Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Учебная практика

Программная реализация алгоритма MD5

Выполнил студент гр. 150502: Селезнев А.В

Проверил: Луцик Ю.А

Минск 2022

**Содержание**

1 Постановка задачи...................................................................................3

2Хеширование…………...........................................................................4

2.1 Криптографические хеш-функции………………..……….....4

2.2 Подробнее о хешировании ……………………………..….....4

3 Описание алгоритма хеширования MD5...............................................6

3.1 Алгоритм MD5....……………………………………………....6

3.2Шаг 1. Выравнивание потока.………………………………...6

3.3 Шаг 2. Добавление длины сообщения......................................7

3.4 Шаг 3. Инициализация буфера ……………………………….8

3.5 Шаг 4. Вычисление в цикле…………………………………...8

3.6 Шаг 5. Результат вычислений………………………………..10

3.7 Вывод результата …………………………………………….10

4 Криптоанализ…………………………………………………………..11

4.1 Извлечение сообщения из хеша……………………………...11

4.2 Подбор сообщений …………………………………...............11

4.3 Защита дайджестов от перебора ………………….…..……..14

5 Область применения хеша……………………………………….……15

5.1. Проверка целостности файлов………………………………15

5.2. Хеширования паролей.……….................................................15

Список использованных источников…………………….......................16

ПРИЛОЖЕНИЕ А……………………………………..............................17

ПРИЛОЖЕНИЕ Б……………………………………...............................19

**1 Постановка задачи**

Реализовать алгоритм MD5 на основе имеющегося опыта в алгоритмизации и программирования, а также закрепить базовые знания языка “С”.

**2 Хеширование**

**2.1 Криптографические хеш-функции**

Криптографическая хеш-функция - это математический алгоритм, который отображает данные произвольного размера в битовый массив фиксированного размера. Результат, производимый хеш-функцией, называется «хеш-суммой», «дайджестом сообщения» или же просто «хешем», а входные данные часто называют «сообщением» (см. рисунок 2.1).

