Федеральное государственное бюджетное образовательное учреждение высшего образования Национальный исследовательский университет "МЭИ"  
Кафедра Автоматики и вычислительной техники.

Курсовой проект по дисциплине «Защита данных»

Тема: Программная реализация функции хеширования MD6.

Студент Черток В.Д.

Группа А-05-14

Преподаватель Хорев П.Б.

Москва 2017

Оглавление

[Введение 3](#_Toc498972725)

[Глава 1 Описание алгоритма MD6 3](#_Toc498972726)

[1.1 Входные данные 3](#_Toc498972727)

[1.2 Операции вычисления дерева 4](#_Toc498972728)

[1.3 Графические иллюстрации процесса вычисления 4](#_Toc498972729)

[1.4 Основная процедура вычисления дерева 6](#_Toc498972730)

[1.5 Операция PAR 6](#_Toc498972731)

[1.6 Операция SEQ 7](#_Toc498972732)

[1.7 Графическое представление внутренних параметров 8](#_Toc498972733)

[1.8 Функция сжатия MD6 8](#_Toc498972734)

[Глава 2. Результаты проектирования программы. 10](#_Toc498972735)

[2.1. Результаты проектирования графического интерфейса пользователя. 10](#_Toc498972736)

[2.1.1. Выбор файла. 10](#_Toc498972737)

[2.1.2. Выбор ключа. 11](#_Toc498972738)

[2.1.3. Работа с хеш-значением. 11](#_Toc498972739)

[2.1.4. Возможность сохранения результатов работы в программе. 12](#_Toc498972740)

[2.1.5. Настройки программы. 12](#_Toc498972741)

[2.1.6. Окно справки. 13](#_Toc498972742)

[2.2. Результаты проектирования алгоритма шифрования. 14](#_Toc498972743)

[Глава 3. Результаты тестирования разработанной программы. 15](#_Toc498972744)

[Заключение. 23](#_Toc498972745)

[Список источников 23](#_Toc498972746)

[Приложение 24](#_Toc498972747)

[Исходные коды программы 24](#_Toc498972748)

[MainWindow.xaml 24](#_Toc498972749)

[Options.xaml 33](#_Toc498972750)

[PasswordOptions.xaml 36](#_Toc498972751)

[PasswordInput.xaml 38](#_Toc498972752)

[ComparePasswordInput.xaml 40](#_Toc498972753)

[MD6.cs 42](#_Toc498972754)

[About.xaml 48](#_Toc498972755)

[App.xaml 49](#_Toc498972756)

# Введение

Цель работы - разработка приложения, позволяющего хешировать по алгоритму MD6 как выбираемые произвольные файлы, так и вводимые текстовые сообщения с возможностью сохранения в файле полученного хеш-значения. Должна быть возможность вычисления, сохранения и проверки контрольного хеш-значения, зависящего от хешируемых данных и секретного ключа, выводимого из парольной фразы с регулируемой минимальной длиной и сложностью.

В процессе выполнения работы были поставлены следующие задачи:

* Проектирование графического интерфейса программы
* Изучение и реализация алгоритма хеширования
* Реализация графического интерфейса и интеграция в него алгоритма хеширования
* Отладка графического интерфейса
* Отладка алгоритма хеширования

При выполнении данной работы все указанные задачи были решены.

# Глава 1 Описание алгоритма MD6

Алгоритм хеширования MD6 (MD6 Message-Digest Algorithm) предназначен для создания дайджестов сообщений произвольной длины. Был разработан профессором Рональдом Ривестом из Массачусетского Технологического Института в 2008 году и предлагается на смену алгоритму хеширования MD5.

В процессе хеширования алгоритм использует четырех-арное дерево Меркла, при этом возможно как параллельное, так и последовательное вычисление дерева, в зависимости от параметров алгоритма.

## 1.1 Входные данные

Таблица 1.1 Входные данные

|  |  |
| --- | --- |
| **M** | исходное сообщение длиной m, 1 ≤ m ≤ (264 - 1) бит |
| **d** | длина результирующего хэша в битах, 1 ≤ d ≤ 512 |
| **K** | произвольное ключевое значение длиной keylen байт (0 ≤ keylen ≤ 64), дополненное справа нулями числом в 64 - keylen |
| **L** | неотрицательный параметр размером 1 байт, 0 ≤ L ≤ 64 (по умолчанию L = 64), обозначающий число параллельных вычислений и глубину дерева |
| **r** | неотрицательный параметр размером 12 бит: число раундов (по умолчанию без ключа r = 40 + [d/ 4], |
| **Q** | массив из 15 элементов по 8 байт, являющимися дробной частью корня из 6, его значение указано ниже. |

Таблица 1.2 Значения параметра Q

|  |  |
| --- | --- |
| Q | |
| 0 | 0x7311c2812425cfa0 |
| 1 | 0x6432286434aac8e7 |
| 2 | 0xb60450e9ef68b7c1 |
| 3 | 0xe8fb23908d9f06f1 |
| 4 | 0xdd2e76cba691e5bf |
| 5 | 0x0cd0d63b2c30bc41 |
| 6 | 0x1f8ccf6823058f8a |
| 7 | 0x54e5ed5b88e3775d |
| 8 | 0x4ad12aae0a6d6031 |
| 9 | 0x3e7f16bb88222e0d |
| 10 | 0x8af8671d3fb50c2c |
| 11 | 0x995ad1178bd25c31 |
| 12 | 0xc878c1dd04c4b633 |
| 13 | 0x3b72066c7a1552ac |
| 14 | 0x0d6f3522631effcb |

## 1.2 Операции вычисления дерева

Таблица 1.3 Операции вычисления

|  |  |
| --- | --- |
| **f** | функция сжатия MD6, преобразовывающая 712 байт входных данных в результат размером 128 байт с использованием r раундов вычислений |
| **PAR** | параллельная операция сжатия для каждого уровня дерева, никогда не вызывается при L = 0 |
| **SEQ** | последовательная операция сжатия, никогда не вызывается при L = 64 |

## 1.3 Графические иллюстрации процесса вычисления

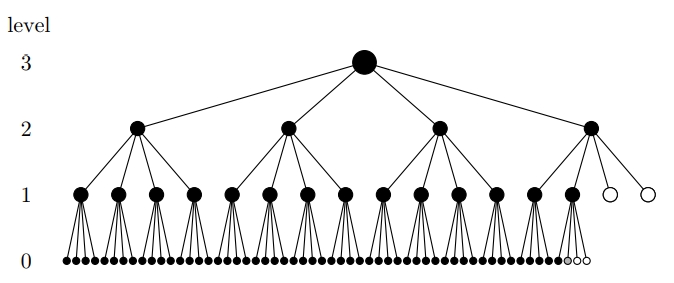


Рисунок .1 Стандартный режим вычисления

Структура процедуры вычисления дерева в стандартном режиме работы (L=64). Дерево вычисляется снизу вверх. Каждая вершина имеет размер в 16 машинных слов. Все вершины на уровне 0 представляют собой фрагмент исходного сообщения длиной 16 слов. Серая вершина дополнена нулями справа до размера в 16 слов. Белая вершина обозначает все нули. Во всех вершинах кроме уровня 0 применяется функция сжатия. Вершина на самом верхнем уровне является желаемым хеш-значением.

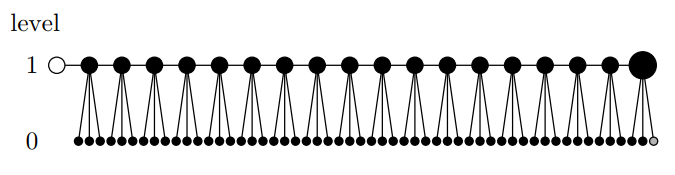


Рисунок 1.2 Последовательный режим вычисления

Структура процедуры вычисления дерева в последовательном режиме (L=0). Вычисление происходит слева на право. Первый уровень представляет собой выполнение операции SEQ. Результирующий хеш будет получен в самой правой вершине первого уровня.

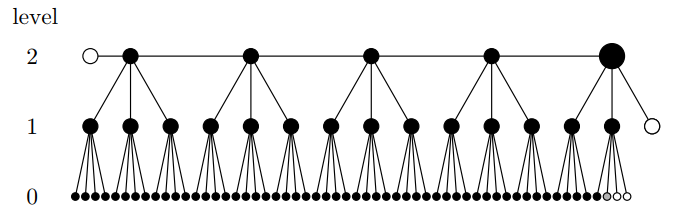


Рисунок 1.3 Ограничение высоты дерева

Структура процедуры вычисления дерева при ограничении на высоту (L=1). Вычисление выполняется снизу вверх, слева на право. Второй уровень представляет собой выполнение операции SEQ. Результирующий хеш будет получен в самой правой вершине второго уровня.

## 1.4 Основная процедура вычисления дерева

**Входные данные**  
Параметры M, d, K, L, r.

**Выходные данные**  
Результирующее хеш-значение размером d бит.

**Тело процедуры**  
**Инициализация**  
Обозначим    
**Основной цикл по уровням дерева**

* *l* = *l* + 1.
* Если *l* = *L* + 1, возвращаем SEQ(*Ml-1*,*d*,*K*,*L*,*r*) в качестве результата.
* *Ml* = PAR(*Ml-1*,*d*,*K*,*L*,*r*,*l*). Обозначим *ml* как длину *Ml* в битах.
* Если *ml* = 8 \* *c* (т.е. длина *Ml* составляет 8 \* *c* байт), Возвращаем последние *d* битов *Ml*. Иначе возвращаемся к началу цикла.

## 1.5 Операция PAR

PAR возвращает сообщение длиной *ml* = 1024 \* max(1, [*ml-1* / 4096])

**Тело процедуры**

* Если требуется, то расширяем *Ml-1*, добавляя справа нулевые биты, пока его длина не станет кратна 512 байт. Теперь разобьём *Ml-1* на блоки *B0*, *B1*, …, *Bj-1*, где *j* = max(1, [*ml-1* / 512]);
* Для каждого блока *Bi*, *i* = 0, 1, …, *j* - 1, параллельно вычисляем *Ci* по следующему алгоритму:
* Обозначим за *p* число дополненных битов *Bi*, 0 ≤ *p* ≤ 4096. (*p* всегда больше нуля для *i*= *j* - 1.);
* Обозначим *z* = 1, если *j* = 1, иначе *z* = 0;
* Обозначим *V* как 8-байтовое значение *r* ǁ *L* ǁ *z* ǁ *p* ǁ *keylen* ǁ *d* таким образом:

1. 4 бита нулей;
2. 12 бит *r*;
3. 8 бит *L*;
4. 4 бита *z*;
5. 16 бит *p*;
6. 8 бит *keylen*.
7. 12 бит *d*.

* Обозначим *U* = *l* \* 256 + *i* – уникальный 8-байтовый идентификатор текущего блока;
* *Ci* = ƒ(*Q* ǁ *K* ǁ *U* ǁ *V* ǁ *Bi*).
* Возвращаем *Ml* = *C0* ǁ *C1* ǁ … ǁ *Cj-1*.

## 1.6 Операция SEQ

SEQ возвращает хэш длиной *d* бит. Данная операция никогда не вызывается, если *L* = 64.

**Тело процедуры**

* Обозначим за C-1 нулевой вектор длиной 128 байт.
* Если требуется, то расширяем *ML*, добавляя справа нулевые биты, пока его длина не станет кратна 384 байт. Теперь разобьём *ML* на блоки *B0*, *B1*, …, *Bj-1*, где *j* = max(1, [*mL* / 384]).
* Для каждого блока *Bi*, *i* = 0, 1, …, *j* - 1, параллельно вычисляем *Ci* по следующему алгоритму:
* Обозначим за *p* число дополненных битов *Bi*, 0 ≤ *p* ≤ 3072. (*p* всегда больше нуля для *i*= *j* - 1.);
* Обозначим *z* = 1, если *i* = *j* - 1, иначе *z* = 0;
* Обозначим *V* как 8-байтовое значение *r* ǁ *L* ǁ *z* ǁ *p* ǁ *keylen* ǁ *d* аналогично предыдущей операции;
* Обозначим *U* = (*L* + 1) \* 256 + *i* – уникальный 8-байтовый идентификатор текущего блока;
* *Ci* = ƒ(*Q* ǁ *K* ǁ *U* ǁ *V* ǁ *Ci-1* ǁ *Bi*).
* Возвращаем последние *d* бит значения *Cj-1* как итоговый хэш.

