**Федеральное Государственное Бюджетное**

**Образовательное Учреждение**

**Высшего Профессионального Образования**

**Национальный Исследовательский Университет «МЭИ»**

**Институт информационных и вычислительных технологий**

Кафедра Прикладной Математики

**«Программная реализация алгоритма шифрования на эллиптических кривых»**

**Курсовой проект**

по учебной дисциплине

«Защита данных»

Выполнил:

Крылов К.С.

Преподаватель:

Хорев П.Б.

**Москва 2021**

Оглавление

[ВВЕДЕНИЕ 3](#_Toc501596439)

[1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 4](#_Toc501596440)

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

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

# ВВЕДЕНИЕ

## До того, как криптография на эллиптических кривых стала популярной, почти все алгоритмы с открытым ключом основывались на RSA, DSA и DH, альтернативных криптосистемах на основе модулярной арифметики. Они всё еще популярны, и часто используются вместе с эллиптическими кривыми. Однако несмотря на то, что математика, лежащая в фундаменте RSA и подобных ей алгоритмов легко объяснима и понятна многим, а [грубые реализации пишутся довольно просто](http://code.activestate.com/recipes/578838-rsa-a-simple-and-easy-to-read-implementation/), они обладают рядом недостатков по сравнению с эллиптической криптографией, о которых будет изложено ниже.

Сегодня криптосистемы на эллиптических кривых используются в [TLS](https://tools.ietf.org/html/rfc4492), [PGP](https://tools.ietf.org/html/rfc6637) и [SSH](https://tools.ietf.org/html/rfc5656), важнейших технологиях, на которых базируются современный мир информационных технологий, а также криптовалютах, например Bitcoin.

**Цель** работы – разработка приложения для шифрования текстов по алгоритму ECDH под систему android.

Были поставлены и решены следующие **задачи**:

* проектирование и реализация графического интерфейса;
* проектирование логики взаимодействия компонентов программы;
* реализация основного алгоритма шифрования;
* тестирование и отладка программы.

В **первой** главе будет приведена краткая справка о эллиптических кривых, их основные преимущества и недостатки.

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

## Эллиптические кривые над вещественными числами и групповой закон

## Эллиптической кривой будем считать множество точек, описываемых уравнением y2=x3+ax+b, при этом введем дополнительное условие, что 4a2+27b2 ≠ 0, которое необходимо для того, чтобы исключить сингулярные кривые – кривые, имеющие точки, в которых они не определены.

## Т.к. необходимо чтобы частью кривой являлась бесконечно удаленная точка (которую обозначим символом 0), поэтому полное определение эллиптической кривой:

## { (x, y) ∈ ℝ2 | y2 = x3 + ax + b, 4a2 + 27b2 ≠ 0 } ⋃ { 0 }

## Абелевой группой будем называть множество G, для элементов которого обозначена операция сложения, имеющая свойства:

1. **замыкание:** если a ∈ G и b ∈ G, то a + b ∈ G;
2. **ассоциативность:** (a + b) + c = a + (b + c);
3. **коммутативность:** a + b = b + a;
4. существует **нейтральный элемент** 0, такой, что a + 0 = 0 + a = 0;
5. у каждого элемента a есть **обратная величина – -**a, такая что: a + -a = 0

## Определим Абелеву группу для эллиптических кривых:

1. **Элементы группы –** точки аi на эллиптической кривой.
2. **Операция сложения** задается правилом а1 + а2 + а3 = 0, где точки лежат на одной прямой и их порядок не важен, из чего немедленно следует коммутативность и ассоциативность операции сложения.
3. **Нейтральный элемент** – это бесконечно удаленная точка 0.
4. **Обратная величина точки** – это точка, симметричная данной относительно оси Х.

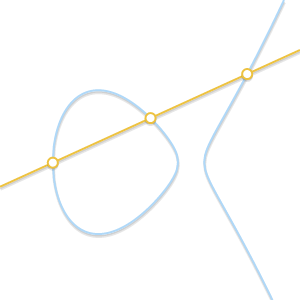


Рис.1.1 – Иллюстрация операции сложения на группе точек эллиптической кривой.