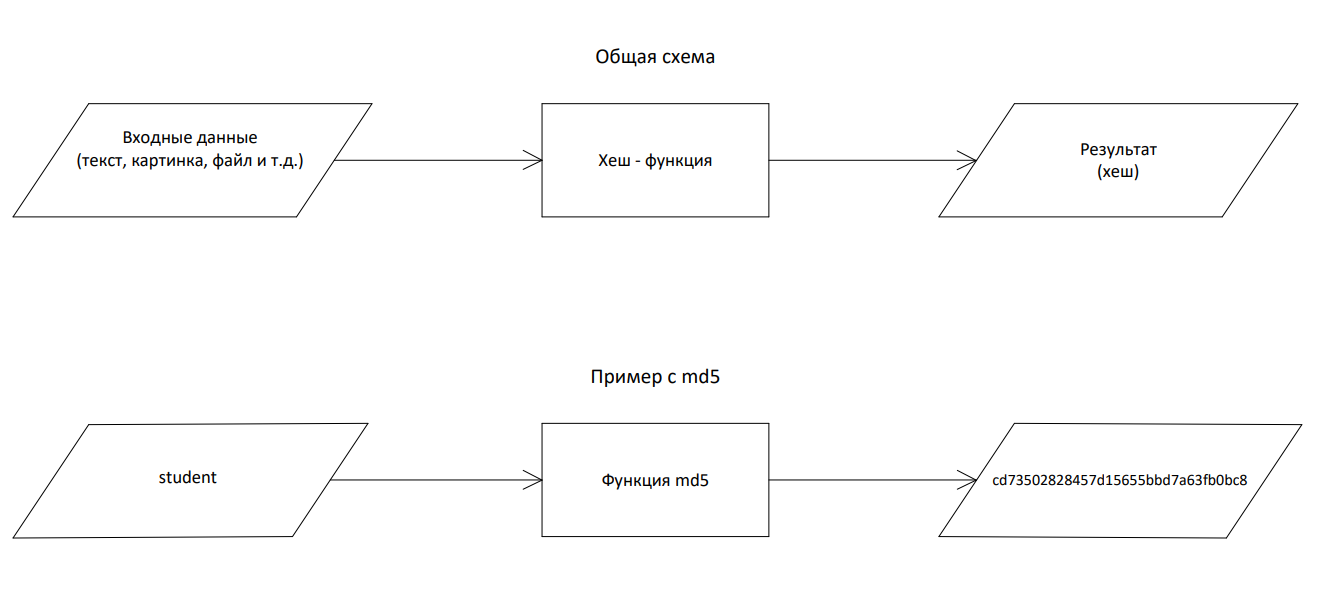
****

Рисунок 2.1 – схема хеширования

**2.2 Подробнее о хешировании**

* Хеширование – это не зашифрование.

Хеширование — это преобразование входных данных в уникальную последовательность символов, из которой почти невозможно получить исходное сообщение.

Например, при передаче сообщения через интернет:

хеширование используется, чтобы никто не смог незаметно изменить ваше сообщение;

шифрование используется, чтобы ваше сообщение не смог прочитать никто посторонний.

* Достать исходные данные из хеша почти невозможно.

Большая часть исходных данных фактически "потеряна" как часть преобразования. И хотя существуют радужные таблицы с предварительно вычисленными хешами, соль (случайные данные) делает ее неэффективной.

* При одинаковых входных данных результат идентичен.

(см. рисунки 2.2.1, 2.2.2).

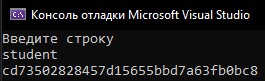


Рисунок 2.2.1 – хеширование в программе

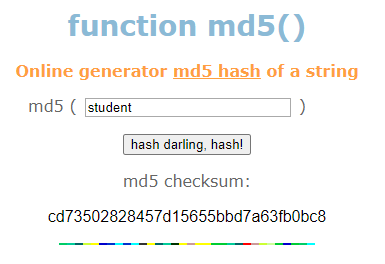


Рисунок 2.2.2 – хеширование онлайн

* Даже при различии в один бит хеш должен сильно отличаться.

Лавинный эффект — понятие в криптографии, при котором, изменение значения малого количества битов во входном сообщении ведет к «лавинному» изменению значений выходных битов хеша. Существует зависимость всех выходных битов от каждого входного бита.

(см. рисунки 2.2.3, 2.2.4).

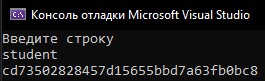


Рисунок 2.2.3 – Хеширование в программе

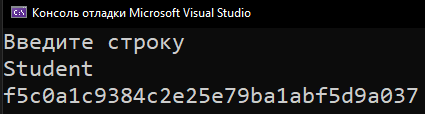


Рисунок 2.2.4 – Хеширование измененного сообщения

**3 Описание алгоритма хеширования MD5**

**3.1 Алгоритм MD5.**

На вход алгоритма поступает входной поток данных, хеш которого необходимо найти. Длина сообщения измеряется в битах и может быть любой (в том числе нулевой). Запишем длину сообщения в initial\_len.

Это число целое и неотрицательное. Кратность каким-либо числам необязательна. После поступления данных идёт процесс подготовки потока к вычислениям

**3.2 Шаг 1. Выравнивание потока.**

Сначала к концу потока дописывают единичный бит:

(см. рисунки 3.2.1, 2.2.2).

Затем добавляют некоторое число нулевых бит такое, чтобы новая длина потока стала сравнима с 448 по модулю 512. (см. рисунки 3.2.1, 2.2.2).



Рисунок 3.2.1 – Увеличение длинны сообщения

Деление на 8 используется, т.к. “new\_len” и “initial\_len” переменные типа size\_t который хранит данные в байтах.

Выравнивание происходит в любом случае, даже если длина исходного потока уже сравнима с 448.

После увеличения длинны сообщения, добавляем бит "1" к сообщению

добавляем биты "0" до тех пор, пока длина сообщения в битах

не станет ≡ 448 (mod 512). (см. рисунок 3.2.2).