## 1.7 Графическое представление внутренних параметров

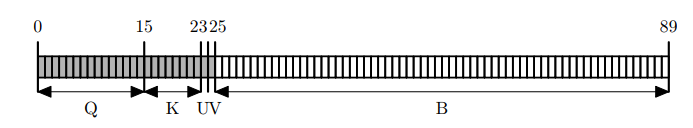


Рисунок 1.4 Входные данные функции сжатия

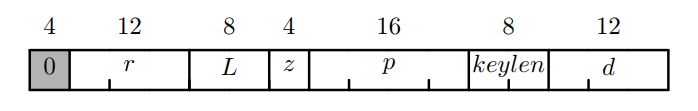


Рисунок 1.5 Структура параметра V

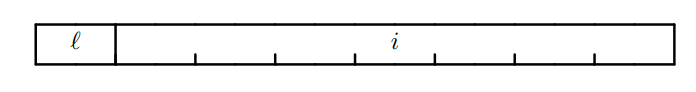


Рисунок 1.6 Структура параметра U

## 1.8 Функция сжатия MD6

**Входные данные**

В качестве входных данных функция принимает массив *N*, состоящий из *n* = 89 8-байтовых слов (712 байт) и число раундов *r*.

**Выходные данные**  
 Функция возвращает массив *C* из 16 элементов по 8 байт.

**Константы**

Таблица 1.4 Константы позиции

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **t0** | **t1** | **t2** | **t3** | **t4** |
| 17 | 18 | 21 | 31 | 67 |

Таблица 1.5 Константы сдвига

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **(*i* - *n*) mod 16** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| *ri-n* | 10 | 5 | 13 | 10 | 11 | 12 | 2 | 7 | 14 | 15 | 7 | 13 | 11 | 7 | 6 | 12 |
| *li-n* | 11 | 24 | 9 | 16 | 15 | 9 | 27 | 15 | 6 | 2 | 29 | 8 | 15 | 5 | 31 | 9 |

*Si-n* = *S|(i-n)/16*  
*S|0* = 0x0123456789abcde  
S\* = 0x7311c2812425cfa0  
*S|j+1* = (*S|j* <<< 1) ⊕ (*S|j* ^ S\*)

**Тело функции**

**Инициализация**

Обозначим *t* = 16*r*. (В каждом раунде будет 16 шагов)  
Обозначим за *A*[0..*t* + *n* - 1] массив из *t* + *n* 8-байтовых элементов. Первые *n* элементов необходимо скопировать из входного массива *N*.

**Основной цикл функции**  
for i = n to t + n - 1: /\* t steps \*/

x = Si-n ⊕ Ai-n ⊕ Ai-t0

x = x ⊕ (Ai-t1 ^ Ai-t2) ⊕ (Ai-t3 ^ Ai-t4)

x = x ⊕ (x >> ri-n)

Ai = x ⊕ (x << li-n)

Возвратить *A*[*t* + *n* - 16 .. *t* + *n* - 1].

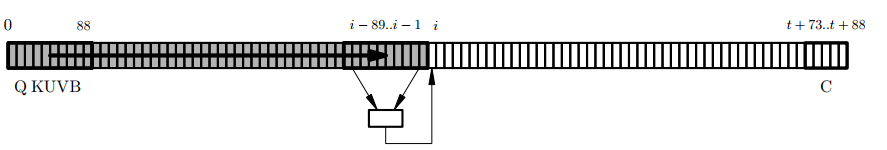


Рисунок 1.7 Основной цикл функции сжатия

Основной цикл функции сжатия. Первый участок массива A[0..88] проинициализирован входным параметром функции. После чего для i = 89, 90, . . . , t + 88, A[i] вычисляется как функция от параметра A[i−89..i−1]. Участок A[i−89..i−1] можно рассматривать как скользящее окно. Последние 16 машинных слов финального окна A[t..t + 88], являющиеся A[t + 73..t + 88], образуют результат функции сжатия.

# Глава 2. Результаты проектирования программы.

## 2.1. Результаты проектирования графического интерфейса пользователя.

Основной интерфейс приложения сгруппирован в главном окне. Визуально его можно разделить на 3 вертикальные части, относящиеся к выбору файла, выбору ключа и работе с хеш-значением соответственно.

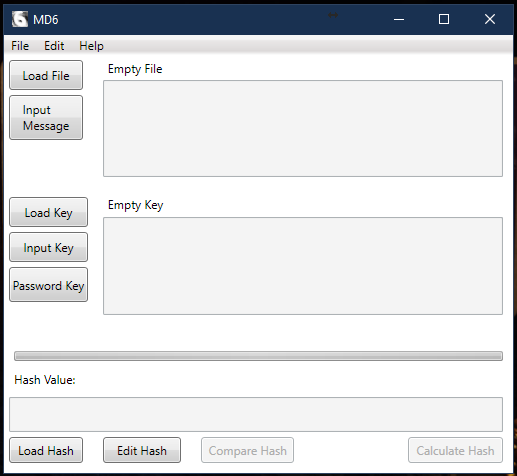


Рисунок .1 Главное окно приложения

### 2.1.1. Выбор файла.

Программа поддерживает следующие возможности:

* Ручной ввод сообщения для хеширования (кнопка Input Message). Введенное сообщение считывается в кодировке UTF-8.
* Загрузка сообщения из файла посредством диалогового окна (кнопка Load File). Загруженный файл будет считан побайтово.

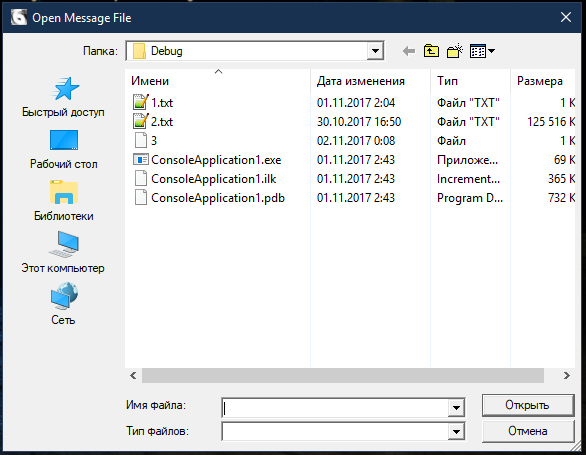


Рисунок 2.2 Диалог выбора файла

### 2.1.2. Выбор ключа.

Программа поддерживает следующие возможности:

* Загрузка ключа из файла посредством диалогового окна (кнопка Load Key). Файл с ключом считывается побайтово.
* Ручной ввод ключа (кнопка Input Key). Ключ считывается в кодировке UTF-8.
* Генерация секретного ключа на основе парольной фразы (кнопка Password Key). Введенный пароль будет захеширован функцией MD6 с использованием текущих настроек хеширующей функции и пустым ключем (входной параметр K). И полученное хеш-значение будет использоваться в роли секретного ключа.

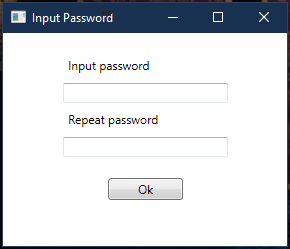


Рисунок 2.3 Диалог ввода парольной фразы

### 2.1.3. Работа с хеш-значением.

Программа поддерживает следующие возможности:

* Вычисление хеш-значения выбранного файла (кнопка Calculate Hash).
* Загрузка хеш значения из файла (кнопка Load Hash). Хеш считывается в кодировке UTF-8.
* Редактирование вычисленного\загруженного хеш-значения (кнопка Edit Hash).
* Сравнение загруженного хеш-значения с вычисляемым хеш-значением файла (кнопка Compare Hash). При нажатии на кнопку, будет предложено ввести пароль, из которого посредством хеширования функцией MD6 генерируется ключ. После чего произойдет вычисление хеш-значения файла с использованием сгенерированного ключа (сгенерированный ключ подается на вход хеширующей функции в качестве параметра K).  
  *Если будет введен пустой пароль, то хеш-значение файла будет вычисляться без использования ключа!*

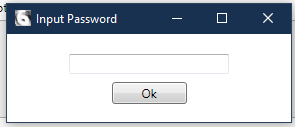


Рисунок 2.4 Диалог ввода парольной фразы при сравнении

Так, из пароля “123” будет сгенерирован секретный ключ с hex значением "4a56c5d61a2bf080e4bb945b0b6cd8a98e8812749dbc104881e35c35e29202d1",  
при использовании параметров хеширования (d=256, L=64, r=default).

### 2.1.4. Возможность сохранения результатов работы в программе.

Данные функции располагаются во вкладке меню File.

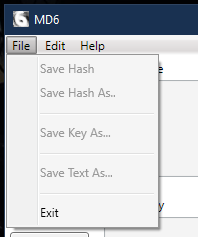


Рисунок 2.5 Меню File

С их помощью возможно:

* Сохранить отредактированное хеш-значение в файл из которого оно было загружено. Хеш-значение сохраняется в виде текста в кодировке UTF-8.
* Сохранить вычисленное хеш-значение в новый файл. Хеш-значение сохраняется в виде текста в кодировке UTF-8.
* Сохранить введенный вручную пользователем ключ в файл. (данная опция не доступна для ключа загруженного из файла и для секретного ключа). Ключ сохраняется в виде текста в кодировке UTF-8.
* Сохранить введенное пользователем сообщение для хеширования в новый файл. Сообщение сохраняется в виде текста в кодировке UTF-8.

### 2.1.5. Настройки программы.

Данные функции располагаются во вкладке меню Edit.

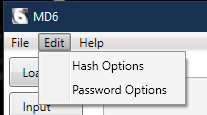


Рисунок 2.6 Меню Edit

С их помощью можно:

* Задать дополнительные параметры функции хеширования.

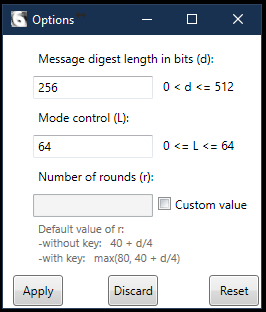


Рисунок 2.7 Настройки алгоритма

* Задать ограничения на сложность пароля, на основе которого генерируется секретный ключ.

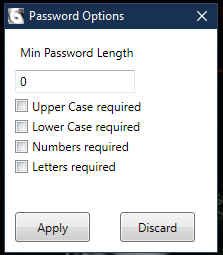


Рисунок 2.8 Настройки пароля

### 2.1.6. Окно справки.

Попасть в него можно через пункт меню Help.

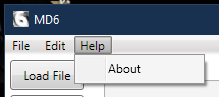


Рисунок 2.9 Меню Help

В нем содержится сведения о программе.

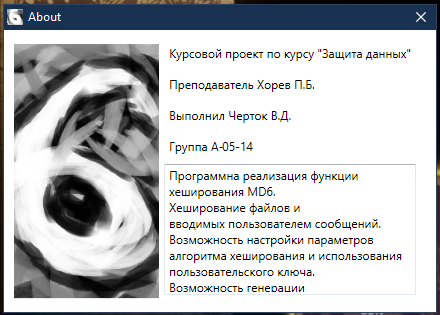


Рисунок 2.10 Окно About

## 2.2. Результаты проектирования алгоритма шифрования.

Алгоритм шифрования реализован наивным образом, именуемым разработчиками как layer-by-layer mode of operation [1, с. 47].

В проекте весь код ответственный за вычисление хеш-значения вынесен в отдельный класс под названием MD6.

Основные параметры алгоритма задаются при вызове конструктора класса MD6.

При этом класс реализует следующие методы для считывания файла и ключа: readMessageFile, readMessageString, readKeyFile, readKeyString. Необходимо явно вызвать пару из этих методов в зависимости от поданного пользователем значения ключа и файла, перед запуском вычисления хеш-значения.

Непосредственный запуск вычисления хеш-значения реализован в методе Hash, который внутри оперирует методами ModeOfOperation, PAR, SEQ, Compress в соответствии с описанием алгоритма из главы 1.

Сам алгоритм хеширования запускается в асинхронном потоке реализуемым системным классом BackgroundWorker. Это позволило добавить в основное окно динамический индикатор, показывающий процесс вычисления хеш-значения, и оставить отзывчивость интерфейса во время работы алгоритма.

# Глава 3. Результаты тестирования разработанной программы.

Программа разрабатывалась и отлаживалась на языке программирования C#, ориентированного на .NET Framework 3.5 в среде программирования Microsoft Visual Studio 2015.

Сравним работу программы с авторской реализацией алгоритма [5] на примерах предложенных в [1, с. 202].

