

{

Close();

}

private void textBox\_min\_length\_KeyPress(object sender, KeyPressEventArgs e)

{

if (e.KeyChar < '0' || e.KeyChar > '9') e.KeyChar = '\0';

}

private void textBox\_max\_length\_KeyPress(object sender, KeyPressEventArgs e)

{

if (e.KeyChar < '0' || e.KeyChar > '9') e.KeyChar = '\0';

}

private void button1\_Click(object sender, EventArgs e)

{

try

{

min = Convert.ToUInt16(textBox\_min\_length.Text);

}

catch (Exception)

{

min = 0;

}

try

{

max = Convert.ToUInt16(textBox\_max\_length.Text);

}

catch (Exception)

{

max = 0;

}

if (max < min)

{

max = min;

}

Main.keyRestrict = new KeyRestrict(min, max, checkBox\_Numbers.Checked, checkBox\_lowercase.Checked, checkBox\_capitals.Checked, checkBox\_spec\_chars.Checked);

Close();

}

}

}

## Файл KeyInput.cs

using System;

using System.Linq;

using System.Windows.Forms;

namespace SHA3

{

public partial class KeyInput : Form

{

string s, l;

public KeyInput()

{

InitializeComponent();

}

private void Cancel\_Key\_button\_Click(object sender, EventArgs e)

{

Close();

}

private void Ok\_Key\_button\_Click(object sender, EventArgs e)

{

if (Pass\_is\_good(Pass\_Key\_textBox.Text))

{

if (Pass\_Key\_textBox.Text == textBox\_confirm.Text)

{

Main.key = Main.Get\_SHA3\_Hash(Pass\_Key\_textBox.Text);

Close();

}

else

{

label\_err.Text = "Парольная фраза и подтверждение не совпадают!";

}

}

else

{

l = "";

if (Main.keyRestrict.Min\_length != 0)

{

l = " больше " + Main.keyRestrict.Min\_length;

s = Main.keyRestrict.Min\_length.ToString();

if (s.Substring(s.Length - 1) == "1") l += " символа,";

else l += " символов,";

}

if (Main.keyRestrict.Max\_length != 0)

{

l += " меньше " + Main.keyRestrict.Max\_length;

s = Main.keyRestrict.Max\_length.ToString();

if (s.Substring(s.Length - 1) == "1") l += " символа,";

else l += " символов,";

}

if (Main.keyRestrict.Numbers) l += " цифры,";

if (Main.keyRestrict.Capitals) l += " заглавные буквы,";

if (Main.keyRestrict.Lowercase) l += " строчные буквы,";

if (Main.keyRestrict.Special\_chars) l += " специальные символы,";

label\_err.Text = "Парольная фраза должна содержать" + l.Substring(0, l.Length - 1) + "!";

}

}

private bool Pass\_is\_good(string pass)

{

if (pass.Length < Main.keyRestrict.Min\_length ||

Main.keyRestrict.Max\_length != 0 && pass.Length > Main.keyRestrict.Max\_length ||

Main.keyRestrict.Lowercase && !pass.Any(c => char.IsLower(c)) ||

Main.keyRestrict.Numbers && !pass.Any(c => char.IsNumber(c)) ||

Main.keyRestrict.Capitals && !pass.Any(c => char.IsUpper(c)) ||

Main.keyRestrict.Special\_chars && !pass.Any(c => char.IsSymbol(c)))

return false;

return true;

}

private void Pass\_Key\_checkBox\_Click(object sender, EventArgs e)

{

if (Pass\_Key\_checkBox.Checked)

{

Pass\_Key\_textBox.PasswordChar = '\0';

textBox\_confirm.PasswordChar = '\0';

}

else

{

Pass\_Key\_textBox.PasswordChar = '\*';

textBox\_confirm.PasswordChar = '\*';