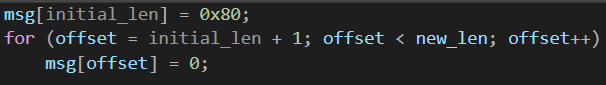


Рисунок 3.2.1 – Выравнивание потока.

Мы добавляем единицу в конец потока

(приставка “0x” в си подобных языках означает 16-ричную систему счисления “HEX”)

Первый бит — старший (порядок байтов little-endian), следовательно используем “”, а не “”

**3.3 Шаг 2. Добавление длины сообщения.**

В конец сообщения дописывают 64-битное представление длины данных (количество бит в сообщении) до выравнивания, каждый символ вводимого сообщения занимает 8 бит. (порядок байтов little-endian)

(см. рисунки 3.3.1, 3.3.2).



Рисунок 3.3.1 –Добавление длинны данных в битах (“первые” 4 байт).

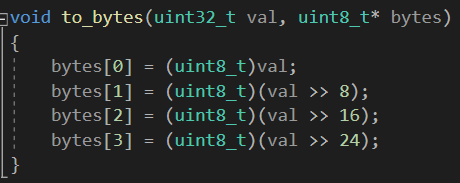


Рисунок 3.3.2 – Выравнивание потока.

Сначала записывают младшие 4 байта, затем старшие. Если длина превосходит , то дописывают только младшие биты (эквивалентно взятию по модулю ). После этого длина потока станет кратной 512.

Побитовый сдвиг вправо используется, чтоб собрать 8 бит в байт

(число 256 не помещается в один байт, т.к. 256 > – 1)

;

bytes[0] = 0;

bytes[1] = 1. (порядок байтов little-endian)

Далее добавляются следующие 4 байт (см. рисунок 3.3.3).



Рисунок 3.3.3 –Добавление длинны данных в битах (“вторые” 4 байт).

initial\_len >> 29 = initial\_len \* 8 >> 32

т.к 32 – 29 = = 8

Не такой большой побитовый сдвиг позволит избежать переполнения.

uint32\_t занимает ровно 4 байта( от 0 до – 1 )

**3.4 Шаг 3. Инициализация буфера.**

Для вычислений инициализируются четыре переменные размером по 32 бита, начальные значения которых задаются шестнадцатеричными числами (порядок байтов little-endian) (см. рисунок 3.4.1).

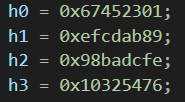


Рисунок 3.4.1 – Инициализация буфера.

В этих переменных будут храниться результаты промежуточных вычислений. Начальное состояние ABCD называется инициализирующим вектором.

Определим таблицу констант K[1..64] — 64-элементная таблица данных, построенная следующим образом: K[n] = floor( \* ).

Константы — это целая часть синусов целых чисел (в радианах) \* 2^32.

(см. рисунок 3.4.2).

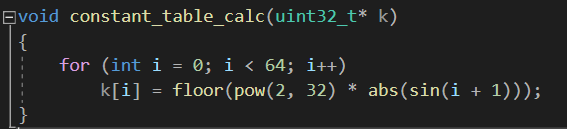


Рисунок 3.4.2 – Построение таблицы констант.

**3.5 Шаг 4. Вычисление в цикле.**

После добавления длины сообщения в его конец, длина потока станет кратной 512. Далее мы обрабатываем сообщение последовательными

512-битными порциями: (см. рисунок 3.5.1).

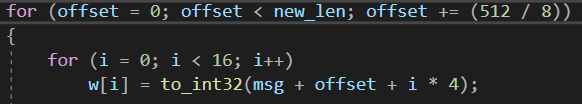


Рисунок 3.5.1 – Основной цикл.

Смещение равно 64 байта.

Так как каждый 512-битный блок проходит 4 этапа вычислений по 16 раундов, мы разбиваем наш блок на шестнадцать 32-битных слов

w[j], 0 ≤ j ≤ 15 (см. рисунки 3.5.1, 3.5.2).