**Пример 1**

Хеширование сообщения “abc” с использованием 5 раундов  
Результат авторской реализации:

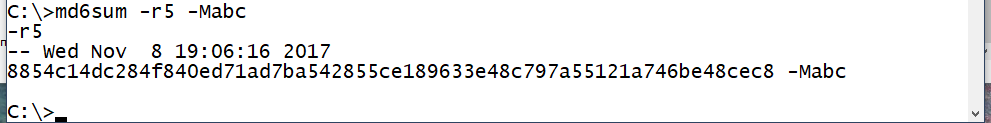


Рисунок 3.1 Пример 1 - Авторская реализация

Результат программы:

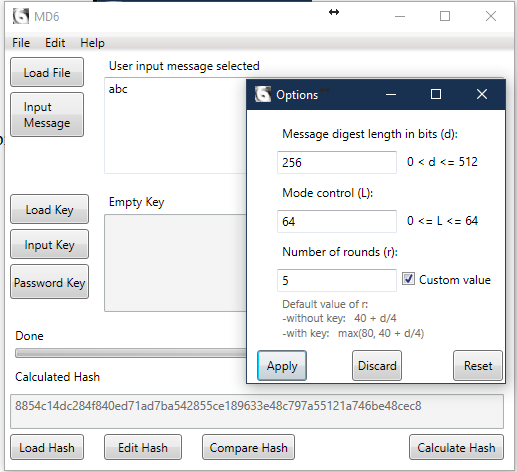


Рисунок 3.2 Пример 1 - Результат программы

**Пример 2**

Хеширование сообщения представимого в 16-ричной с.с. в виде последовательности “11223344556677112233...” длиной 600 символов, с ключем “abcde12345”, размером хеш-значения в 224бита, и использованием 5ти раундов вычисления.

Структура сообщения:  


Рисунок 3.3 Пример 2 - Структура сообщения

Результат авторской реализации:

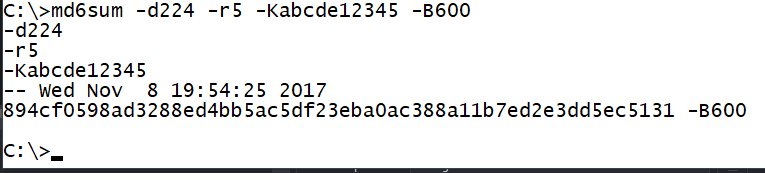


Рисунок 3.4 Пример 2 - Авторская реализация

Результат программы:

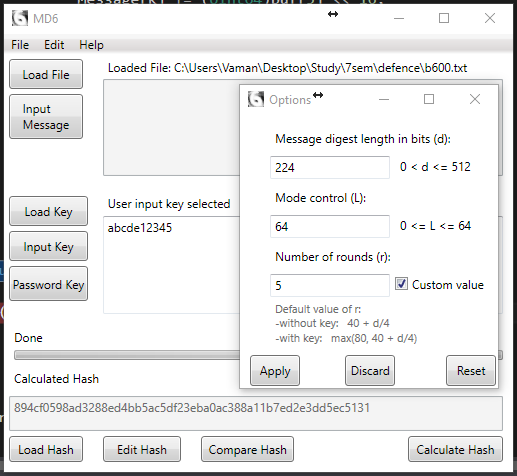


Рисунок 3.5 Пример 2 - Результат программы

**Пример 3**

Хеширование 800 символьного сообщения той же структуры, что и во втором примере, в последовательном режиме (L=0)

Результат авторской реализации:

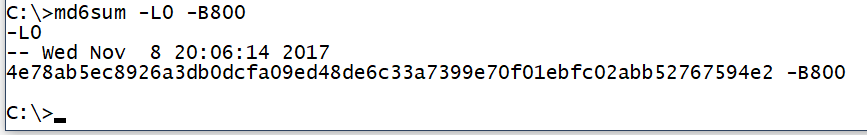


Рисунок 3.6 Пример 3 - Авторская реализация

Результат программы:

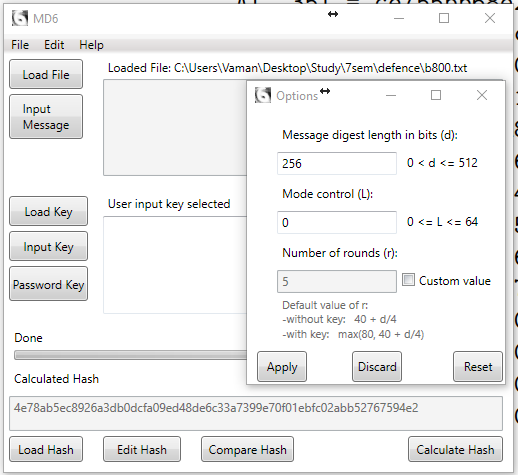


Рисунок 3.7 Пример 3 - Результат программы

**Тестирование сравнения хеш-значения**

Загрузим в программу файл из второго примера, сгенерируем ключ на основе пароля “123”, выставим в параметрах хеш функции значения по молчанию, и вычислим хеш-значение.

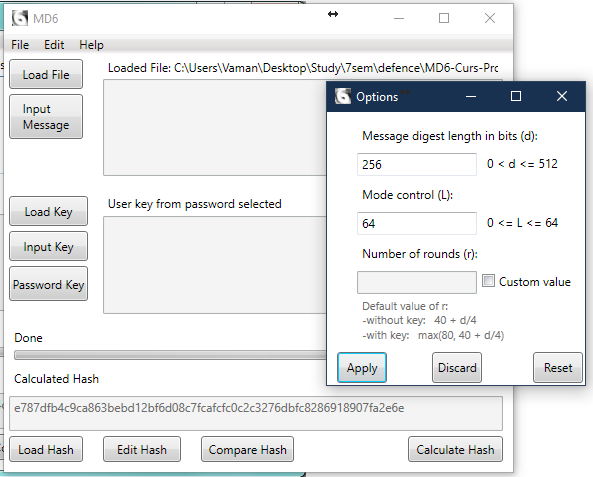


Рисунок 3.8 Сравнение хеш -значения - Первое вычисление

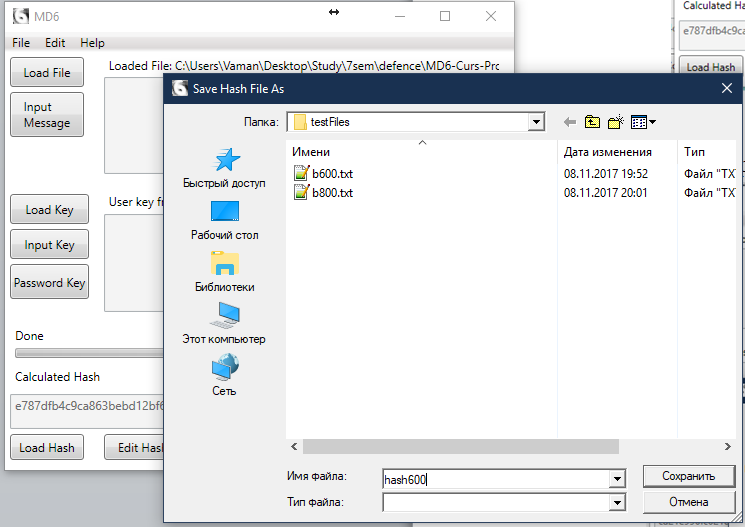
Сохраним полученное хеш-значение в файл.  


Рисунок 3.9 Сравнение хеш -значения - Сохранение хеш-значения

Перезапустим программу, заново загрузим файл из второго примера.

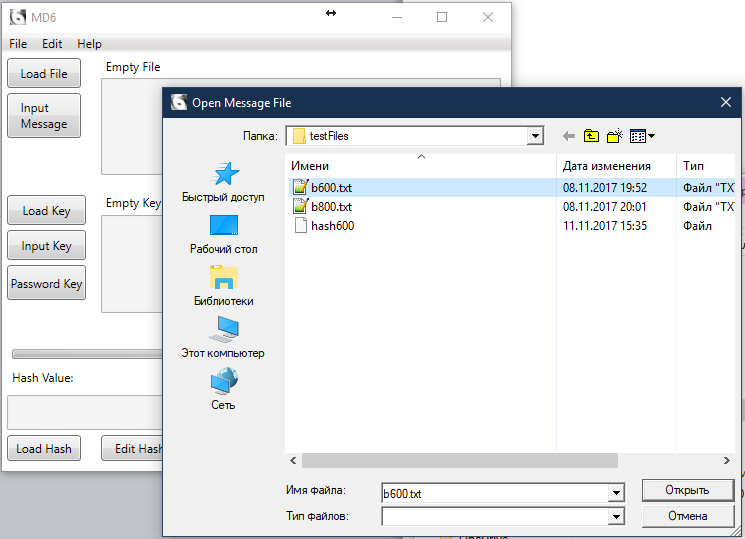


Рисунок .10 Сравнение хеш -значения - Загрузка файла

Загрузим сохраненное хеш-значение

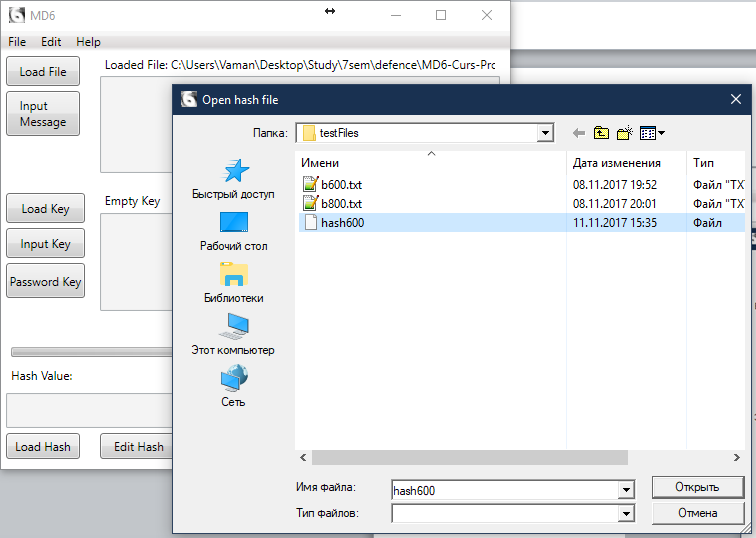


Рисунок 3.11 Сравнение хеш -значения - Загрузка хеш-значения

Сравним загруженное хеш-значение, используя пароль “123”.

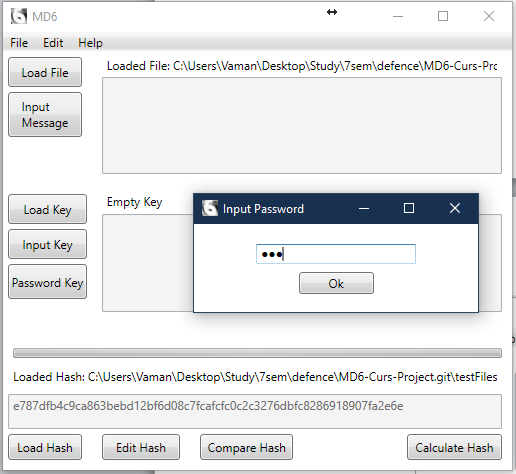


Рисунок 3.12 Сравнение хеш -значения - Ввод пароля

Результат сравнения

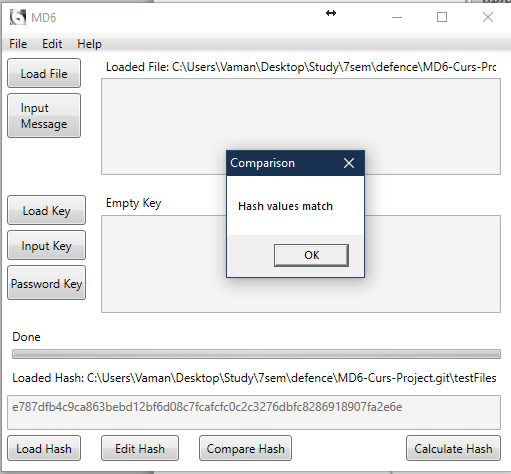


Рисунок 3.13 Сравнение хеш -значения - Первый результат

Теперь вручную изменим загруженное хеш-значение и попробуем сравнить

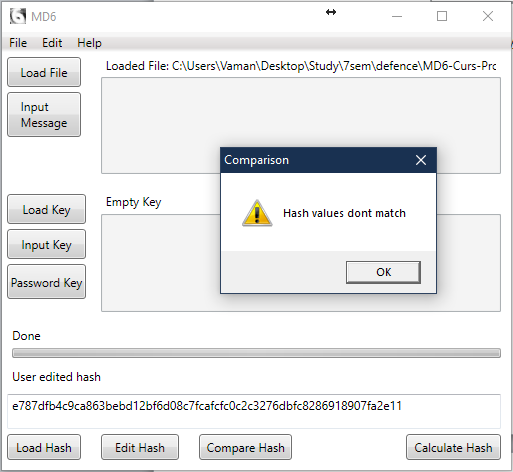


Рисунок 3.14 Сравнение хеш -значения - Измененное хеш-значение

Заново загрузим корректное хеш-значение из файла, но теперь сгенерируем секретный ключ из неправильного пароля, и попробуем сравнить.