## Любая Абелева группа имеет естественную структуру [модуля над кольцом](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C_%D0%BD%D0%B0%D0%B4_%D0%BA%D0%BE%D0%BB%D1%8C%D1%86%D0%BE%D0%BC) [целых чисел](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BB%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE), поэтому а1 + а2 + а3 = 0 можно записать как а1 + а2 = - а3. Таким образом, геометрически, сумма двух точек а1 и а2 – есть точка, обратная точке а3, через которая проходит прямая, проходящая через а1 и а2.

## 

## Рис.1.2 – Иллюстрация операции сложения на группе точек эллиптической кривой.

## В случае, если а1 или а2 равны нулю мы не можем провести прямую через бесконечно удаленную точку, воспользуемся свойством нейтрального элемента: а1 + 0 = а1.

## Если а2 = -a1 и прямая, проходящие через эти точки вертикальна, и не существует третьей точки пересечения кривой и прямой, можно воспользоваться свойством обратного элемента: a1 + -a1 = 0, т.е. бесконечно удаленная точка.

## Самый сложный случай сложения - a1 + a1, т.к. через одну точку можно провести бесконечно большое количество прямых. В этом случае будем считать, что одна из точек не равна a1, а лишь стремится к ней – в таком случае прямая, проходящая через эти точки будет касательной к эллиптической кривой, тогда a1 + a1 = -a2, где a2 – точка, лежащая на пересечении касательной к эллиптической кривой и самой кривой. Таким же методом можно разрешить ситуацию, когда a1 ≠ a2 но a3 не существует – в этом случае а1 + а2 = - а1.

## 

## Рис.1.3 – Иллюстрация операции сложения в случае отсутствие 3й точки пересечения кривой и прямой.

## Перейдем к алгебраическому методу сложения (нужно отметить, что большая часть ранее рассмотренных исключительные случаи аналогична геометрическому способу и повторно для алгебраического способа рассматриваться не будут).

## Итак, есть две не нулевые точки P(xP, yP) и Q(xQ, yQ). Если они не совпадают (т.е. xP ≠ xQ), то прямая, проходящая через них имеет наклон, . Пересечение этой прямой и кривой – точка R(m2 – xP - xQ, yP + m(xR - xP)), т.е. (xP, yP) + (xQ, yQ) = (xR, -yR)

## В случае, когда P = Q для наклона необходимо использовать уравнение

## Добавим еще одну операцию – скалярное умножение. Скалярное умножение точки на число n есть сложение n таких чисел. Наивная реализация скалярного умножения требует n сложений, который имеет сложность О(2к), где к – количество десятичных разрядов, однако существуют алгоритмы, например алгоритм удвоения сложения[2], имеющий сложность О(k), где k – битовая длина.

## Эллиптические кривые над конечными полями и задача дискретного логарифмирования

## Конечным полем будем называть конечное множество, для которого определены операции сложения, взятия противоположного значения, умножения и деления, при этом умножение и сложение обладают свойствами замкнутости, коммутативности и ассоциативности. Конечным полем, например, является множество чисел по модулю p, где p – простое число. Множество целых чисел с непростым модулем не является полем, так как не имеет операции обратной умножению.

## Теперь ограничим эллиптические кривые конечным полем:

## { (x, y) ∈ F2 | y2 = x3 + ax + b (mod p), 4a2 + 27b2 ≠ 0 (mod p)} ⋃ { 0 },

## Где a и b – целые числа. Все операции будут работать как и в предыдущем параграфе, но теперь используются только целые числа по модулю p. Уточним лишь, что две точки лежат на прямой в конечном поле, если существует прямая, соединяющие эти точки

## 

## Point addition for elliptic curves in Z/p

## Рис 1.4 Иллюстрация суммы двух точек для эллиптической кривой над конечным полем.

## Определим уравнения для суммы P+Q=-R для конечного поля. Если P(xP, yP) ≠ Q(xQ, yQ), то R = (m2 – xP - xQ (mod p), yP + m(xR - xP) (mod p)), где m=( yP - yQ)( xP- xQ)-1 (mod p) и при P = Q m = 3x2P+a) (2yP)-1 (mod p)

## В конечном поле над эллиптической кривой, очевидно, находится конечное количество точек. Количество этих точек называется порядком группы. Вычисление количества точек перебором требует экспоненциального времени работы, но в 1985 году Рене Шуф изобрел алгоритм, работающий за полиномиальное время.