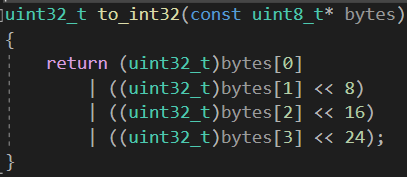


Рисунок 3.5.2 –Конвертация блока в массив uint32[16].

Для каждого раунда потребуется своя функция.

1-й этап: FunF(X, Y, Z) = (XY) (XZ)

2-й этап: FunG(X, Y, Z) = (X Z) (Z Y)

3-й этап: FunH(X, Y, Z) = X YZ

4-й этап: FunI(X, Y, Z) = Y (Z X)

(см. рисунок 3.5.3).

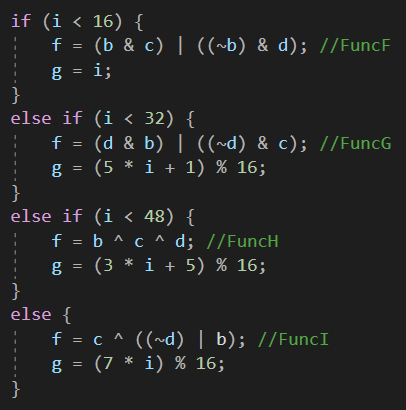


Рисунок 3.5.3 – Применение функций в коде.

Прибавляем результат текущего "куска" к общему результату

(см. рисунок 3.5.4).

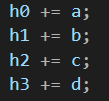


Рисунок 3.5.4 – Суммирование с результатом предыдущего цикла.

**3.6 Шаг 5. Результат вычислений.**

Результат вычислений находится в буфере h0h1h2h3, это и есть хеш. Если выводить побайтово, начиная с младшего байта h1 и заканчивая старшим байтом h3, то мы получим MD5-хеш.

h0 + h1 + h2 + h3 (порядок байтов little-endian)

(см. рисунок 3.6.1).

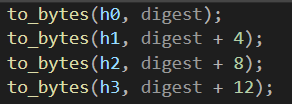
****

Рисунок 3.6.1 – Приведение хеша к “нормальному” виду.

**3.7 Вывод результата.**

После завершения работы функции “MD5” в коде ее результат запишется в массив “result” типа uint8\_t, имея 16 элементов(128 бит, 16 элементов по 8 бит), где каждый элемент хранит в себе 2 символа по 4 бита,

Вывод осуществляется по 2 элемента.

В функции “printf” «%.2x» 2 определяет точность: в результате будет не менее 2 шестнадцатеричных цифр, если представление значения имеет меньше цифр, оно имеет префикс 0. (см. рисунок 3.7.1).

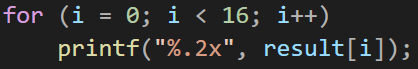


Рисунок 3.6.1 – Вывод MD5 хеша.

**4 Криптоанализ**

**4.1 Извлечение сообщения из хеша.**

MD5 всегда имеет длину 128 бит, это означает, что есть

возможные хеши MD5, при том, что входное сообщение может быть бесконечно большим. Таким образом, существует бесконечное количество возможностей для данных, которые будут хешироваться до одного и того же значения. Но, при этом, невероятно сложно найти две части данных, которые хешируются с одинаковым значением, и вероятность того, что это произойдет случайно, почти равна 0.

То, что длина хеша равна 128 бит, также означает, что как только сообщение было запущено через хэш-функцию, часть данных теряется, что делает невозможным “обратное хеширование” или прямое извлечение из хеша.

**4.2 Подбор сообщений.**

В 2011 году IETF согласилось внести изменения в RFC 1321 (MD5) и RFC 2104 (HMAC-MD5). Так появился документ RFC 6151. Он признает алгоритм шифрования MD5 небезопасным и рекомендует отказаться от его использования. Так почему алгорит небезопасный, если извлечь сообщение из хеша невозможно?