}

}

}

}

## Файл SHA3.cs

using System;

using System.Linq;

using System.Text;

using System.IO;

using System.Globalization;

namespace SHA3

{

class SHA3

{

#region constants

private static Byte matrixSize = 5 \* 5;

private static Byte state\_length = 6; //(default: 6)(l)

private static Byte state\_width = 64; //(default: 64)(bit) ширина массива состояний (w = 2^l)

private static Byte state\_width\_in\_bytes = 8;

private static UInt16 permutation\_width = 1600; //(default: 1600)(bit) ширина перестановок (b = 25 \* w)

//при необходимости можно добавлять нестандартные режимы, при условии сохранения относительного расположения значений

private static UInt16[] rate\_array = { 1152, 1088, 832, 576 }; //(bit) (r = b - c)

private static UInt16[] rate\_in\_bytes\_array = { 144, 136, 104, 72 };

private static UInt16[] capacity\_array = { 448, 512, 768, 1024 }; //(bit) (c = security \* 2)

private static UInt16[] capacity\_in\_bytes\_array = { 56, 64, 96, 128 };

private static UInt16[] security\_array = { 224, 256, 384, 512 };

private static UInt16[] security\_in\_bytes\_array = { 28, 32, 48, 64 };

private static Byte rounds\_number = 24; //(default: 24) 12 + 2 \* l число раундов Keccak-f

//((t + 1)(t + 2)/2) % (state\_width = 64)

private static Byte[,] offset = {

{0, 36, 3, 41, 18},

{1, 44, 10, 45, 2},

{62, 6, 43, 15, 61},

{28, 55, 25, 21, 56},

{27, 20, 39, 8, 14}};

//rotation \_constants for l = 6 (rounds\_number = 24)

private static UInt64[] RC = {

0x0000000000000001,

0x0000000000008082,

0x800000000000808A,

0x8000000080008000,

0x000000000000808B,

0x0000000080000001,

0x8000000080008081,

0x8000000000008009,

0x000000000000008A,

0x0000000000000088,

0x0000000080008009,

0x000000008000000A,

0x000000008000808B,

0x800000000000008B,

0x8000000000008089,

0x8000000000008003,

0x8000000000008002,

0x8000000000000080,

0x000000000000800A,

0x800000008000000A,

0x8000000080008081,

0x8000000000008080,

0x0000000080000001,

0x8000000080008008};

#endregion

private static UInt64[,] B = new UInt64[5, 5];

private static UInt64[] C = new UInt64[5];

private static UInt64[] D = new UInt64[5];

private static Byte[] Keccak(UInt16 security, Byte[] MessageB, Byte[] Key)

{

int constant\_index = constant\_index = Array.FindIndex(security\_array, s => s == security);

if (constant\_index == -1) return new Byte[1] { 0x00 };

UInt16 rate\_in\_bytes = rate\_in\_bytes\_array[constant\_index];

UInt64[,] S = new UInt64[5, 5];

for (Byte i = 0; i < 5; ++i)

for (Byte j = 0; j < 5; ++j)

S[i, j] = 0;

int k = 0;

if (Key != null)

{

if (Key.Length != rate\_in\_bytes) return new Byte[1] { 0x00 };

State\_Change(S, Key, rate\_in\_bytes);

}

while (k <= MessageB.Length - rate\_in\_bytes)

{

State\_Change(S, SubArray(MessageB, k, rate\_in\_bytes), rate\_in\_bytes);

k += rate\_in\_bytes;

}

Last\_block\_proc(S, SubArray(MessageB, k, MessageB.Length - k), rate\_in\_bytes);

return Squeezing(S, rate\_in\_bytes).Take(security / 8).ToArray();

}

private static Byte[] Keccak(UInt16 security, FileStream MessageFS, Byte[] Key)

{

int constant\_index = constant\_index = Array.FindIndex(security\_array, s => s == security);

if (constant\_index == -1) return new Byte[1] { 0x00 };