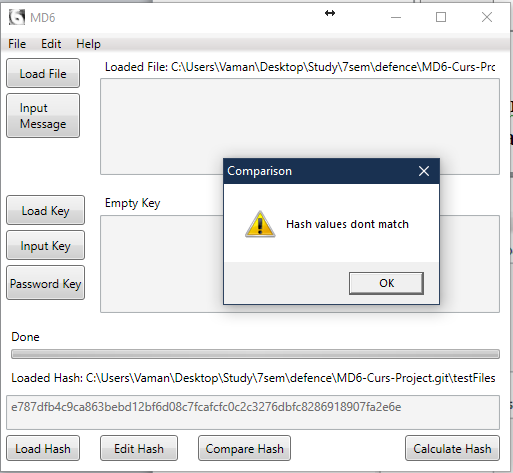


Рисунок 3.15 Сравнение хеш -значения - Неправильный пароль

В качестве дополнительного теста, вычислим хеш-значение файла из примера 3 и сравним его с самим собой, без использования ключа.

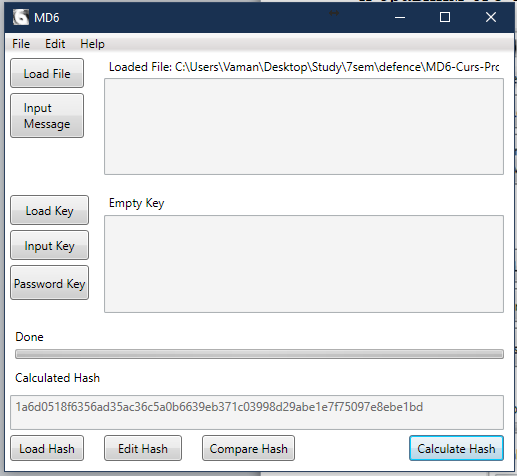


Рисунок 3.16 Сравнение хеш -значения - Вычисление файла из примера 3

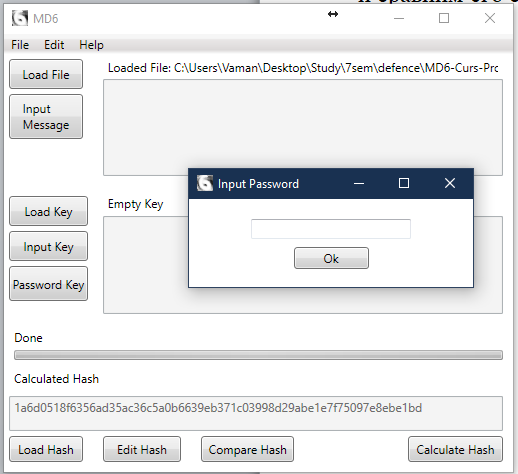


Рисунок 3.17 Сравнение хеш -значения - Ввод пустого пароля

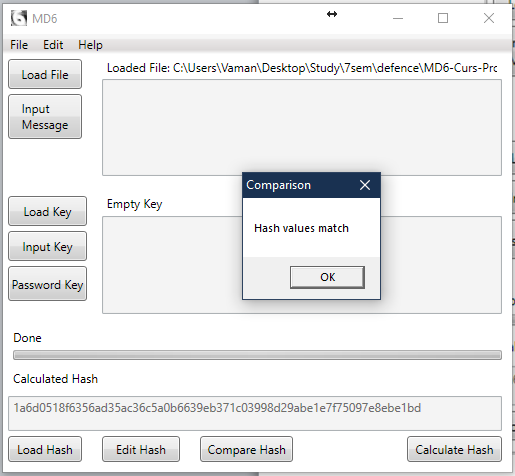


Рисунок 3.18 Сравнение хеш -значения - Результат сравнения с самим собой

Тестирование и отладка графического интерфейса пользователя позволили исключить выполнение пользователем некорректных в точки зрения логики программы действия, в том числе не предусмотренные возможности параллельного запуска алгоритма хеширования, и некорректное обновление активности пунктов меню.

# Заключение.

В результате выполнения данной работы было разработано программное обеспечение, позволяющее хешировать по алгоритму MD6 как выбираемые произвольные файлы, так и вводимые текстовые сообщения с возможностью сохранения в файле полученного хеш-значения. И возможностью вычисления, сохранения и проверки контрольного хеш-значения, зависящего от хешируемых данных и секретного ключа, выводимого из парольной фразы с регулируемой минимальной длиной и сложностью.

При реализации крипто алгоритма были изучены основы ассиметричного программирования на примере использования класса BackgroundWorker.

При реализации графического интерфейса были изучены основы WPF и платформы .Net.

# Список источников

1. Ronald Rivest, Benjamin Agre, Daniel Bailey, Christopher Crutchfield, Yevgeniy Dodis, Kermin Elliott Fleming, Asif Khan, Jayant Krishnamurthy, Yuncheng Lin, Leo Reyzin, Emily Shen, Jim Sukha, Drew Sutherland, Eran Tromer, and Yiqun Lisa yin. The MD6 Hash Function: a Proposal to NIST for SHA-3, 2008.

2. Описание алгоритма хеширования MD6 в русской Википедии  
https://ru.wikipedia.org/wiki/MD6

3. Описание алгоритма хеширования MD6 в английской Википедии  
https://en.wikipedia.org/wiki/MD6

3. Описание дерева Меркла  
https://ru.wikipedia.org/wiki/TTH

4. Документация Microsoft о классе BackgroundWorker  
https://msdn.microsoft.com/ru-ru/library/system.componentmodel.backgroundworker(v=vs.110).aspx

5. Программная реализация алгоритма MD6  
http://groups.csail.mit.edu/cis/md6/downloads.html

# Приложение

## Исходные коды программы

### MainWindow.xaml

<Window x:Class="MD6Project.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:MD6Project"

mc:Ignorable="d"

Title="MD6" Height="477" Width="525" Loaded="Window\_Loaded" Closed="Window\_Closed" Icon="IconMain.ico">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="0\*"/>

<ColumnDefinition Width="90\*"/>

<ColumnDefinition Width="100\*"/>

<ColumnDefinition Width="105\*"/>

<ColumnDefinition Width="105\*"/>

<ColumnDefinition Width="117\*"/>

<ColumnDefinition Width="0\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="155\*"/>

<RowDefinition Height="140\*"/>

<RowDefinition Height="151\*"/>

</Grid.RowDefinitions>

<Menu x:Name="menu" Height="20" VerticalAlignment="Top" Grid.Column="1" Grid.ColumnSpan="5">

<MenuItem x:Name="menuFileButton" Header="\_File">

<MenuItem x:Name="menuFileSaveHashButton" Header="Save Hash" HorizontalAlignment="Left" Width="165" Margin="0,0,-20.493,0" IsEnabled="False" Click="menuFileSaveHashButton\_Click"/>

<MenuItem x:Name="menuFileSaveHashAsButton" Header="Save Hash As..." HorizontalAlignment="Left" Width="165" Margin="0,0,-20.493,0" IsEnabled="False" Click="menuFileSaveHashAsButton\_Click"/>

<Separator/>

<MenuItem x:Name="menuFileSaveKeyAsButton" Header="\_Save Key As..." IsEnabled="False" Margin="0,0,-20.461,0" Click="menuFileSaveKeyAsButton\_Click"/>

<Separator/>

<MenuItem x:Name="menuFileSaveTextAsButton" Header="Save Text As..." HorizontalAlignment="Left" Width="165" Margin="0,0,-20.493,0" IsEnabled="False" Click="menuFileSaveTextAsButton\_Click"/>

<Separator/>

<MenuItem x:Name="menuFileExitButton" Header="\_Exit" Margin="0,0,-20.493,0" Click="menuFileExitButton\_Click"/>

</MenuItem>

<MenuItem x:Name="MenuEditButton" Header="\_Edit">

<MenuItem x:Name="MenuEditOptionsButton" Header="Hash Options" HorizontalAlignment="Left" Width="188.05" Click="MenuEditOptionsButton\_Click" Margin="0,0,-48.543,0"/>

<MenuItem x:Name="MenuEditPasswordoptions" Header="Password Options" HorizontalAlignment="Left" Width="188.05" Margin="0,0,-48.543,0" Click="MenuEditPasswordoptions\_Click"/>

</MenuItem>

<MenuItem x:Name="MenuHelpButton" Header="\_Help">

<MenuItem x:Name="menuHelpAboutButton" Header="\_About" Click="menuHelpAboutButton\_Click"/>

</MenuItem>

</Menu>

<TextBox x:Name="textFileToHash" Margin="10,45,10,10" TextWrapping="Wrap" Grid.Column="2" IsEnabled="False" Grid.ColumnSpan="4"/>

<Button x:Name="openFileToHashButton" Content="Load File" Margin="5,25,10,0" VerticalAlignment="Top" Click="openFileToHash\_Click" Height="30" Grid.Column="1"/>

<Button x:Name="openKeyButton" Content="Load Key" Margin="5,10,5,0" Grid.Row="1" RenderTransformOrigin="0.267,0.501" Height="30" VerticalAlignment="Top" Click="openKeyButton\_Click" Grid.Column="1"/>

<TextBox x:Name="textKey" Grid.Column="2" Margin="10,30,10,10" Grid.Row="1" TextWrapping="Wrap" IsEnabled="False" Grid.ColumnSpan="4"/>

<Label x:Name="labelFileToHashname" Content="Empty File" Grid.Column="2" Margin="10,20,10,0" Height="25.96" VerticalAlignment="Top" Grid.ColumnSpan="4"/>

<Label x:Name="labelKey" Content="Empty Key" Grid.Column="2" Margin="10,4.04,10,0" Grid.Row="1" Height="25.96" VerticalAlignment="Top" Grid.ColumnSpan="4"/>

<Button x:Name="InputMessageButton" Content="Input &#xD;&#xA;Message" Margin="5,60,10,0" VerticalAlignment="Top" Height="45" Click="InputMessageButton\_Click" Grid.Column="1"/>

<Button x:Name="InputkeyButton" Content="Input Key" Margin="5,45,5,0" Grid.Row="1" Click="InputkeyButton\_Click" Height="30" VerticalAlignment="Top" Grid.Column="1"/>

<ProgressBar x:Name="MD6ProgressBar" Grid.Column="1" Margin="10,25.96,10,0" Grid.Row="2" Grid.ColumnSpan="5" Height="10" VerticalAlignment="Top"/>

<Label x:Name="MD6ProgressBarLabel" Grid.ColumnSpan="5" Content="" Margin="5,0,0,0" Grid.Row="2" Height="25.96" VerticalAlignment="Top" HorizontalAlignment="Left" Width="502" Grid.Column="1"/>

<TextBox x:Name="MD6Hash" Grid.ColumnSpan="5" Margin="5,71.92,10,41" Grid.Row="2" TextWrapping="Wrap" Grid.Column="1" IsEnabled="False"/>

<Button x:Name="LoadHashButton" Content="Load Hash" Margin="5,0,10,10" Grid.Row="2" Height="26" VerticalAlignment="Bottom" Grid.Column="1" Click="LoadHashButton\_Click"/>

<Button x:Name="CalculateHashButton" Content="Calculate Hash" Grid.Column="5" Margin="10,0,10,10" Grid.Row="2" Height="26" VerticalAlignment="Bottom" Click="CalculateHashButton\_Click" IsEnabled="False"/>

<Label x:Name="MD6HashResultLabel" Grid.ColumnSpan="5" Content="Hash Value:" Margin="5,40.96,10,0" Grid.Row="2" Height="25.96" VerticalAlignment="Top" Grid.Column="1"/>

<Button x:Name="EditHashbutton" Content="Edit Hash" Grid.Column="2" Margin="10,0,10,10" Grid.Row="2" VerticalAlignment="Bottom" Height="26" Click="EditHashbutton\_Click"/>