На данный момент существуют несколько видов «взлома» хешей

MD5 — подбора сообщения с заданным хешем:

1. Перебор по словарю, использующий метод полного перебора

(brute-force) предполагаемых паролей путём последовательного пересмотра всех слов. (см. рисунок 4.2.1, 4.2.2).

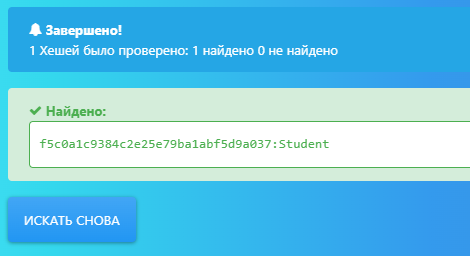


Рисунок 4.2.1 – Онлайн перебор по словарю

1. «Радужные» таблицы — это особый тип словаря, который содержит

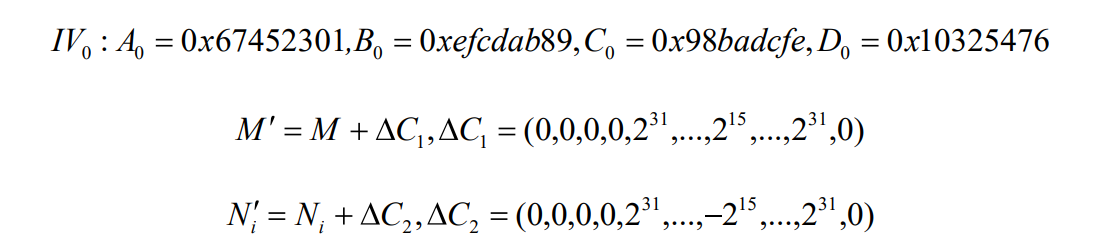
цепочки паролей и позволяет подобрать пароль в течение нескольких секунд или минут

1. Коллизия хеш-функции — это получение одинакового

значения функции для разных сообщений и идентичного начального буфера.

MD5 слаб к сопротивлению к коллизиям, например, простой Pentium 2,4 ГГц может вычислить искусственные хэш-коллизии в течение нескольких секунд.

H. Dobbertin обнаружил коллизию свободного запуска, состоящую из двух разных 512-битных сообщений с выбранным начальным значением





На IBM P690 требуется около одного часа, чтобы найти такие M и M', после этого требуется всего от 15 секунд до 5 минут, чтобы найти N и N', так что (M, N2) и (M', N2) выдадут одно и то же хэш-значение. Более того, эта атака работает для любого заданного начального значения.

Перебор по словарю является очень быстрым методом, он быстрее чем радужные таблицы и активно используется злоумышленниками.

Я попробовал извлечь сообщение из хеша, используя перебор по словарю, с помощью GPU (намного быстрее чем использование CPU) и, загрузив небольшой словарь (133MB), извлек хеш за 3 секунды.

Использовалась программа “hashcat”.

Загрузка словаря в кеш – 1с;

Поиск по словарю брут форсом – 2c.

(см. рисунок 4.2.2).

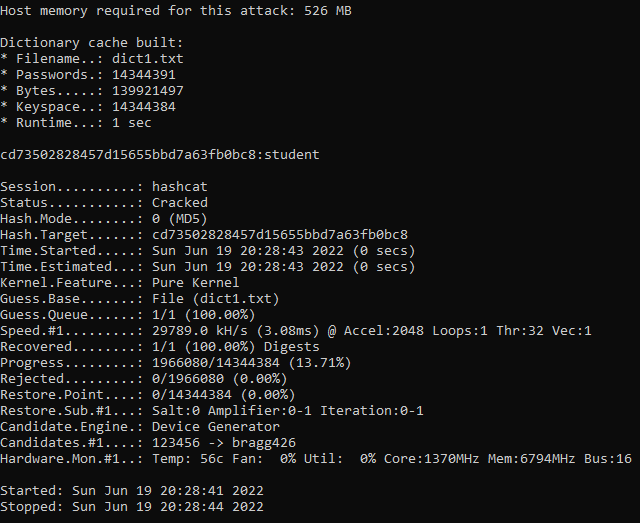


Рисунок 4.2.2 – Использования программы “hashcat”

Видеокарта “NVIDIA GeForce RTX 2060” способна вычислять 25 млрд. MD5 хешей в секунду (см. рисунок 4.2.3).