## Скалярное умножение для целых чисел определено ток же, но с модулем как и для вещественных: nP = , но обладает одним важным свойством. Мало того, что количество точек, которые можно получить складывая точки последовательно не может превосходить p2, что следует из p – число вариантов каждой координаты в двумерном пространстве, на самом деле число таких точек может быть существенно меньше p2. Число таких точек будем называть порядком подгруппы, порожденной точкой. Согласно теореме Лагранжа, порядок подгруппы — это делитель порядка исходной группы, то есть если эллиптическая кривая содержит Т точек, а одна из подгрупп – n, то n является делителем N. Таким образом, чтобы найти порядок P группы - минимальное положительное целое n, такое, что nP = 0 – необходимо найти порядок кривой с помощью алгоритма Шуфа, найти все делители этого порядка и выбрать минимальный.

## Для алгоритмов на эллиптических кривых требуются подгруппы с высоким порядком, поэтому обычно выбирается эллиптическая кривая, вычисляется её порядок (N), в качестве порядка группы (n) выбирается большой делитель, а потом находится подходящая базовая точка, а не наоборот. Число h = N/n по теореме Лагранжа всегда целое и имеет свое название – это кофактор группы. Теперь мы можем написать алгоритм поиска базовой точки на прямой:

1. Вычисляем порядок N эллиптической кривой.
2. Выбираем порядок n подгруппы. Чтобы алгоритм сработал, число должно быть простым и быть делителем N.
3. Вычисляем кофактор h =N/n.
4. Выбираем на кривой случайную точку P.
5. Вычисляем G = hP.
6. Если G равно 0, то возвращаемся к шагу 4. В противном случае мы нашли генератор подгруппы с порядком n и кофактором h.

Теперь мы должны обсудить вопрос: **е**сли мы знаем P и Q, то каким будет k, такое, что Q = kP?  
  
Эта задача, известная как **задача дискретного логарифмирования** для эллиптических кривых, считается «сложной», для которой не обнаружено алгоритма полиномиального времени, выполняемого на классическом компьютере. Однако у этой точки зрения нет математических доказательств.  
  
Эта задача аналогична задаче дискретного логарифмирования, используемой в других криптосистемах, таких как DSA, протокол Диффи-Хеллмана и схема Эль-Гамаля. Названия задач совпадают неслучайно. Их разница в том, что в этих алгоритмах используется не скалярное умножение, а возведение в степень по модулю. Их задачу дискретного логарифмирования можно сформулировать так: если известны a и b, то каким будет k , такое, что ak mod(p) = b?  
  
Обе эти задачи дискретны, потому что в них используются конечные множества (а конкретнее — циклические подгруппы). И они являются логарифмическими.  
  
ECC интересна тем, что на сегодняшний момент задача дискретного логарифмирования для эллиптических кривых считается сложнее по сравнению с другими схожими задачами, используемыми в криптографии. Это подразумевает, что нам потребуется меньше бит для целого k, чтобы получить тот же уровень защиты, что и в других криптосистемах.

**ECDH**

Алгоритмы эллиптических кривых будут работать в циклической подгруппе эллиптической кривой над конечным полем. Поэтому алгоритмам потребуются следующие параметры:

* Простое p, задающее размер конечного поля.
* Коэффициенты a и b уравнения эллиптической кривой.
* Базовая точка G, генерирующая подгруппу.
* Порядок n подгруппы.
* Кофактор h подгруппы.

Имея всю необходимую информацию, мы можем приступить к формированию ключей. Закрытый ключ – это случайное целое число d от 1 до n – 1 (n – порядок подгруппы). Открытым ключом будет точка H = dG.

ECDH – протокол согласования ключей. Его алгоритм выглядит следующим образом:

1). Сначала Алиса и Боб генерируют собственные закрытые и открытые ключи. У Алисы есть закрытый ключ dA и открытый ключ HA = dAG, у Боба есть ключи dB и HB. Алиса, и Боб используют одинаковые параметры области определения: одну базовую точку G на одной эллиптической кривой в одинаковом конечном поле.

2). Алиса и Боб обмениваются открытыми ключами HA и HB по незащищённому каналу. Посредник (Man In the Middle) перехватывает HA и HB, но не может определить ни dA, ни dB, не решив задачу дискретного логарифмирования.

