Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчет по лабораторной работе №5

Выполнил:

студент гр. 953504

Кондрашов И.Д.

Проверил:

Протько М.И.

Минск 2022

**Оглавление**

[**1. Постановка задачи** 2](#_Toc116311933)

[**2. Теоретическая сведения** 2](#_Toc116311934)

[**Хэш-функция MD5** 2](#_Toc116311935)

[**3. Блок-схема алгоритма** 4](#_Toc116311936)

[**4.Пример работы программы** 5](#_Toc116311937)

[**5. Код программы** 6](#_Toc116311938)

[**6. Вывод** 7](#_Toc116311939)

# **1. Постановка задачи**

Необходимо реализовать программные средства формирования хэш-функций на примере алгоритма MD5*.*

**ЗАДАНИЕ:**

1. Изучить теоретические сведения.
2. Создать программы, читающие данные из файла и формирующие в качестве выхода дайджест сообщения длиной 128 бит по алгоритму MD5.

# **2. Теоретическая сведения**

## **Хэш-функция MD5**

Алгоритм MD5 получает на входе сообщение произвольной длины и создает в качестве выхода *дайджест сообщения* длиной 128 бит.

Алгоритм состоит из следующих шагов:

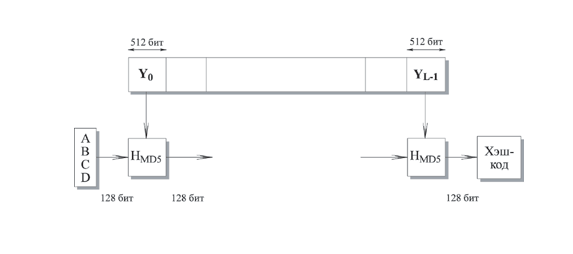


Рисунок 1 –  Логика выполнения MD5

***Шаг 1: добавление недостающих битов***

Сообщение дополняется таким образом, чтобы его длина стала равна 448 по модулю 512 (длина≡448 mod 512).

***Шаг 2: добавление длины***

64-битное представление длины исходного (до добавления) сообщения в битах присоединяется к результату первого шага. Если первоначальная длина больше, чем 264, то используются только последние 64 бита.

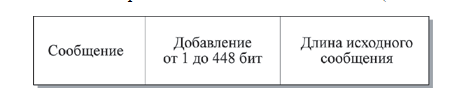


Рисунок 2 –  Структура расширенного сообщения

***Шаг 3: инициализация MD-буфера***

Используется 128-битный буфер для хранения промежуточных и окончательных результатов *хэш-функции*. Буфер может быть представлен как четыре 32-битных регистра (A, B, C, D). Эти регистры инициализируются следующими шестнадцатеричными числами:

А = 01234567; В = 89ABCDEF; C = FEDCBA98; D = 76543210

***Шаг 4: обработка последовательности 512-битных (16-словных) блоков***

Основой алгоритма является модуль, состоящий из четырех циклических обработок, обозначенный как HMD5. Четыре цикла имеют похожую структуру, но каждый цикл использует свою элементарную логическую функцию, обозначаемую fF, fG, fH и fI соответственно.

Каждый цикл принимает в качестве входа текущий 512-битный блок Yq, обрабатывающийся в данный момент, и 128-битное значение буфера ABCD, которое является промежуточным значением *дайджеста*, и изменяет содержимое этого буфера. Каждый цикл также использует четвертую часть 64-элементной таблицы T[1 ... 64], построенной на основе функции sin. i-ый элемент T, обозначаемый T[i], имеет значение, равное целой части от 232 \* abs (sin (i)), i задано в радианах. Так как abs (sin (i)) является числом между 0 и 1, каждый элемент Т является целым, которое может быть представлено 32 битами. Таблица обеспечивает "случайный" набор 32-битных значений, которые должны ликвидировать любую регулярность во входных данных.

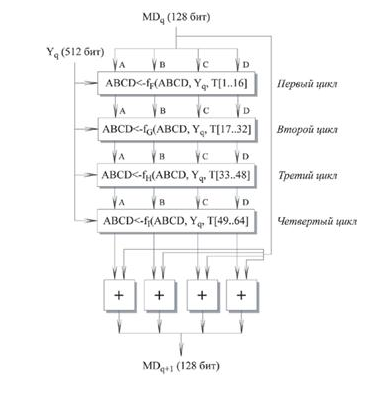
Для получения MDq+1 выход четырех циклов складывается по модулю 232 с MDq. Сложение выполняется независимо для каждого из четырех слов в буфере.

***Шаг 5: выход***

После обработки всех *L* 512-битных блоков выходом *L*-ой стадии является 128-битный *дайджест сообщения*.

# **3. Блок-схема алгоритма**

**Обработка 512-битного блока:**



# **4.Пример работы программы**

Text

Description automatically generated

# **5. Код программы**

from hashlib import sha1  
  
trans\_5C = bytes((x ^ 0x5C) for x in range(256))  
trans\_36 = bytes((x ^ 0x36) for x in range(256))  
  
digest\_size = None  
  
class HMAC:  
 blocksize = 64  
  
 def \_\_init\_\_(self, key, msg=None, digestmod=''):  
 if not isinstance(key, (bytes, bytearray)):  
 raise TypeError("key: expected bytes or bytearray, but got %r" % type(key).\_\_name\_\_)  
  
 if not digestmod:  
 raise TypeError("Missing required parameter 'digestmod'.")  
  
 self.digest\_cons = digestmod  
 self.outer = self.digest\_cons()  
 self.inner = self.digest\_cons()  
 self.digest\_size = self.inner.digest\_size  
  
 blocksize = self.inner.block\_size  
  
 self.block\_size = blocksize  
  
 if len(key) > blocksize:  
 key = self.digest\_cons(key).digest()  
  
 key = key.ljust(blocksize, b'\0')  
 self.outer.update(key.translate(trans\_5C))  
 self.inner.update(key.translate(trans\_36))  
 if msg is not None:  
 self.update(msg)  
  
 def update(self, msg):  
 self.inner.update(msg)  
  
 def \_current(self):  
 h = self.outer.copy()  
 h.update(self.inner.digest())  
 return h  
  
 def digest(self):  
 h = self.\_current()  
 return h.digest()  
  
 def hexdigest(self):  
 h = self.\_current()  
 return h.hexdigest()  
  
def hmac(key, msg=None, digestmod=''):  
 return HMAC(key, msg, digestmod)  
  
def main():  
 text = b"Hello World!"  
 key = b"SECRET KEY"  
 print("Message: %r" % text)  
 print("Key: %r" % key)  
 hashed = hmac(key=key, msg=text, digestmod=sha1)  
 signature = hashed.hexdigest()  
 print("Result: %r" % signature)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

# **6. Вывод**

MD5 — один из серии алгоритмов по построению дайджеста сообщения, разработанный профессором Рональдом Л. Ривестом из Массачусетского технологического института. Был разработан в 1991 году как более надёжный вариант предыдущего алгоритма MD4. Описан в RFC 1321. Позже Гансом Доббертином были найдены недостатки алгоритма MD4.