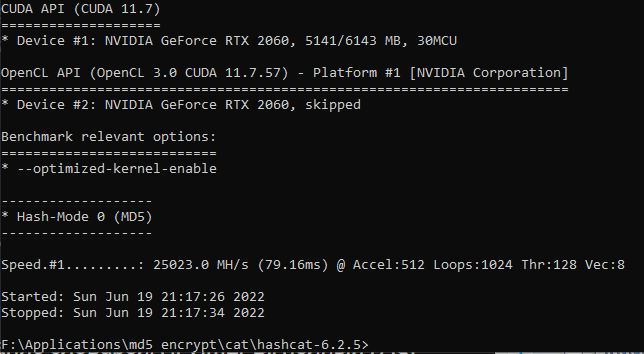


Рисунок 4.2.3 – Тест производительности видеокарты

“NVIDIA GeForce RTX 2060 (CUDA 11.7)” (MD5)

**4.3 Защита дайджестов от перебора.**

Существуют эффективные защиты от перебора хеша

1. Использование “ключа растяжения” делает слабый пароль менее

уязвимым к брут форсу.

Выполнять растяжку ключа можно несколькими способами.

Один из способов - многократно в цикле применять криптографическую хеш-функцию или блочный шифр. Например, в приложениях, где ключ используется для шифра, расписание ключей в шифре может быть изменено таким образом, чтобы для его выполнения требовалось определенное время.

Другой способ - использовать криптографические хеш-функции, которые требуют больших объемов памяти - они могут быть эффективны в предотвращении атак злоумышленников, привязанных к памяти.

1. «Соленое» хеширование. Соль в криптографии - это случайные

данные, которые используются в качестве дополнительных входных данных для хеш-функций. Для каждого пароля случайным образом генерируется новая соль. Обычно соль и пароль (или его версия после растяжения ключа) объединяются и передаются в криптографическую хеш-функцию, а выходное хеш-значение (но не исходный пароль) сохраняется вместе с солью в базе данных.

Радужные таблицы и таблицы поиска работают только потому, что каждый пароль хешируется совершенно одинаковым способом.

Вот пример соления хеша:

saltedhash(password) = md5(password + salt)

или же

saltedhash(password) = md5(md5(password) + salt)

но в тоже время, не стоит использовать слишком “сложные” комбинации хеш-функций.

Ниже приведены примеры неправильных функций:

md5(sha1(password))

md5(md5(salt) + md5(password))

sha1(sha1(password))

sha1(str\_rot13(password + salt))

md5(sha1(md5(md5(password) + sha1(password)) + md5(password)))

Так как взломщик обычно обладает доступом к исходному коду (особенно если это приложение бесплатное или с открытым исходным кодом), и ему известны несколько пар пароль/хеш из целевой системы, поэтому не составляет никакого труда выполнить реверсивное программирование алгоритма.

Таким образом, лучше использовать новые алгоритмы хеширования такие, как SHA256, SHA512, RipeMD, WHIRLPOOL, SHA3, и т.д;

проверенные алгоритмы растяжения ключа, такие как PBKDF2, bcrypt и scrypt;

**5 Область применения хеша**

**5.1. Проверка целостности файлов.**

С помощью MD5 проверяли целостность и подлинность скачанных файлов — так, некоторые программы поставляются вместе со значением контрольной суммы. Например, пакеты для инсталляции свободного ПО

**5.2. Хеширования паролей.**

MD5 использовался для хеширования паролей. В системе UNIX каждый пользователь имеет свой пароль и его знает только пользователь. Для защиты паролей используется хеширование.