Алиса вычисляет S = dAHB (с помощью собственного закрытого ключа и открытого ключа Боба), а Боб вычисляет S = dBHA (с помощью собственного закрытого ключа и открытого ключа Алисы). S одинаков и для Алисы, и для Боба:

S = dAHB = dA(dBG) = dB(dAG) = dBHA

Получив общий секретный ключ, Алиса и Боб могут обмениваться данными с симметричным шифрованием. Например, они могут использовать координату x  ключа  как ключ для шифрования сообщений.

## 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.

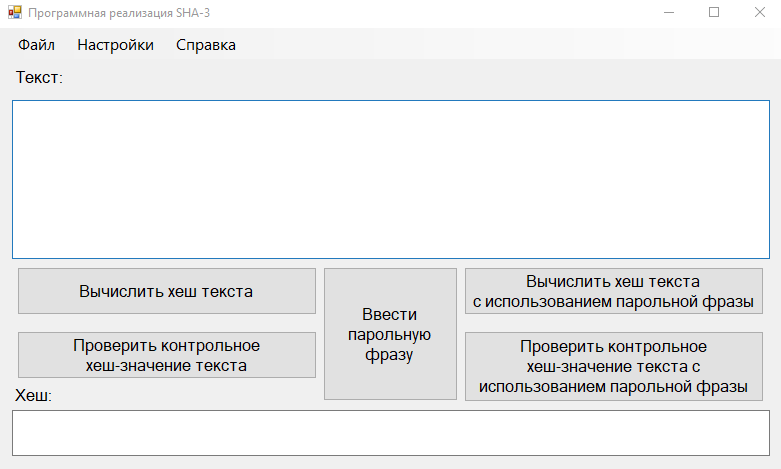
Если парольная фраза не задана и производится попытка вычисления с использованием парольной фразы, то выведется сообщение о необходимости сначала ввести парольную фразу (рис.2.4), после чего откроется окно ввода парольной фразы (рис.2.5). Если парольная фраза не удовлетворяет ограничениям или подтверждение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

Рисунок 2.1. Текстовый режим работы.

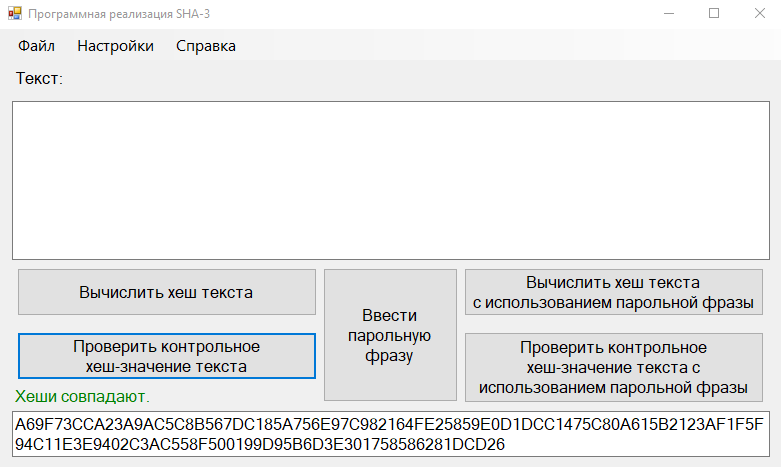
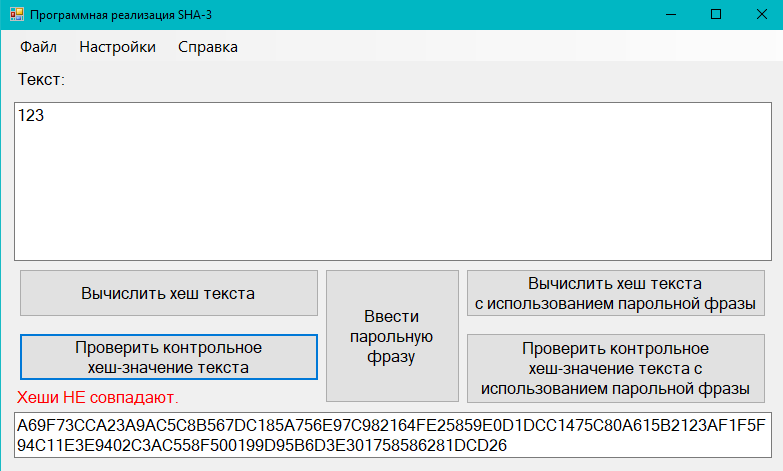
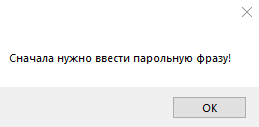


Рисунок 2.3. Отрицательный результат сравнения.