<Button x:Name="PasswordKeyButton" Grid.ColumnSpan="2" Content="Password Key" Margin="5,80,5,0" Grid.Row="1" Height="35" VerticalAlignment="Top" Click="PasswordKeyButton\_Click"/>

<Button x:Name="CompareHashButton" Content="Compare Hash" Grid.Column="3" Margin="10,0,0,10" Grid.Row="2" Height="26" VerticalAlignment="Bottom" Click="CompareHashButton\_Click" IsEnabled="False"/>

</Grid>

</Window>

#### MainWindow.xaml.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using System.IO;

using Microsoft.Win32;

using System.ComponentModel;

namespace MD6Project

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public static MainWindow Instance { get; private set; }

public BackgroundWorker bw;

MD6Project.Options opt;

MD6Project.PasswordOptions PassOpt;

static MainWindow()

{

Instance = new MainWindow();

}

private MainWindow()

{

InitializeComponent();

messageFile = "";

keyFile = "";

hashFile = "";

PasswordString = "";

opt = MD6Project.Options.Instance;

PassOpt = MD6Project.PasswordOptions.Instance;

bw = new BackgroundWorker();

bw.WorkerSupportsCancellation = true;

bw.DoWork +=

new DoWorkEventHandler(bw\_DoWork);

}

private void Window\_Loaded(object sender, RoutedEventArgs e)

{

System.Windows.Forms.Application.EnableVisualStyles();

}

private void menuHelpAboutButton\_Click(object sender, RoutedEventArgs e)

{

About about = new About();

about.ShowDialog();

about = null;

}

private void openFileToHash\_Click(object sender, RoutedEventArgs e)

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Title = "Open Message File";

if (openFileDialog.ShowDialog() == true)

{

FileInfo fileInf = new FileInfo(openFileDialog.FileName);

if (fileInf.Exists)

{

labelFileToHashname.Content = "Loaded File: " + openFileDialog.FileName;

messageFile = openFileDialog.FileName;

CalculateHashButton.IsEnabled = true;

textFileToHash.IsEnabled = false;

CompareHashButton.IsEnabled = true;

menuFileSaveTextAsButton.IsEnabled = false;

}

else

{

System.Windows.MessageBox.Show("File does not exist", "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

}

}

private void openKeyButton\_Click(object sender, RoutedEventArgs e)

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Title = "Open Key File";

if (openFileDialog.ShowDialog() == true)

{

FileInfo fileInf = new FileInfo(openFileDialog.FileName);

if (fileInf.Exists)

{

if(fileInf.Length > 512)

{

System.Windows.MessageBox.Show("Key file too big.\nExpected length <= 512 bytes.", "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

textKey.IsEnabled = false;

menuFileSaveKeyAsButton.IsEnabled = false;

labelKey.Content = "Loaded Key: " + openFileDialog.FileName;

keyFile = openFileDialog.FileName;

PasswordString = "";

}

else

{

System.Windows.MessageBox.Show("File does not exist", "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

}

}

private void InputkeyButton\_Click(object sender, RoutedEventArgs e)

{

textKey.IsEnabled = true;

menuFileSaveKeyAsButton.IsEnabled = true;

labelKey.Content = "User input key selected";

keyFile = "";

PasswordString = "";

}

private void InputMessageButton\_Click(object sender, RoutedEventArgs e)

{

CalculateHashButton.IsEnabled = true;

textFileToHash.IsEnabled = true;

menuFileSaveTextAsButton.IsEnabled = true;

labelFileToHashname.Content = "User input message selected";

messageFile = "";

if(MD6Hash.IsEnabled || (MD6Hash.Text != ""))

CompareHashButton.IsEnabled = true;

}

private void LoadHashButton\_Click(object sender, RoutedEventArgs e)

{

MD6ProgressBarLabel.Content = "";

menuFileSaveHashButton.IsEnabled = false;

menuFileSaveHashAsButton.IsEnabled = false;

MD6Hash.IsEnabled = false;

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Title = "Open hash file";

if (openFileDialog.ShowDialog() == true)

{

FileInfo fileInf = new FileInfo(openFileDialog.FileName);

if (fileInf.Exists)

{

if (fileInf.Length > 512)

{

System.Windows.MessageBox.Show("Hash file too big.\nExpected length <= 512 bytes.", "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

MD6HashResultLabel.Content = "Loaded Hash: " + openFileDialog.FileName;

MD6Hash.Text = File.ReadAllText(openFileDialog.FileName);

hashFile = openFileDialog.FileName;

if(CalculateHashButton.IsEnabled)

CompareHashButton.IsEnabled = true;

}

else

{

System.Windows.MessageBox.Show("Hash File does not exist", "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

}

}

private void CalculateHashButton\_Click(object sender, RoutedEventArgs e)

{

if (bw.IsBusy != true)

{

System.GC.Collect();

bw.RunWorkerCompleted +=

new RunWorkerCompletedEventHandler(bw\_RunWorkerCompleted);

MD6ProgressBarLabel.Content = "Calculating";

MD6HashResultLabel.Content = "";

MD6Hash.Text = "";

MD6ProgressBar.IsIndeterminate = true;

List<object> arguments = new List<object>();

arguments.Add(Options.Instance.dVal);

arguments.Add(Options.Instance.Lval);

arguments.Add(Options.Instance.rVal);

arguments.Add(messageFile);

arguments.Add(textFileToHash.Text);

arguments.Add(keyFile);

if (PasswordString != "")

{

arguments.Add(PasswordString);

arguments.Add(true);//Flag indicates that string is hex value

}

else

{

arguments.Add(textKey.Text);

arguments.Add(false);//Flag indicates that string is not hex value

}

bw.RunWorkerAsync(arguments);

}

}

private void EditHashbutton\_Click(object sender, RoutedEventArgs e)

{

if (hashFile != "")

{

menuFileSaveHashButton.IsEnabled = true;

}

menuFileSaveHashAsButton.IsEnabled = true;

MD6ProgressBarLabel.Content = "";

MD6Hash.IsEnabled = true;

MD6HashResultLabel.Content = "User edited hash";

CompareHashButton.IsEnabled = true;

}

private void bw\_DoWork(object sender, DoWorkEventArgs e)

{

try

{

List<object> genericlist = e.Argument as List<object>;

uint d = (uint)Convert.ToInt32(genericlist[0]);

uint L = (uint)Convert.ToInt32(genericlist[1]);

uint r = (uint)Convert.ToInt32(genericlist[2]);

string messageFilepath = (string)genericlist[3];

string messageText = (string)genericlist[4];

string keyFilepath = (string)genericlist[5];

string keyText = (string)genericlist[6];

bool isHexString = (bool)genericlist[7];

MD6 HashFunction = new MD6(d, L, r);

try

{

if (messageFilepath == "")

{

HashFunction.readMessageString(messageText);

}

else

{

if (HashFunction.readMessageFile(messageFilepath) != MD6.OK)

{

throw new Exception("Message File does not exist");

}

}

}

catch(Exception ex)

{

throw new Exception("Error when reading file.\n" + ex.Message);

}

try

{

int errNum;

if (keyFilepath == "")

{

errNum = HashFunction.readKeyString(keyText, isHexString);

}

else

{

errNum = HashFunction.readKeyFile(keyFilepath);

}

if (errNum == MD6.NO\_FILE)

{

throw new Exception("Key File does not exist");

}

if (errNum == MD6.WRONG\_FILE\_SIZE)

{

throw new Exception("Key file too big.\nExpected length <= 512 bytes.");

}

}

catch (Exception ex)

{

throw new Exception("Error when reading key.\n" + ex.Message);

}

try

{

e.Result = HashFunction.Hash();

}

catch (Exception ex)

{

throw new Exception("Error when calculating hash value.\n" + ex.Message);

}

}

catch(Exception ex)

{

throw ex;

}

}

private void bw\_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)

{

bw.RunWorkerCompleted -=

new RunWorkerCompletedEventHandler(bw\_RunWorkerCompleted);

if ((e.Cancelled == true))

{

MD6ProgressBarLabel.Content = "Cancelled";

MD6ProgressBar.IsIndeterminate = false;

System.Windows.MessageBox.Show("Process Canceled", "Cancelled", MessageBoxButton.OK, MessageBoxImage.Warning);

}

else if (!(e.Error == null))

{

MD6ProgressBarLabel.Content = "Error";

MD6ProgressBar.IsIndeterminate = false;

System.Windows.MessageBox.Show(e.Error.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

else

{

MD6ProgressBarLabel.Content = "Done";

MD6HashResultLabel.Content = "Calculated Hash";

MD6Hash.Text = e.Result.ToString();

MD6Hash.IsEnabled = false;

MD6ProgressBar.IsIndeterminate = false;

hashFile = "";

menuFileSaveHashButton.IsEnabled = false;

menuFileSaveHashAsButton.IsEnabled = true;

CompareHashButton.IsEnabled = true;

}

}

public void bw\_RunWorkerCompletedPassword(object sender, RunWorkerCompletedEventArgs e)

{

bw.RunWorkerCompleted -=

new RunWorkerCompletedEventHandler(bw\_RunWorkerCompletedPassword);

if ((e.Cancelled == true))

{

MD6ProgressBarLabel.Content = "Cancelled";

MD6ProgressBar.IsIndeterminate = false;

System.Windows.MessageBox.Show("Process Canceled", "Cancelled", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

else if (!(e.Error == null))

{

MD6ProgressBarLabel.Content = "Error";

MD6ProgressBar.IsIndeterminate = false;

System.Windows.MessageBox.Show(e.Error.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

else

{

MD6ProgressBarLabel.Content = "Done";

PasswordString = e.Result.ToString();

MD6ProgressBar.IsIndeterminate = false;

hashFile = "";

textKey.IsEnabled = false;

menuFileSaveKeyAsButton.IsEnabled = false;

labelKey.Content = "User key from password selected";

keyFile = "";

}

}

public void bw\_RunWorkerCompletedComparePasswordInput(object sender, RunWorkerCompletedEventArgs e)

{

bw.RunWorkerCompleted -=

new RunWorkerCompletedEventHandler(bw\_RunWorkerCompletedComparePasswordInput);

if ((e.Cancelled == true))

{

MD6ProgressBarLabel.Content = "Cancelled";

MD6ProgressBar.IsIndeterminate = false;

System.Windows.MessageBox.Show("Process Canceled", "Cancelled", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

else if (!(e.Error == null))

{

MD6ProgressBarLabel.Content = "Error";

MD6ProgressBar.IsIndeterminate = false;

System.Windows.MessageBox.Show(e.Error.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

else

{

System.GC.Collect();

bw.RunWorkerCompleted +=

new RunWorkerCompletedEventHandler(bw\_RunWorkerCompletedCompare);

List<object> arguments = new List<object>();

arguments.Add(Options.Instance.dVal);

arguments.Add(Options.Instance.Lval);

arguments.Add(Options.Instance.rVal);

arguments.Add(messageFile);

arguments.Add(textFileToHash.Text);

arguments.Add("");//No keyfile

arguments.Add(e.Result.ToString());//Key as result of hash function

arguments.Add(true);//Flag indicates that string is hex value

bw.RunWorkerAsync(arguments);

}

}

public void bw\_RunWorkerCompletedCompare(object sender, RunWorkerCompletedEventArgs e)

{

bw.RunWorkerCompleted -=

new RunWorkerCompletedEventHandler(bw\_RunWorkerCompletedCompare);

if ((e.Cancelled == true))

{

MD6ProgressBarLabel.Content = "Cancelled";

MD6ProgressBar.IsIndeterminate = false;

System.Windows.MessageBox.Show("Process Canceled", "Cancelled", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

else if (!(e.Error == null))

{

MD6ProgressBarLabel.Content = "Error";

MD6ProgressBar.IsIndeterminate = false;

System.Windows.MessageBox.Show(e.Error.Message, "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

else

{

MD6ProgressBarLabel.Content = "Done";

MD6ProgressBar.IsIndeterminate = false;

if(MD6Hash.Text == e.Result.ToString())

{

System.Windows.MessageBox.Show("Hash values match", "Comparison", MessageBoxButton.OK, MessageBoxImage.None);

}

else

{

System.Windows.MessageBox.Show("Hash values dont match", "Comparison", MessageBoxButton.OK, MessageBoxImage.Warning);

}

}

}

private void MenuEditOptionsButton\_Click(object sender, RoutedEventArgs e)

{

opt.Show();

opt.Activate();

}

public string messageFile;

public string keyFile;

public string hashFile;

public string PasswordString;

private void menuFileSaveHashAsButton\_Click(object sender, RoutedEventArgs e)