Многие системы используют базы данных для аутентификации пользователей и существует несколько способов хранения паролей:

* Пароли хранятся как есть. При взломе такой базы все пароли станут известны.
* Хранятся только хеши паролей. Найти пароли можно, используя заранее подготовленные таблицы хешей. Такие таблицы составляются из хешей простых или популярных паролей.
* К каждому паролю добавляется несколько случайных символов

(соль) и результат хешируется. Полученный хеш вместе с «солью» сохраняются в открытом виде. Найти пароль с помощью таблиц таким методом не получится.

Блок схема алгоритма в приложении А. Исходный код - в приложении Б.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Xie Tao; Fanbao Liu & Dengguo Feng (2013). "Fast Collision Attack on MD5"

https://eprint.iacr.org/2013/170.pdf

[2] Xiaoyun Wang, Dengguo Feng, Xuejia Lai3, Hongbo.

“Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD”.

<https://eprint.iacr.org/2004/199.pdf>

[3] Rivest, R. (April 1992). "Step 4. Process Message in 16-Word Blocks".

The MD5 Message-Digest Algorithm. IETF.

https://datatracker.ietf.org/doc/html/rfc1321

[4] "ISC Diary – Hashing Passwords".

https://www.dshield.org/diary/Hashing+Passwords/11110

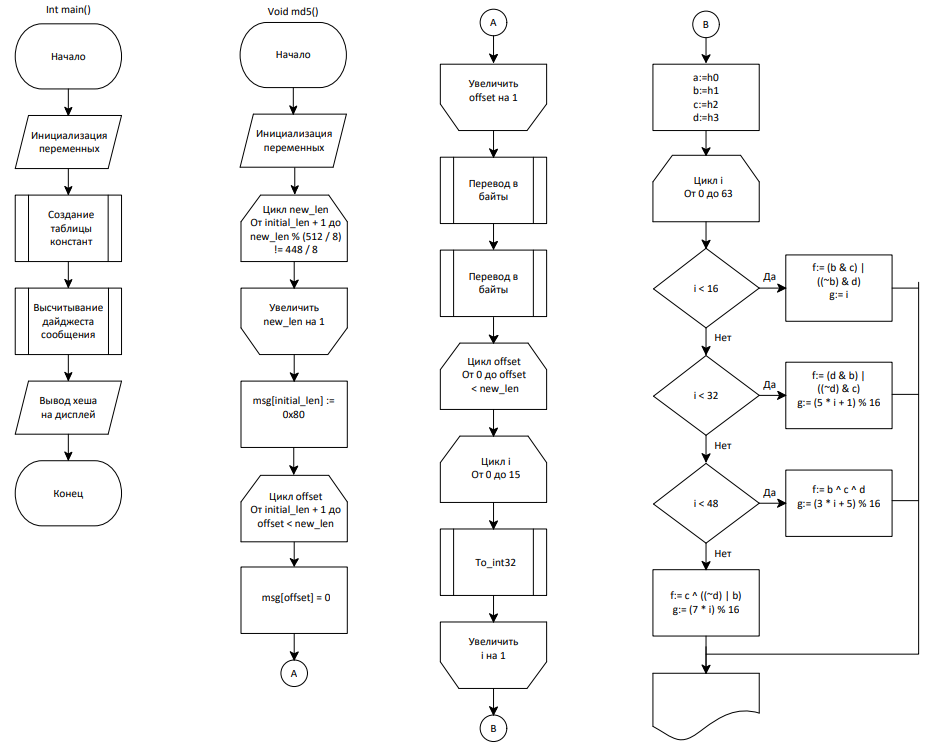
[5] "How Rainbow Tables work".

https://kestas.kuliukas.com/RainbowTables

ПРИЛОЖЕНИЕ А

*(обязательное)*

Блок-схема основного модуля





ПРИЛОЖЕНИЕ Б

*(обязательное)*

Исходный код программы

Модуль *main.cpp*:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdint.h>

#include <locale.h>

#include <math.h>

#include <iostream>

// r указывает количество смен за раунд

const uint32\_t r[] = { 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,

5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20,

4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,

6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21 };

void to\_bytes(uint32\_t val, uint8\_t\* bytes)