Рисунок 2.2. Положительный результат сравнения.

Рисунок 2.4. Предупреждение.

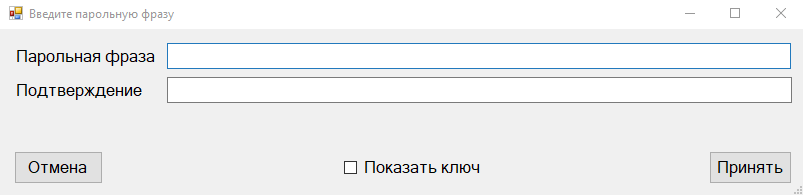
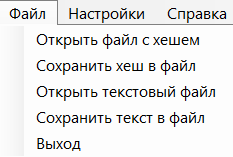
В пункте меню «Файл» доступны возможности: открыть текст или хеш из файла, сохранить текст или хеш в файл, выход из программы (рис.2.6).

Рисунок 2.5. Окно ввода парольной фразы.

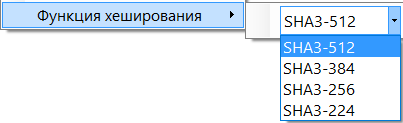
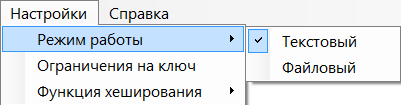
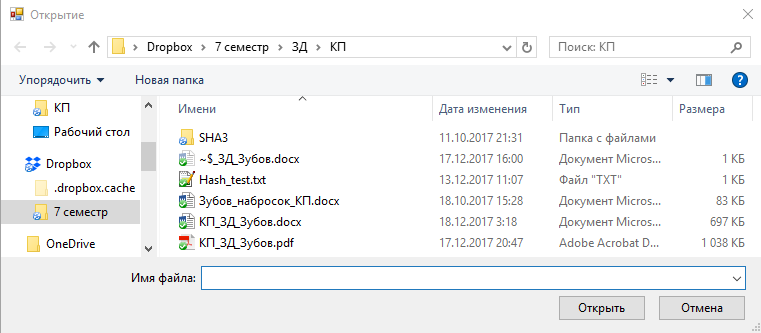
 Пути для работы с файлами запрашиваются при помощи проводника windows (рис.2.7).

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.

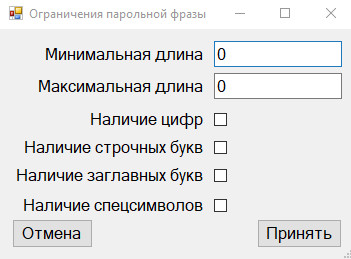
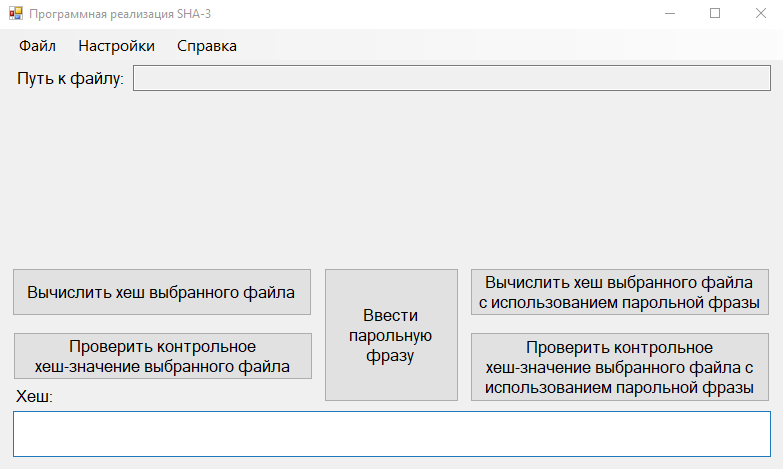
При переходе в файловый режим кнопки меняются на аналогичные кнопкам в текстовом режиме, но предназначенные для работы с файлами, поле для текста скрывается и становится видимым поле выбора пути к файлу, хеш которого требуется находить (рис.2.11).