{

try

{

SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.Title = "Save Hash File As";

saveFileDialog.ShowDialog();

if (saveFileDialog.FileName != "")

{

System.IO.FileStream fs =

(System.IO.FileStream)saveFileDialog.OpenFile();

byte[] stringArray = Encoding.UTF8.GetBytes(MD6Hash.Text);

fs.Write(stringArray, 0, stringArray.Length);

fs.Close();

menuFileSaveHashButton.IsEnabled = true;

hashFile = saveFileDialog.FileName;

}

}

catch(Exception ex)

{

System.Windows.MessageBox.Show(ex.ToString(), "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

}

private void menuFileSaveHashButton\_Click(object sender, RoutedEventArgs e)

{

try

{

System.IO.FileStream fs = new System.IO.FileStream(hashFile, System.IO.FileMode.Create);

byte[] stringArray = Encoding.UTF8.GetBytes(MD6Hash.Text);

fs.Write(stringArray, 0, stringArray.Length);

fs.Close();

}

catch (Exception ex)

{

System.Windows.MessageBox.Show(ex.ToString(), "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

}

private void menuFileSaveKeyAsButton\_Click(object sender, RoutedEventArgs e)

{

try

{

SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.Title = "Save Key File As";

saveFileDialog.ShowDialog();

if (saveFileDialog.FileName != "")

{

System.IO.FileStream fs =

(System.IO.FileStream)saveFileDialog.OpenFile();

byte[] stringArray = Encoding.UTF8.GetBytes(textKey.Text);

fs.Write(stringArray, 0, stringArray.Length);

fs.Close();

menuFileSaveHashButton.IsEnabled = true;

}

}

catch (Exception ex)

{

System.Windows.MessageBox.Show(ex.ToString(), "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

}

private void menuFileSaveTextAsButton\_Click(object sender, RoutedEventArgs e)

{

try

{

SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.Title = "Save Message File As";

saveFileDialog.ShowDialog();

if (saveFileDialog.FileName != "")

{

System.IO.FileStream fs =

(System.IO.FileStream)saveFileDialog.OpenFile();

byte[] stringArray = Encoding.UTF8.GetBytes(textFileToHash.Text);

fs.Write(stringArray, 0, stringArray.Length);

fs.Close();

menuFileSaveHashButton.IsEnabled = true;

}

}

catch (Exception ex)

{

System.Windows.MessageBox.Show(ex.ToString(), "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

}

private void menuFileExitButton\_Click(object sender, RoutedEventArgs e)

{

System.Windows.Application.Current.Shutdown();

}

private void Window\_Closed(object sender, EventArgs e)

{

System.Windows.Application.Current.Shutdown();

}

private void MenuEditPasswordoptions\_Click(object sender, RoutedEventArgs e)

{

PassOpt.Show();

PassOpt.Activate();

}

private void PasswordKeyButton\_Click(object sender, RoutedEventArgs e)

{

if (bw.IsBusy != true)

{

PasswordInput passInt = new PasswordInput();

passInt.ShowDialog();

passInt = null;

return;

}

}

private void CompareHashButton\_Click(object sender, RoutedEventArgs e)

{

if (bw.IsBusy != true)

{

ComparePasswordinput passInt = new ComparePasswordinput();

passInt.ShowDialog();

passInt = null;

return;

}

}

}

}

### Options.xaml

<Window x:Class="MD6Project.Options"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:MD6Project"

mc:Ignorable="d"

Title="Options" Height="312" Width="274" Closing="Window\_Closing" Icon="IconMain.ico">

<Grid>

<Label x:Name="label" Content="Message digest length in bits (d):" HorizontalAlignment="Left" Margin="30,10,0,0" VerticalAlignment="Top"/>

<TextBox x:Name="MessageDigestLength" HorizontalAlignment="Left" Height="23" Margin="30,40.96,0,0" TextWrapping="Wrap" Text="256" VerticalAlignment="Top" Width="120"/>

<Label x:Name="label1" Content="Mode control (L):" HorizontalAlignment="Left" Margin="30,68.96,0,0" VerticalAlignment="Top"/>

<TextBox x:Name="ModeControlText" HorizontalAlignment="Left" Height="23" Margin="30,99.92,0,0" TextWrapping="Wrap" Text="64" VerticalAlignment="Top" Width="120"/>

<Label x:Name="label2" Content="Number of rounds (r):" HorizontalAlignment="Left" Margin="30,127.92,0,0" VerticalAlignment="Top"/>

<TextBox x:Name="NumberOfRoundsText" HorizontalAlignment="Left" Height="23" Margin="30,158.88,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Width="120" IsEnabled="False"/>

<CheckBox x:Name="CustomRoundsCheck" Content="Custom value" HorizontalAlignment="Left" Margin="155,160.88,0,0" VerticalAlignment="Top" Click="CustomRoundsCheck\_Click"/>

<Label x:Name="label3" Content="0 &lt; d &lt;= 512" HorizontalAlignment="Left" Margin="155,38,0,0" VerticalAlignment="Top"/>

<Label x:Name="label4" Content="0 &lt;= L &lt;= 64" HorizontalAlignment="Left" Margin="155,96.96,0,0" VerticalAlignment="Top"/>

<Label x:Name="label5" Content="Default value of r:&#xD;&#xA;-without key: 40 + d/4&#xD;&#xA;-with key: max(80, 40 + d/4)" HorizontalAlignment="Left" Margin="30,180.978,0,0" VerticalAlignment="Top" Width="165" Height="54.022" IsEnabled="False" FontSize="11"/>

<Button x:Name="OptionsApplyButton" Content="Apply" HorizontalAlignment="Left" Margin="10,240,0,0" VerticalAlignment="Top" Width="50" Height="31" Click="OptionsApplyButton\_Click"/>

<Button x:Name="OptionsDiscardButton" Content="Discard" HorizontalAlignment="Left" Margin="105.21,240,0,0" VerticalAlignment="Top" Width="50" Height="31" Click="OptionsDiscardButton\_Click"/>

<Button x:Name="OptionsResetButton" Content="Reset" HorizontalAlignment="Left" Margin="206,240,0,0" VerticalAlignment="Top" Width="50" Height="31" Click="OptionsResetButton\_Click"/>

</Grid>

</Window>

#### Options.xaml.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace MD6Project

{

/// <summary>

/// Interaction logic for Options.xaml

/// </summary>

public partial class Options : Window

{

public static Options Instance { get; private set; }

static Options()

{

Instance = new Options();

}

private Options()

{

InitializeComponent();

Lval = 64;

dVal = 256;

rVal = 0;

CustomRounds = false;

}

private void Window\_Closing(object sender, System.ComponentModel.CancelEventArgs e)

{

e.Cancel = true;

this.Visibility = Visibility.Hidden;

}

private void CustomRoundsCheck\_Click(object sender, RoutedEventArgs e)

{

NumberOfRoundsText.IsEnabled = (bool)CustomRoundsCheck.IsChecked;

}

public int Lval = 64;

public int dVal = 256;

public int rVal = 0;

bool CustomRounds = false;

private void OptionsApplyButton\_Click(object sender, RoutedEventArgs e)

{

int LvalTemp = 0;

int dValTemp = 0;

int rValTemp = 0;

bool CustomRoundsTemp = (bool)CustomRoundsCheck.IsChecked;

try

{

dValTemp = Convert.ToInt32(MessageDigestLength.Text, 10);

if ((dValTemp <= 0) || (dValTemp > 512))

{

System.Windows.MessageBox.Show("Wrong message digest length (d)", "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

}

catch(Exception ex)

{

System.Windows.MessageBox.Show("Wrong message digest length (d)", "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

try

{

LvalTemp = Convert.ToInt32(ModeControlText.Text, 10);

if ((LvalTemp < 0) || (LvalTemp > 64))

{

System.Windows.MessageBox.Show("Wrong mode control (L)", "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

}

catch (Exception ex)

{

System.Windows.MessageBox.Show("Wrong mode control (L)", "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

if ((bool)CustomRoundsCheck.IsChecked)

{

try

{

rValTemp = Convert.ToInt32(NumberOfRoundsText.Text, 10);

if ((rValTemp < 1))

{

System.Windows.MessageBox.Show("Wrong number of rounds (r)", "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

}

catch (Exception ex)

{

System.Windows.MessageBox.Show("Wrong number of rounds (r)", "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

CustomRoundsTemp = true;

}

else

{

CustomRoundsTemp = false;

}

Lval = LvalTemp;

dVal = dValTemp;

if (CustomRoundsTemp)

{

rVal = rValTemp;

CustomRounds = CustomRoundsTemp;

}

else

{

rVal = 0;

CustomRounds = CustomRoundsTemp;

}

}

private void OptionsDiscardButton\_Click(object sender, RoutedEventArgs e)

{

MessageDigestLength.Text = dVal.ToString();

ModeControlText.Text = Lval.ToString();

if (CustomRounds)

{

CustomRoundsCheck.IsChecked = true;

NumberOfRoundsText.IsEnabled = true;

NumberOfRoundsText.Text = rVal.ToString();

}

else

{

NumberOfRoundsText.IsEnabled = false;

CustomRoundsCheck.IsChecked = false;

NumberOfRoundsText.Text = "";

}

}

private void OptionsResetButton\_Click(object sender, RoutedEventArgs e)

{

Lval = 64;

dVal = 256;

rVal = 0;

CustomRounds = false;

NumberOfRoundsText.IsEnabled = false;

CustomRoundsCheck.IsChecked = false;

MessageDigestLength.Text = "256";

ModeControlText.Text = "64";

NumberOfRoundsText.Text = "";

}

}

}

### PasswordOptions.xaml

<Window x:Name="PasswordOpt" x:Class="MD6Project.PasswordOptions"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:MD6Project"

mc:Ignorable="d"

Title="Password Options" Height="252" Width="218" Icon="IconMain.ico" ResizeMode="NoResize" Closing="PasswordOpt\_Closing">

<Grid>

<TextBox x:Name="PasswordLengthTextBox" HorizontalAlignment="Left" Height="23" Margin="10,40.96,0,0" TextWrapping="Wrap" Text="0" VerticalAlignment="Top" Width="120"/>

<Label x:Name="PasswordLengthlabel" Content="Min Password Length" HorizontalAlignment="Left" Margin="10,10,0,0" VerticalAlignment="Top"/>

<CheckBox x:Name="UpperCaseCheckBox" Content="Upper Case required" HorizontalAlignment="Left" Margin="10,68.958,0,0" VerticalAlignment="Top" Click="UpperCaseCheckBox\_Click"/>

<CheckBox x:Name="LowerCaseCheckBox" Content="Lower Case required" HorizontalAlignment="Left" Margin="10,89.057,0,0" VerticalAlignment="Top" Click="LowerCaseCheckBox\_Click"/>

<CheckBox x:Name="NumbersCheckBox" Content="Numbers required" HorizontalAlignment="Left" Margin="10,109.155,0,0" VerticalAlignment="Top"/>

<Button x:Name="PasswordApplyButton" Content="Apply" HorizontalAlignment="Left" Margin="10,183,0,0" VerticalAlignment="Top" Width="75" Height="30" Click="PasswordApplyButton\_Click"/>

<Button x:Name="PasswordDiscardButton" Content="Discard" HorizontalAlignment="Left" Margin="115,183,0,0" VerticalAlignment="Top" Width="75" Height="30" Click="PasswordDiscardButton\_Click"/>

<CheckBox x:Name="LettersCheckBox" Content="Letters required" HorizontalAlignment="Left" Margin="10,129.253,0,0" VerticalAlignment="Top" Click="LettersCheckBox\_Click"/>

</Grid>

</Window>

#### PasswordOptions.xaml.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace MD6Project

{

/// <summary>

/// Interaction logic for PasswordOptions.xaml

/// </summary>

public partial class PasswordOptions : Window

{

public static PasswordOptions Instance { get; private set; }

static PasswordOptions()

{

Instance = new PasswordOptions();

}

private PasswordOptions()

{

InitializeComponent();

}

public bool Upper = false;

public bool Lower = false;

public bool Number = false;

public bool Letter = false;

public int minLength = 0;

private void PasswordApplyButton\_Click(object sender, RoutedEventArgs e)

{

bool TempUpper = (bool)UpperCaseCheckBox.IsChecked;

bool TempLower = (bool)LowerCaseCheckBox.IsChecked;

bool TempNumber = (bool)NumbersCheckBox.IsChecked;

bool TempLetter = (bool)LettersCheckBox.IsChecked;

int TempminLength;

try

{

TempminLength = Convert.ToInt32(PasswordLengthTextBox.Text, 10);

if (TempminLength < 0)