{

bytes[0] = (uint8\_t)val;

bytes[1] = (uint8\_t)(val >> 8);

bytes[2] = (uint8\_t)(val >> 16);

bytes[3] = (uint8\_t)(val >> 24);

}

uint32\_t to\_int32(const uint8\_t\* bytes)

{

return (uint32\_t)bytes[0]

| ((uint32\_t)bytes[1] << 8)

| ((uint32\_t)bytes[2] << 16)

| ((uint32\_t)bytes[3] << 24);

}

void all\_free(char\* msg, uint32\_t\* k)

{

free(msg);

free(k);

}

void constant\_table\_calc(uint32\_t\* k)

{

for (int i = 0; i < 64; i++)

k[i] = floor(pow(2, 32) \* abs(sin(i + 1)));

}

void md5(const uint8\_t\* initial\_msg, size\_t initial\_len, uint8\_t\* digest,

uint32\_t\* k)

{

uint8\_t\* msg = NULL;

size\_t new\_len, offset;

uint32\_t w[16];

uint32\_t a, b, c, d, i, f, g, temp;

uint32\_t h0, h1, h2, h3;

h0 = 0x67452301;

h1 = 0xefcdab89;

h2 = 0x98badcfe;

h3 = 0x10325476;

for (new\_len = initial\_len + 1; new\_len % (512 / 8) != 448 / 8;

new\_len++);

msg = (uint8\_t\*)malloc(new\_len + 8);

memcpy(msg, initial\_msg, initial\_len);

msg[initial\_len] = 0x80;

for (offset = initial\_len + 1; offset < new\_len; offset++)

msg[offset] = 0;

to\_bytes(initial\_len \* 8, msg + new\_len);

to\_bytes(initial\_len >> 29, msg + new\_len + 4);

for (offset = 0; offset < new\_len; offset += (512 / 8))

{

for (i = 0; i < 16; i++)

w[i] = to\_int32(msg + offset + i \* 4);

a = h0;

b = h1;

c = h2;

d = h3;

for (i = 0; i < 64; i++)

{

if (i < 16)

{

f = (b & c) | ((~b) & d); //FuncF

g = i;

}

else if (i < 32) {

f = (d & b) | ((~d) & c); //FuncG

g = (5 \* i + 1) % 16;

}

else if (i < 48) {

f = b ^ c ^ d; //FuncH

g = (3 \* i + 5) % 16;

}

else {

f = c ^ ((~d) | b); //FuncI

g = (7 \* i) % 16;

}

temp = d;

d = c;

c = b;

b = b + ((((a + f + k[i] + w[g])) << (r[i])) |

(((a + f + k[i] + w[g])) >> (32 - (r[i]))));

a = temp;

}

h0 += a;

h1 += b;

h2 += c;

h3 += d;

}

free(msg);

to\_bytes(h0, digest);

to\_bytes(h1, digest + 4);

to\_bytes(h2, digest + 8);

to\_bytes(h3, digest + 12);

}

int main()

{

setlocale(LC\_ALL, "Russian");

char\* msg = NULL;

size\_t len;

int i;

uint8\_t result[16];

uint32\_t\* k = NULL;

k = (uint32\_t\*)malloc(64 \* sizeof(uint32\_t));

if (!k)

{

puts("Ошибка выделения памяти");

return 1;

}

constant\_table\_calc(k);

msg = (char\*)malloc(80 \* sizeof(char));

if (!msg)

{

puts("Ошибка выделения памяти");

return 2;

}

puts("Введите строку");

msg = gets\_s(msg, 80);

while(!msg)

{

puts("Неправильный ввод");

msg = gets\_s(msg, 80);

}

len = strlen(msg);

md5((uint8\_t\*)msg, len, result, k);

for (i = 0; i < 16; i++)

printf("%.2x", result[i]);

puts("");

all\_free(msg, k);

return 0;

}