Рисунок 2.11. Файловый режим работы.

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

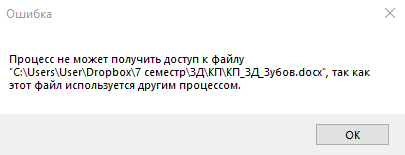
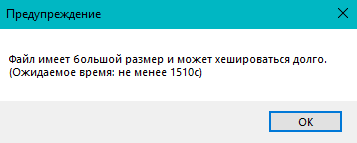
При выборе файла большого размера будет выдано предупреждение о том, что вычисление хеша может быть долгим, с примерной оценкой требующегося времени (рис.2.13).

Рисунок 2.13. Предупреждение.

Рисунок 2.12. Сообщение об ошибке.



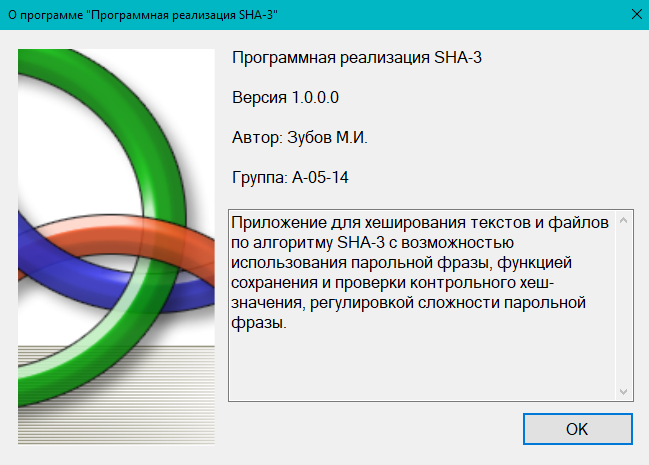
В пункте меню «справка» (рис.2.14) можно посмотреть сведения о программе (рис.2.15).

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.

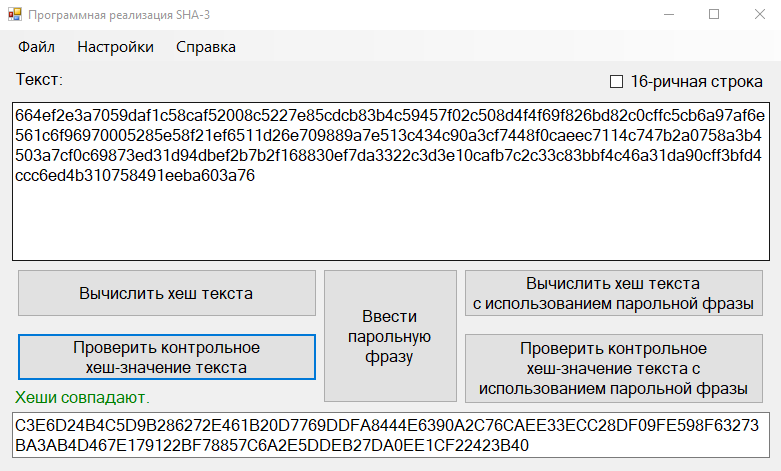


Рисунок 3.1. Тестирование функции проверки хеш-значения.

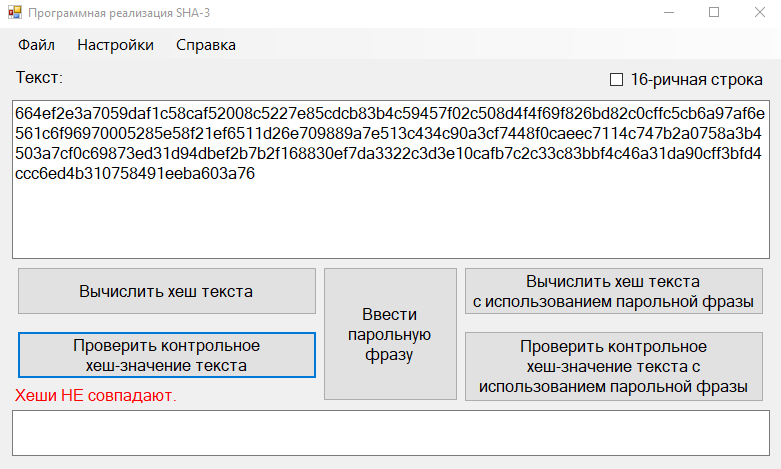


Рисунок 3.2. Тестирование функции проверки хеш-значения.