{

System.Windows.MessageBox.Show("Wrong minimum password length", "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

}

catch (Exception ex)

{

System.Windows.MessageBox.Show("Wrong minimum password length", "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

Upper = TempUpper;

Lower = TempLower;

Number = TempNumber;

Letter = TempLetter;

minLength = TempminLength;

}

private void PasswordDiscardButton\_Click(object sender, RoutedEventArgs e)

{

PasswordLengthTextBox.Text = minLength.ToString();

UpperCaseCheckBox.IsChecked = Upper;

LowerCaseCheckBox.IsChecked = Lower;

NumbersCheckBox.IsChecked = Number;

LettersCheckBox.IsChecked = Letter;

}

private void UpperCaseCheckBox\_Click(object sender, RoutedEventArgs e)

{

if ((bool)UpperCaseCheckBox.IsChecked)

{

LettersCheckBox.IsChecked = true;

}

}

private void LowerCaseCheckBox\_Click(object sender, RoutedEventArgs e)

{

if ((bool)LowerCaseCheckBox.IsChecked)

{

LettersCheckBox.IsChecked = true;

}

}

private void LettersCheckBox\_Click(object sender, RoutedEventArgs e)

{

if (!(bool)LettersCheckBox.IsChecked)

{

UpperCaseCheckBox.IsChecked = false;

LowerCaseCheckBox.IsChecked = false;

}

}

private void PasswordOpt\_Closing(object sender, System.ComponentModel.CancelEventArgs e)

{

e.Cancel = true;

this.Visibility = Visibility.Hidden;

}

}

}

### PasswordInput.xaml

<Window x:Name="PasswordInputWindow" x:Class="MD6Project.PasswordInput"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:MD6Project"

mc:Ignorable="d"

Title="Input Password" Height="252" Width="300">

<Grid>

<PasswordBox x:Name="FirstpasswordBox" HorizontalAlignment="Left" Margin="60,50,0,0" VerticalAlignment="Top" Width="165"/>

<PasswordBox x:Name="SecondpasswordBox" HorizontalAlignment="Left" Margin="60,103.92,0,0" VerticalAlignment="Top" Width="165"/>

<Button x:Name="PasswordinputButton" Content="Ok" HorizontalAlignment="Left" Margin="105,145,0,0" VerticalAlignment="Top" Width="75" RenderTransformOrigin="0.8,0.501" Click="PasswordinputButton\_Click"/>

<Label x:Name="label" Content="Input password" HorizontalAlignment="Left" Margin="60,19.04,0,0" VerticalAlignment="Top"/>

<Label x:Name="label1" Content="Repeat password" HorizontalAlignment="Left" Margin="60,72.96,0,0" VerticalAlignment="Top"/>

</Grid>

</Window>

#### PasswordInput.xaml.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using System.ComponentModel;

namespace MD6Project

{

/// <summary>

/// Interaction logic for PasswordInput.xaml

/// </summary>

public partial class PasswordInput : Window

{

public PasswordInput()

{

InitializeComponent();

}

private void PasswordinputButton\_Click(object sender, RoutedEventArgs e)

{

bool Upper = PasswordOptions.Instance.Upper;

bool Lower = PasswordOptions.Instance.Lower;

bool Number = PasswordOptions.Instance.Number;

bool Letter = PasswordOptions.Instance.Letter;

int minLength = PasswordOptions.Instance.minLength;

bool CurUpper = false;

bool CurLower = false;

bool CurNumber = false;

bool CurLetter = false;

if (FirstpasswordBox.Password.Length == 0)

{

System.Windows.MessageBox.Show("Password can't be empty", "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

if (FirstpasswordBox.Password.Length < minLength)

{

System.Windows.MessageBox.Show("Password too short.\nMust be at least " + minLength +" symbols.", "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

int i = 0;

while ((i < FirstpasswordBox.Password.Length))

{

CurUpper |= Char.IsUpper(FirstpasswordBox.Password[i]);

CurLower |= Char.IsLower(FirstpasswordBox.Password[i]);

CurLetter |= Char.IsLetter(FirstpasswordBox.Password[i]);

CurNumber |= Char.IsNumber(FirstpasswordBox.Password[i]);

++i;

}

if (Upper && !CurUpper)

{

System.Windows.MessageBox.Show("Password must contain Upper Case letters", "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

if (Lower && !CurLower)

{

System.Windows.MessageBox.Show("Password must contain Lower Case letters", "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

if (Letter && !CurLetter)

{

System.Windows.MessageBox.Show("Password must contain letters", "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

if (Number && !CurNumber)

{

System.Windows.MessageBox.Show("Password must contain numbers", "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

if (FirstpasswordBox.Password != SecondpasswordBox.Password)

{

System.Windows.MessageBox.Show("Passwords dont match", "Error", MessageBoxButton.OK, MessageBoxImage.Warning);

return;

}

MainWindow.Instance.bw.RunWorkerCompleted +=

new RunWorkerCompletedEventHandler(MainWindow.Instance.bw\_RunWorkerCompletedPassword);

MainWindow.Instance.MD6ProgressBarLabel.Content = "Calculating";

MainWindow.Instance.MD6ProgressBar.IsIndeterminate = true;

List<object> arguments = new List<object>();

arguments.Add(Options.Instance.dVal);

arguments.Add(Options.Instance.Lval);

arguments.Add(Options.Instance.rVal);

arguments.Add("");

arguments.Add(FirstpasswordBox.Password);

arguments.Add("");

arguments.Add("");

arguments.Add(false);//Flag indicates that string is not hex value

MainWindow.Instance.bw.RunWorkerAsync(arguments);

this.Close();

}

}

}

### ComparePasswordInput.xaml

<Window x:Name="CompPassinputWindow" x:Class="MD6Project.ComparePasswordinput"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:MD6Project"

mc:Ignorable="d"

Title="Input Password" Height="127" Width="300" Icon="IconMain.ico">

<Grid>

<PasswordBox x:Name="passwordBox" Margin="0,20,0,0" VerticalAlignment="Top" Width="160" HorizontalAlignment="Center"/>

<Button x:Name="button" Content="Ok" HorizontalAlignment="Center" Margin="0,30,0,0" VerticalAlignment="Center" Width="75" Click="button\_Click"/>

</Grid>

</Window>

#### ComparePasswordInput.xaml.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using System.ComponentModel;

namespace MD6Project

{

/// <summary>

/// Interaction logic for ComparePasswordinput.xaml

/// </summary>

public partial class ComparePasswordinput : Window

{

public ComparePasswordinput()

{

InitializeComponent();

}

private void button\_Click(object sender, RoutedEventArgs e)

{

MainWindow.Instance.bw.RunWorkerCompleted +=

new RunWorkerCompletedEventHandler(MainWindow.Instance.bw\_RunWorkerCompletedComparePasswordInput);

MainWindow.Instance.MD6ProgressBarLabel.Content = "Calculating";

MainWindow.Instance.MD6ProgressBar.IsIndeterminate = true;

if (passwordBox.Password.Length == 0)

{

RunWorkerCompletedEventArgs fakeComplition = new RunWorkerCompletedEventArgs("",null,false);

MainWindow.Instance.bw\_RunWorkerCompletedComparePasswordInput(this, fakeComplition);

}

else

{

List<object> arguments = new List<object>();

arguments.Add(Options.Instance.dVal);

arguments.Add(Options.Instance.Lval);

arguments.Add(Options.Instance.rVal);

arguments.Add("");

arguments.Add(passwordBox.Password);

arguments.Add("");

arguments.Add("");

arguments.Add(false);//Flag indicates that string is not hex value

MainWindow.Instance.bw.RunWorkerAsync(arguments);

}

this.Close();

}

}

}

### MD6.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

namespace MD6Project

{

class MD6

{

public const int OK = 0;

public const int NO\_FILE = 1;

public const int WRONG\_FILE\_SIZE = 2;

public UInt64[] Message;

private UInt64 Messagelength;

private UInt64[] Key;

uint keylen;

uint d;

public uint L;

uint r;

public const int bw = 64;//words

public const int cw = 16;//words

private UInt64[] Q = { 0x7311c2812425cfa0

,0x6432286434aac8e7

,0xb60450e9ef68b7c1

,0xe8fb23908d9f06f1

,0xdd2e76cba691e5bf

,0x0cd0d63b2c30bc41

,0x1f8ccf6823058f8a

,0x54e5ed5b88e3775d

,0x4ad12aae0a6d6031

,0x3e7f16bb88222e0d

,0x8af8671d3fb50c2c

,0x995ad1178bd25c31

,0xc878c1dd04c4b633

,0x3b72066c7a1552ac

,0x0d6f3522631effcb }; // 960 bits of √6 as a sequence of 15 64-bit words

public MD6(uint dVal = 256, uint LVal = 64, uint rVal = 0)

{

d = dVal;

L = LVal;

r = rVal;

}

public int readMessageFile(string filePath)

{

FileInfo fileInf = new FileInfo(filePath);

if (fileInf.Exists)

{

Messagelength = (UInt64)fileInf.Length;

Message = new UInt64[Messagelength / 8 + (UInt64)(Messagelength % 8 > 0 ? 1 : 0)];

Messagelength \*= 8; //from bytes to bits

using (FileStream fs = fileInf.OpenRead())

{

UInt64 readCount = (UInt64)Message.Length;

byte[] buf = new byte[8];

for (UInt64 k = 0; k < readCount; ++k)

{

fs.Read(buf, 0, buf.Length);

if (BitConverter.IsLittleEndian)

{

Message[k] |= (UInt64)buf[0] << 56;

Message[k] |= (UInt64)buf[1] << 48;

Message[k] |= (UInt64)buf[2] << 40;

Message[k] |= (UInt64)buf[3] << 32;

Message[k] |= (UInt64)buf[4] << 24;

Message[k] |= (UInt64)buf[5] << 16;

Message[k] |= (UInt64)buf[6] << 8;

Message[k] |= (UInt64)buf[7];

}

else

{

Message[k] = BitConverter.ToUInt64(buf, 0);

}

Array.Clear(buf, 0, buf.Length);

}

}

}

else

return (NO\_FILE);

return (OK);

}

public void readMessageString(string str)

{

Messagelength = (UInt64)str.Length;

Message = new UInt64[Messagelength / 8 + (UInt64)(Messagelength % 8 > 0 ? 1 : 0)];

UInt64 readCount = (UInt64)Message.Length;

byte[] buf = new byte[8];

byte[] stringArray = Encoding.UTF8.GetBytes(str);

for (UInt64 k = 0; k < readCount; ++k)

{

int copySize = 0;

if (k < (readCount - 1))

{

copySize = 8;

}

else

{

copySize = (int)((Messagelength % 8 > 0) ? (Messagelength % 8) : 8);

}

Array.Copy(stringArray,(int)k\*8,buf,0, copySize);

if (BitConverter.IsLittleEndian)

{

Message[k] |= (UInt64)buf[0] << 56;

Message[k] |= (UInt64)buf[1] << 48;

Message[k] |= (UInt64)buf[2] << 40;

Message[k] |= (UInt64)buf[3] << 32;

Message[k] |= (UInt64)buf[4] << 24;

Message[k] |= (UInt64)buf[5] << 16;

Message[k] |= (UInt64)buf[6] << 8;

Message[k] |= (UInt64)buf[7];

}

else

{

Message[k] = BitConverter.ToUInt64(buf, 0);

}

Array.Clear(buf, 0, buf.Length);

}

Messagelength \*= 8;//from bytes to bits

}

public int readKeyFile(string filePath)

{

FileInfo fileInf = new FileInfo(filePath);

if (fileInf.Exists)

{

UInt64 Testlength = (UInt64)fileInf.Length;

if(Testlength > 512)

{

return(WRONG\_FILE\_SIZE);

}

keylen = (uint)Testlength;

Key = new UInt64[8];

using (FileStream fs = fileInf.OpenRead())

{

UInt64 readCount = (UInt64)(keylen / 8 + ((keylen % 8 > 0) ? 1 : 0));

byte[] buf = new byte[8];

for (UInt64 k = 0; k < readCount; ++k)

{

fs.Read(buf, 0, buf.Length);

if (BitConverter.IsLittleEndian)

{

Key[k] |= (UInt64)buf[0] << 56;

Key[k] |= (UInt64)buf[1] << 48;

Key[k] |= (UInt64)buf[2] << 40;

Key[k] |= (UInt64)buf[3] << 32;

Key[k] |= (UInt64)buf[4] << 24;