UInt16 rate\_in\_bytes = rate\_in\_bytes\_array[constant\_index];

UInt64[,] S = new UInt64[5, 5];

for (Byte i = 0; i < 5; ++i)

for (Byte j = 0; j < 5; ++j)

S[i, j] = 0;

Byte[] I64 = new Byte[rate\_in\_bytes];

int len;

if (Key != null)

{

if (Key.Length != rate\_in\_bytes) return new Byte[1] { 0x00 };

State\_Change(S, Key, rate\_in\_bytes);

}

while ((len = MessageFS.Read(I64, 0, rate\_in\_bytes)) == rate\_in\_bytes)

{

State\_Change(S, I64, rate\_in\_bytes);

}

Last\_block\_proc(S, SubArray(I64, 0, len), rate\_in\_bytes);

return Squeezing(S, rate\_in\_bytes).Take(security / 8).ToArray();

}

private static UInt64[,] Last\_block\_proc(UInt64[,] State, Byte[] MessageB, UInt16 Rate\_in\_bytes)

{

UInt16 delta = (UInt16)(Rate\_in\_bytes - MessageB.Length);

Byte[] Pad\_Message = new Byte[Rate\_in\_bytes];

int pos = 0;

for (int i = 0; i < MessageB.Length; ++i)

{

Pad\_Message[pos] = MessageB[i];

++pos;

}

if (delta == 1)

{

Pad\_Message[pos] = 0x86;//0x61;//0110 0001

}

else

{

Pad\_Message[pos] = 0x06;//0x60;//0110 0000

delta -= 2;

while (delta > 0)

{

Pad\_Message[pos + delta] = 0x00;

--delta;

}

Pad\_Message[Pad\_Message.Length - 1] = 0x80;//0x01;

}

State\_Change(State, Pad\_Message, Rate\_in\_bytes);

return State;

}

private static UInt64[,] State\_Change(UInt64[,] State, Byte[] Message, UInt16 Rate\_in\_bytes)

{

int pos;

for (Byte ib = 0; ib < 5; ++ib)

for (Byte jb = 0; jb < 5; ++jb)

{

pos = ib + jb \* 5;

// По умолчанию < 9, тк длина каждой подстроки 72 байта или 9 слов по 4 байта

if (pos < (Rate\_in\_bytes / state\_width\_in\_bytes))

{

pos \*= state\_width\_in\_bytes;

for (Byte i = 0; i < state\_width\_in\_bytes; ++i)

{

State[ib, jb] ^= ((UInt64)(Message[pos + i]) << (i \* 8));

}

}

else break;

}

//Keccak\_f

for (Byte i = 0; i < rounds\_number; i++)

State = Round(State, RC[i]);

return State;

}

private static Byte[] SubArray(Byte[] Array, int position, int length)

{

Byte[] subArray = new Byte[length];

for (int i = 0; i < length; ++i)

{

subArray[i] = Array[i + position];

}

return subArray;

}

private static Byte[] Squeezing(UInt64[,] State, UInt16 Rate\_in\_bytes)

{

Byte[] hash = new Byte[Rate\_in\_bytes];

byte[] temp = new Byte[8];

int pos, cur = 0;

for (Byte jb = 0; jb < 5; ++jb)

for (Byte ib = 0; ib < 5; ++ib)

{

pos = ib + jb \* 5;

if (pos < (Rate\_in\_bytes / state\_width\_in\_bytes))

{

temp = BitConverter.GetBytes(State[ib, jb]);

for (int j = 0; j < 8; ++j)

{

hash[cur + j] = temp[j];

}

cur += 8;

}

}

return hash;

}

private static UInt64[,] Round(UInt64[,] A, UInt64 RC\_i)

{

Byte i, j;

//Тета

for (i = 0; i < 5; i++)

C[i] = A[i, 0] ^ A[i, 1] ^ A[i, 2] ^ A[i, 3] ^ A[i, 4];

for (i = 0; i < 5; i++)