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

Сгенерируем вручную ключ из парольной фразы для проверки алгоритма.

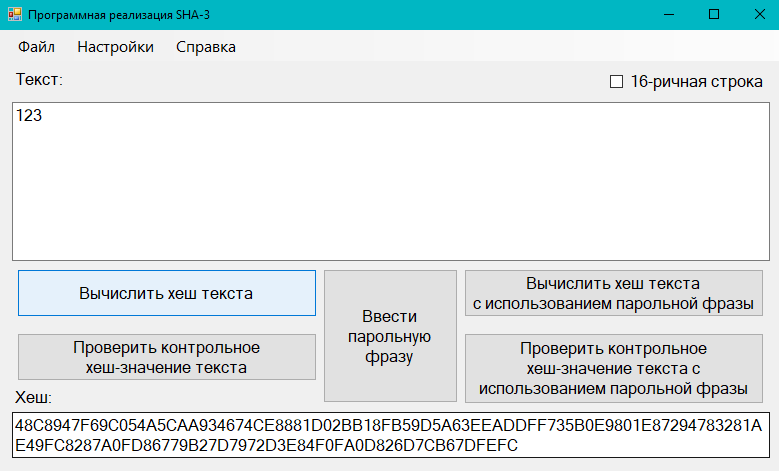


Рисунок 3.3. Ручная генерация ключа.

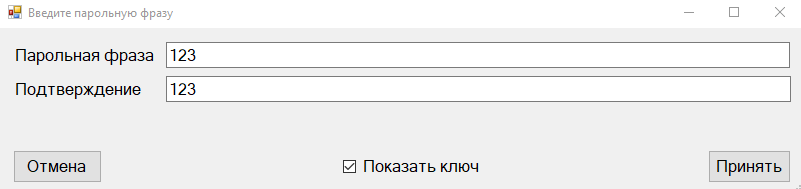


Рисунок 3.4. Ввод парольной фразы.

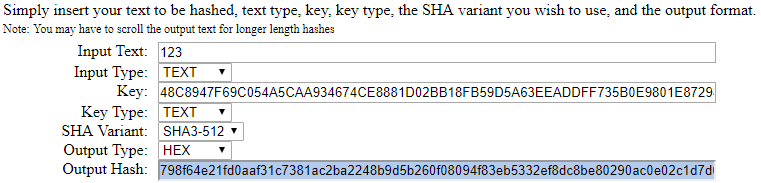


Рисунок 7.5. Генерация хеша с использованием ключа с помощью сторонней реализации.

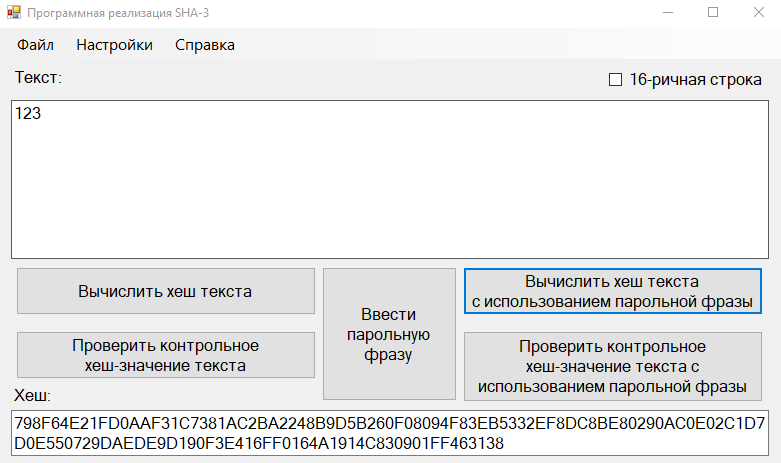


Рисунок 3.6. Генерация хеша с использованием парольной фразы.