Key[k] |= (UInt64)buf[5] << 16;

Key[k] |= (UInt64)buf[6] << 8;

Key[k] |= (UInt64)buf[7];

}

else

{

Message[k] = BitConverter.ToUInt64(buf, 0);

}

Array.Clear(buf, 0, buf.Length);

}

}

if (r == 0)

{

r = 40 + d / 4;

if ((keylen != 0) && (r < 80))

{

r = 80;

}

}

}

else

return (NO\_FILE);

return (OK);

}

public static byte[] StringToByteArray(String hex)

{

int NumberChars = hex.Length;

byte[] bytes = new byte[NumberChars / 2];

for (int i = 0; i < NumberChars; i += 2)

bytes[i / 2] = Convert.ToByte(hex.Substring(i, 2), 16);

return bytes;

}

public int readKeyString(string str, bool isHexString)

{

keylen = (uint)str.Length;

if (keylen > 512)

{

return (WRONG\_FILE\_SIZE);

}

Key = new UInt64[8];

byte[] buf = new byte[8];

byte[] stringArray;

if (!isHexString) {

stringArray = Encoding.UTF8.GetBytes(str);

}

else

{

stringArray = StringToByteArray(str);

keylen /= 2;

}

UInt64 readCount = (UInt64)(keylen / 8 + ((keylen % 8 > 0) ? 1 : 0));

for (UInt64 k = 0; k < readCount; ++k)

{

int copySize = 0;

if (k < (readCount - 1))

{

copySize = 8;

}

else

{

copySize = (int)((keylen % 8 > 0) ? (keylen % 8) : 8);

}

Array.Copy(stringArray, (int)k \* 8, buf, 0, copySize);

if (BitConverter.IsLittleEndian)

{

Key[k] |= (UInt64)buf[0] << 56;

Key[k] |= (UInt64)buf[1] << 48;

Key[k] |= (UInt64)buf[2] << 40;

Key[k] |= (UInt64)buf[3] << 32;

Key[k] |= (UInt64)buf[4] << 24;

Key[k] |= (UInt64)buf[5] << 16;

Key[k] |= (UInt64)buf[6] << 8;

Key[k] |= (UInt64)buf[7];

}

else

{

Message[k] = BitConverter.ToUInt64(buf, 0);

}

Array.Clear(buf, 0, buf.Length);

}

if (r == 0)

{

r = 40 + d / 4;

if ((keylen != 0) && (r < 80))

{

r = 80;

}

}

return (OK);

}

public string Hash()

{

UInt64[] res = ModeOfOperation();

byte[] retVal = new byte[(d + 7) / 8];

int i = 0;

while (i < retVal.Length)

{

UInt64 val = res[res.Length - 1 - i / 8];

int k = 0;

while ((i < retVal.Length) && (k < 8))

{

byte mask = 0xFF;

retVal[retVal.Length - 1 - i++] = (byte)(val & mask);

val = val >> 8;

++k;

}

}

byte bitoffset = (byte)(d % 8);

if (bitoffset > 0)

{

for (int k = 0; k < retVal.Length - 1; ++k)

{

retVal[k] = (byte)((retVal[k] << (8 - bitoffset)) | (retVal[k + 1] >> bitoffset));

}

retVal[retVal.Length - 1] = (byte)(retVal[retVal.Length - 1] << (8 - bitoffset));

}

StringBuilder sb = new StringBuilder(retVal.Length \* 2);

foreach (byte b in retVal)

{

sb.AppendFormat("{0:x2}", b);

}

string hashval = sb.ToString();

if ((d % 8 <5) && (d%8 > 0)) {

hashval = hashval.Remove(hashval.Length - 1);

}

return (hashval);

}

private UInt64[] ModeOfOperation()

{

uint l = 0;

while (true)

{

++l;

if (l == L + 1)

{

return SEQ();

}

else

{

Message = PAR(l);

if (Message.Length == cw)

{

return Message;

}

Messagelength = (UInt64)Message.LongLength \* 64;

System.GC.Collect();

}

}

}

private UInt64[] PAR(uint l)

{

UInt64 p = (UInt64)Message.LongLength \* 64 - Messagelength + ((Message.LongLength % bw) > 0 ? (((UInt64)bw - (UInt64)Message.LongLength % bw) \* 64) : 0);

UInt64 j = (UInt64)(Message.Length / bw) + (UInt64)(Message.LongLength % bw > 0 ? 1 : 0);

uint z = (uint)(j == 1 ? 1 : 0);

UInt64 V = 0;

V |= r;

V = V << 8;

V |= L;

V = V << 4;

V |= z;

V = V << 16;

V |= p;

V = V << 8;

V |= keylen;

V = V << 12;

V |= d;

UInt64 noPadding = 0xFFFFFFF0000FFFFF;

UInt64[] Ci = new UInt64[cw];

UInt64[] Res = new UInt64[cw \* j];

UInt64[] fVal = new UInt64[n];

Array.Copy(Q, 0, fVal, 0, Q.Length);

Array.Copy(Key, 0, fVal, Q.Length, Key.Length);

for (UInt64 i = 0; i < j; ++i)

{

UInt64 localV = V;

if (i < j - 1)

localV &= noPadding;

UInt64 U = l;

U = U << 56;

U |= i;

fVal[Q.Length + Key.Length] = U;

fVal[Q.Length + Key.Length + 1] = localV;

if (i < j - 1)

{

Array.Copy(Message, bw \* (Int64)i, fVal, Q.Length + Key.Length + 2, bw);

}

else

{

if (Message.LongLength % bw > 0)

{

Array.Copy(Message, bw \* (Int64)i, fVal, Q.Length + Key.Length + 2, Message.LongLength % bw);

for (Int64 k = Message.LongLength % bw; k < bw; ++k)

{

fVal[k + Q.Length + Key.Length + 2] = 0;

}

}

else

{

Array.Copy(Message, bw \* (Int64)i, fVal, Q.Length + Key.Length + 2, bw);

}

}

//call to compress

Ci = Compress(ref fVal, r);

Array.Copy(Ci, 0, Res, cw \* (Int64)i, cw);

}

return Res;

}

private UInt64[] SEQ()

{

UInt64 p = (UInt64)Message.LongLength \* 64 - Messagelength + ((Message.LongLength % (bw - cw)) > 0 ? (((UInt64)(bw - cw) - (UInt64)Message.LongLength % (bw - cw)) \* 64) : 0);

UInt64 j = (UInt64)(Message.Length / (bw - cw)) + (UInt64)(Message.LongLength % (bw - cw) > 0 ? 1 : 0);

UInt64 V = 0;

V |= r;

V = V << 8;

V |= L;

V = V << 4;

//V |= z;

V = V << 16;

V |= p;

V = V << 8;

V |= keylen;

V = V << 12;

V |= d;

UInt64 noPadding = 0xFFFFFFF0000FFFFF;

UInt64 z = 0x0000001000000000;

UInt64[] C = new UInt64[cw];

UInt64[] fVal = new UInt64[n];

Array.Copy(Q, 0, fVal, 0, Q.Length);

Array.Copy(Key, 0, fVal, Q.Length, Key.Length);

for (UInt64 i = 0; i < j; ++i)

{

UInt64 localV = V;

if (i < j - 1)

{

localV &= noPadding;

}

else

{

localV |= z;

}

UInt64 U = (L + 1);

U = U << 56;

U |= i;

fVal[Q.Length + Key.Length] = U;

fVal[Q.Length + Key.Length + 1] = localV;

Array.Copy(C, 0, fVal, Q.Length + Key.Length + 2, C.Length);

if (i < j - 1)

{

Array.Copy(Message, (bw - cw) \* (Int64)i, fVal, Q.Length + Key.Length + 2 + C.Length, (bw - cw));

}

else

{

if (Message.LongLength % (bw - cw) > 0)

{

Array.Copy(Message, (bw - cw) \* (Int64)i, fVal, Q.Length + Key.Length + 2 + C.Length, Message.LongLength % (bw - cw));

for (Int64 k = Message.LongLength % (bw - cw); k < (bw - cw); ++k)

{

fVal[k + Q.Length + Key.Length + 2 + C.Length] = 0;

}

}

else

{

Array.Copy(Message, (bw - cw) \* (Int64)i, fVal, Q.Length + Key.Length + 2 + C.Length, (bw - cw));

}

}

//call to compress

C = Compress(ref fVal, r);

}

return C;

}

private const uint n = 89;//words

private const uint t0 = 17;

private const uint t1 = 18;

private const uint t2 = 21;

private const uint t3 = 31;

private const uint t4 = 67;

private int[] ri = { 10, 5, 13, 10, 11, 12, 2, 7, 14, 15, 7, 13, 11, 7, 6, 12 };

private int[] li = { 11, 24, 9, 16, 15, 9, 27, 15, 6, 2, 29, 8, 15, 5, 31, 9 };

private UInt64 S0 = 0x0123456789abcdef;

private UInt64 Sdot = 0x7311c2812425cfa0;

private UInt64[] Compress(ref UInt64[] N, uint r)

{

UInt64[] C = new UInt64[cw];

uint t = r \* cw;

UInt64[] A = new UInt64[t + n];

Array.Copy(N, 0, A, 0, n);

UInt64 Si = S0;

for (uint i = n, j = 0; j < r; ++j)

{

for (uint k = 0; k < 16; ++k, ++i)

{

UInt64 x = Si ^ A[i - n] ^ A[i - t0];

x ^= (A[i - t1] & A[i - t2]) ^ (A[i - t3] & A[i - t4]);

x ^= (x >> ri[(i - n) % 16]);

A[i] = x ^ (x << li[(i - n) % 16]);

}

Si = (Si << 1 | Si >> 63) ^ (Si & Sdot);

}

Array.Copy(A, t + n - cw, C, 0, cw);

return C;

}

}

}

### About.xaml

<Window x:Class="MD6Project.About"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:MD6Project"

mc:Ignorable="d"

Title="About" Height="311" Width="438" ResizeMode="NoResize" Icon="IconMain.ico">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="410\*"/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="0\*"/>

<RowDefinition/>

<RowDefinition Height="0\*"/>

</Grid.RowDefinitions>

<Image x:Name="image" Margin="10,10,0,10" Source="About.png" Grid.Row="1" HorizontalAlignment="Left" Width="145"/>

<Label x:Name="label" Content="Курсовой проект по курсу &quot;Защита данных&quot;" HorizontalAlignment="Left" Margin="160,10,0,0" Grid.RowSpan="2" VerticalAlignment="Top"/>

<Label x:Name="label1" Content="Преподаватель Хорев П.Б." HorizontalAlignment="Left" Margin="160,40.96,0,0" Grid.RowSpan="2" VerticalAlignment="Top" RenderTransformOrigin="1.395,0.541"/>

<Label x:Name="label2" Content="Выполнил Черток В.Д." HorizontalAlignment="Left" Margin="160,71.92,0,0" Grid.RowSpan="2" VerticalAlignment="Top" RenderTransformOrigin="-0.342,0.193"/>

<Label x:Name="label3" Content="Группа А-05-14" HorizontalAlignment="Left" Margin="160,102.88,0,0" Grid.RowSpan="2" VerticalAlignment="Top" RenderTransformOrigin="0,0"/>

<TextBox x:Name="textBox" HorizontalAlignment="Left" Height="131.16" Margin="160,133.84,0,0" Grid.RowSpan="2" TextWrapping="Wrap" VerticalAlignment="Top" Width="251.927" Text="Программна реализация функции &#xA;хеширования MD6.&#xA;Хеширование файлов и&#xA;вводимых пользователем сообщений.&#xA;Возможность настройки параметров &#xA;алгоритма хеширования и использования &#xA;пользовательского ключа.&#xD;&#xA;Возможность генерации пользовательского ключа на основе парольной фразы с регулируемой минимальной длиной и сложностью."/>

</Grid>

</Window>

#### About.xaml.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace MD6Project

{

/// <summary>

/// Interaction logic for About.xaml

/// </summary>

public partial class About : Window

{

public About()

{

InitializeComponent();

}

}

}

### App.xaml

<Application x:Class="MD6Project.App"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:local="clr-namespace:MD6Project">

<Application.Resources>

</Application.Resources>

</Application>

#### App.xaml.cs

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Data;

using System.Linq;

using System.Windows;

namespace MD6Project

{

/// <summary>

/// Interaction logic for App.xaml

/// </summary>

public partial class App : Application

{

public App()

{

Startup += App\_Startup;

}

void App\_Startup(object sender, StartupEventArgs e)

{

MD6Project.MainWindow.Instance.Show();

}

}

}