D[i] = C[(i + 4) % 5] ^ Rotate(C[(i + 1) % 5], 1, state\_width);

for (i = 0; i < 5; i++)

for (j = 0; j < 5; j++)

A[i, j] = A[i, j] ^ D[i];

//Ро и пи

for (i = 0; i < 5; i++)

for (j = 0; j < 5; j++)

B[j, (2 \* i + 3 \* j) % 5] = Rotate(A[i, j], offset[i, j], state\_width);

//Хи

for (i = 0; i < 5; i++)

for (j = 0; j < 5; j++)

A[i, j] = B[i, j] ^ ((~B[(i + 1) % 5, j]) & B[(i + 2) % 5, j]);

//Йота

A[0, 0] = A[0, 0] ^ RC\_i;

return A;

}

private static UInt64 Rotate(UInt64 x, Byte n, Byte w)

{

return ((x << (n % w)) | (x >> (w - (n % w))));

}

private static Byte[][] Key\_transform(string Key, UInt16 Rate\_in\_bytes)

{

int i;

for (i= Key.Length; i < Rate\_in\_bytes; ++i)

{

Key += "\0";

}

Byte[] KeyB = StringToByte(Key);

Byte[] KeyIp = new Byte[Rate\_in\_bytes];

Byte[] KeyOp = new Byte[Rate\_in\_bytes];

if (KeyB.Length > Rate\_in\_bytes)

{

Byte[] temp = Keccak(512, KeyB, null);

KeyB = new Byte[Rate\_in\_bytes];

for (i = 0; i < temp.Length; ++i)

{

KeyB[i] = temp[i];

}

for (; i < Rate\_in\_bytes; ++i)

{

KeyB[i] = 0x00;

}

}

for (i = 0; i < Rate\_in\_bytes; ++i)

{

KeyIp[i] = (byte)(0x36 ^ KeyB[i]);

KeyOp[i] = (byte)(0x5C ^ KeyB[i]);

}

return new Byte[2][] { KeyIp, KeyOp };

}

private static Byte[] StringToByte(string str)

{

str = str.Replace("\r", "");

Byte[] r = Encoding.UTF8.GetBytes(str);

return r;

}

private static String ByteToHex(Byte[] b)

{

string s = "";

for (Int32 i = 0; i < b.Length; ++i)

s += String.Format("{0:X2}", b[i]);

return s;

}

private static Byte[] HexToByte(string str)

{

try

{

str = str.Trim();

byte[] bytes = new byte[str.Length / 2];

for (int i = 0; i < bytes.Length; ++i)

{

bytes[i] = byte.Parse(str.Substring(i \* 2, 2), NumberStyles.HexNumber);

}

return bytes;

}

catch (Exception)

{

return new Byte[0];

}

}

#region initiating methods

public static string SHA3\_512(string Message)

{

return ByteToHex(Keccak(512, StringToByte(Message), null));

}

public static string SHA3\_512\_HEX(string Message)

{

return ByteToHex(Keccak(512, HexToByte(Message), null));

}

public static string SHA3\_512(FileStream MessageFS)

{

return ByteToHex(Keccak(512, MessageFS, null));

}

public static string SHA3\_384(string Message)

{

return ByteToHex(Keccak(384, StringToByte(Message), null));

}

public static string SHA3\_384\_HEX(string Message)

{

return ByteToHex(Keccak(384, HexToByte(Message), null));

}

public static string SHA3\_384(FileStream MessageFS)

{

return ByteToHex(Keccak(384, MessageFS, null));

}

public static string SHA3\_256(string Message)

{

return ByteToHex(Keccak(256, StringToByte(Message), null));

}

public static string SHA3\_256\_HEX(string Message)

{

return ByteToHex(Keccak(256, HexToByte(Message), null));

}

public static string SHA3\_256(FileStream MessageFS)

{

return ByteToHex(Keccak(256, MessageFS, null));