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

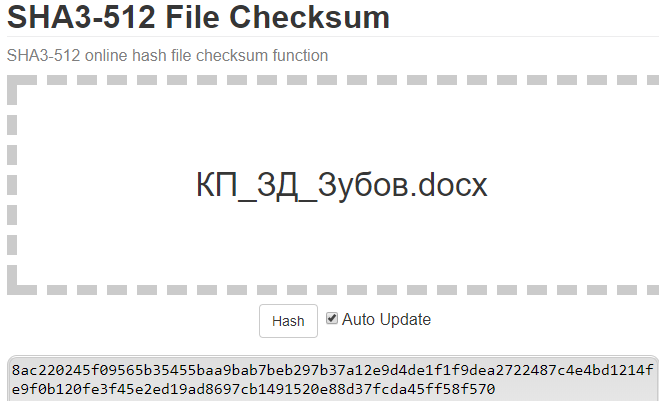


Рисунок 3.7. Генерация хеша файла с помощью сторонней реализации.

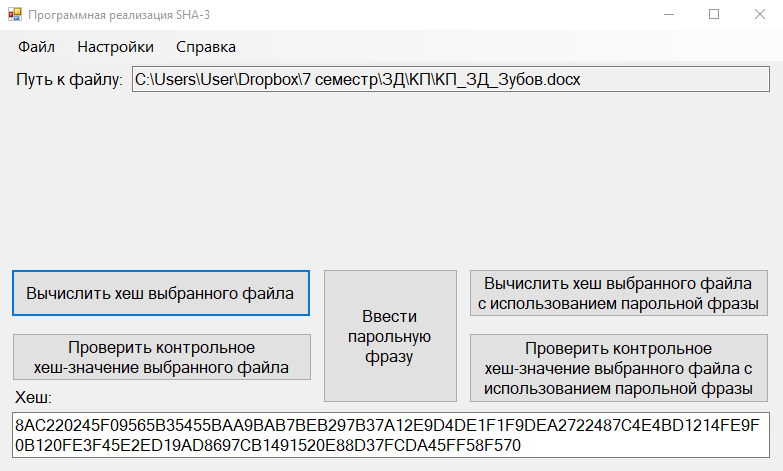


Рисунок 3.8. Генерация хеша файла.

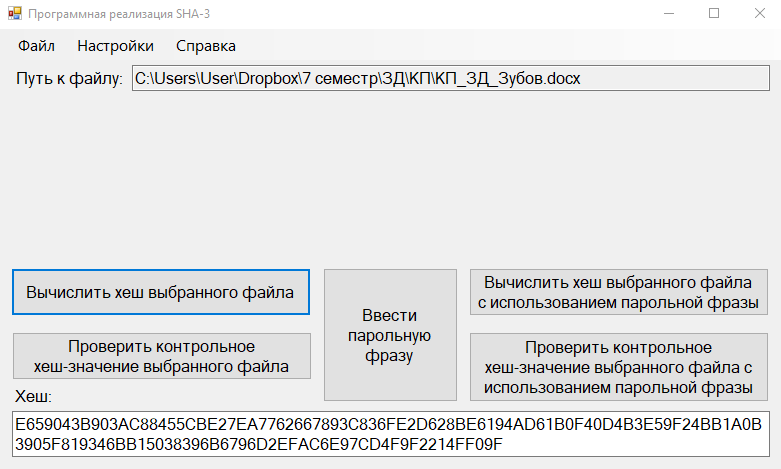
После изменения файла хеш-значение изменилось. 

Рисунок 3.9. Генерация хеша файла после изменения файла.

Проверим работу ограничений парольной фразы.

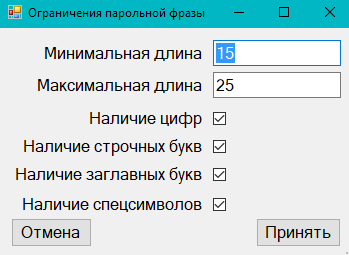


Рисунок 3.10. Ввод ограничений парольной фразы.

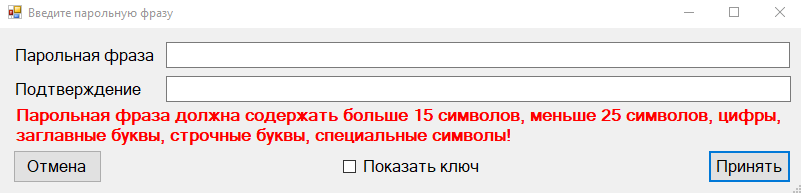


Рисунок 3.11. Проверка работы ограничений парольной фразы.

## 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

}

}