}

public static string SHA3\_224(string Message)

{

return ByteToHex(Keccak(224, StringToByte(Message), null));

}

public static string SHA3\_224\_HEX(string Message)

{

return ByteToHex(Keccak(224, HexToByte(Message), null));

}

public static string SHA3\_224(FileStream MessageFS)

{

return ByteToHex(Keccak(224, MessageFS, null));

}

public static string SHA3\_HMAC\_512(string Message, string Key)

{

Byte[][] KeyIpOp = Key\_transform(Key, 72);

return ByteToHex(Keccak(512, Keccak(512, StringToByte(Message), KeyIpOp[0]), KeyIpOp[1]));

}

public static string SHA3\_HMAC\_512\_HEX(string Message, string Key)

{

Byte[][] KeyIpOp = Key\_transform(Key, 72);

return ByteToHex(Keccak(512, Keccak(512, HexToByte(Message), KeyIpOp[0]), KeyIpOp[1]));

}

public static string SHA3\_HMAC\_512(FileStream MessageFS, string Key)

{

Byte[][] KeyIpOp = Key\_transform(Key, 72);

return ByteToHex(Keccak(512, Keccak(512, MessageFS, KeyIpOp[0]), KeyIpOp[1]));

}

public static string SHA3\_HMAC\_384(string Message, string Key)

{

Byte[][] KeyIpOp = Key\_transform(Key, 104);

return ByteToHex(Keccak(384, Keccak(384, StringToByte(Message), KeyIpOp[0]), KeyIpOp[1]));

}

public static string SHA3\_HMAC\_384\_HEX(string Message, string Key)

{

Byte[][] KeyIpOp = Key\_transform(Key, 104);

return ByteToHex(Keccak(384, Keccak(384, HexToByte(Message), KeyIpOp[0]), KeyIpOp[1]));

}

public static string SHA3\_HMAC\_384(FileStream MessageFS, string Key)

{

Byte[][] KeyIpOp = Key\_transform(Key, 104);

return ByteToHex(Keccak(384, Keccak(384, MessageFS, KeyIpOp[0]), KeyIpOp[1]));

}

public static string SHA3\_HMAC\_256(string Message, string Key)

{

Byte[][] KeyIpOp = Key\_transform(Key, 136);

return ByteToHex(Keccak(256, Keccak(256, StringToByte(Message), KeyIpOp[0]), KeyIpOp[1]));

}

public static string SHA3\_HMAC\_256\_HEX(string Message, string Key)

{

Byte[][] KeyIpOp = Key\_transform(Key, 136);

return ByteToHex(Keccak(256, Keccak(256, HexToByte(Message), KeyIpOp[0]), KeyIpOp[1]));

}

public static string SHA3\_HMAC\_256(FileStream MessageFS, string Key)

{

Byte[][] KeyIpOp = Key\_transform(Key, 136);

return ByteToHex(Keccak(256, Keccak(256, MessageFS, KeyIpOp[0]), KeyIpOp[1]));

}

public static string SHA3\_HMAC\_224(string Message, string Key)

{

Byte[][] KeyIpOp = Key\_transform(Key, 144);

return ByteToHex(Keccak(224, Keccak(224, StringToByte(Message), KeyIpOp[0]), KeyIpOp[1]));

}

public static string SHA3\_HMAC\_224\_HEX(string Message, string Key)

{

Byte[][] KeyIpOp = Key\_transform(Key, 144);

return ByteToHex(Keccak(224, Keccak(224, HexToByte(Message), KeyIpOp[0]), KeyIpOp[1]));

}

public static string SHA3\_HMAC\_224(FileStream MessageFS, string Key)

{

Byte[][] KeyIpOp = Key\_transform(Key, 144);

return ByteToHex(Keccak(224, Keccak(224, MessageFS, KeyIpOp[0]), KeyIpOp[1]));

}

#endregion

}

}