Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №5

По дисциплине «Методы защиты информации»

По теме «Хэш-функции»

Выполнила:

Студентка гр. 653501

Д. А. Гуринович

Проверил:

В. С. Артемьев

Минск 2019

Содержание

[1. Постановка задачи](#_2et92p0)

2. Краткие теоретические сведения

3. Схема алгоритма

4[. Результаты выполнения](#_tyjcwt)

5[. Вывод](#_3dy6vkm)

[ПРИЛОЖЕНИЕ 1. Исходный код программы.](#_1t3h5sf)

# 1. Постановка задачи

В современном мире остро стоит вопрос о безопасности. Обеспечение безопасности является важным аспектом деятельности любой компании. Для обеспечения безопасности используется множество различных средств, как аппаратных, так и программных. Одним из таких средств является хэширование данных.

Хеш-функция, или функция свёртки — функция, осуществляющая преобразование [массива](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B8%D0%B2_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) входных данных произвольной длины в (выходную) [битовую](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82) строку установленной длины, выполняемое [определённым алгоритмом](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%82%D0%B5%D1%80%D0%BC%D0%B8%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC). Преобразование, производимое хеш-функцией, называется хешированием. Исходные данные называются входным массивом, «ключом» или «сообщением». Результат преобразования (выходные данные) называется «[хешем](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88-%D1%81%D1%83%D0%BC%D0%BC%D0%B0)», «хеш-кодом», «хеш-суммой», «сводкой [сообщения](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%BE%D0%B1%D1%89%D0%B5%D0%BD%D0%B8%D0%B5)».

В ходе данной лабораторной работы необходимо изучить теоретическую часть о хэшировании и алгоритме HMAC, на основании которых создать приложение, использующие данные технологии.

**2. Краткие теоретические сведения**

MD5 ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Message Digest 5) — 128-битный алгоритм [хеширования](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), разработанный профессором [Рональдом Л. Ривестом](https://ru.wikipedia.org/wiki/%D0%A0%D0%B8%D0%B2%D0%B5%D1%81%D1%82,_%D0%A0%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%B4) из [Массачусетского технологического института](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B0%D1%87%D1%83%D1%81%D0%B5%D1%82%D1%81%D0%BA%D0%B8%D0%B9_%D1%82%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%81%D1%82%D0%B8%D1%82%D1%83%D1%82) (Massachusetts Institute of Technology, MIT) в [1991 году](https://ru.wikipedia.org/wiki/1991_%D0%B3%D0%BE%D0%B4). Предназначен для создания «отпечатков» или [дайджестов сообщения](https://ru.wikipedia.org/wiki/%D0%94%D0%B0%D0%B9%D0%B4%D0%B6%D0%B5%D1%81%D1%82_%D1%81%D0%BE%D0%BE%D0%B1%D1%89%D0%B5%D0%BD%D0%B8%D1%8F) произвольной длины и последующей проверки их подлинности. Широко применялся для проверки [целостности информации](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BB%D0%BE%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%B8) и хранения хешей паролей.

Алгоритм состоит из пяти шагов:

1)Append Padding Bits

В исходную строку дописывают единичный байт 0х80, а затем дописывают нулевые биты, до тех пор, пока длина сообщения не будет сравнима с 448 по модулю 512. То есть дописываем нули до тех пор, пока длина нового сообщения не будет равна [длина] = (512\*N+448), где N — любое натуральное число, такое, что это выражение будет наиболее близко к длине блока.

2)Append Length

Далее в сообщение дописывается 64-битное представление длины исходного сообщения.

3)Initialize MD Buffer

На этом шаге инициализируется буффер

word A: 01 23 45 67

word B: 89 ab cd ef

word C: fe dc ba 98

word D: 76 54 32 10

Как можно заметить буффер состоит из четырех констант, предназначенный для сбора хэша.

4)Process Message in 16-Word Blocks

На четвертом шаге в первую очередь определяется 4 вспомогательные логические функции, которые преобразуют входные 32-битные слова, в, как ни странно, в 32-битные выходные.

F(X,Y,Z) = XY v not(X) Z

G(X,Y,Z) = XZ v Y not(Z)

H(X,Y,Z) = X xor Y xor Z

I(X,Y,Z) = Y xor (X v not(Z))

Также на этом шаге реализуется так называемый «белый шум» — усиление алгоритма, состоящее 64 элементного массива, содержащего псевдослучайные числа, зависимые от синуса числа i:

T[i]=4,294,967,296\*abs(sin(i))

Далее начинается «магия». Копируем каждый 16-битный блок в массив X[16] и производим манипуляции:

AA = A

BB = B

CC = C

DD = D

Затем происходят «чудесные» преобразования-раунды, которых всего будет 4. Каждый раунд состоит из 16 элементарных преобразований, которые в общем виде можно представить в виде [abcd k s i], которое, в свою очередь, можно представить как A = B + ((A + F(B,C,D) + X[k] + T[i]) <<< s), где

A, B, C, D — регистры

F(B,C,D) — одна из логических функций

X[k] — k-тый элемент 16-битного блока.

T[i] — i-тый элемент таблицы «белого шума»

<<< s — операция циклического сдвига на s позиций влево.

Ну и в конце суммируем результаты вычислений:

A = A + AA

B = B + BB

C = C + CC

D = D + DD

5) Output

Выводя побайтово буффер ABCD начиная с A и заканчивая D получим наш хэш.

HMAC — в [информатике](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0) ([криптографии](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F)), один из механизмов проверки целостности информации, позволяющий гарантировать то, что данные, передаваемые или хранящиеся в ненадёжной среде, не были изменены посторонними лицами (см. [человек посередине](https://ru.wikipedia.org/wiki/%D0%A7%D0%B5%D0%BB%D0%BE%D0%B2%D0%B5%D0%BA_%D0%BF%D0%BE%D1%81%D0%B5%D1%80%D0%B5%D0%B4%D0%B8%D0%BD%D0%B5)). Механизм HMAC использует [MAC](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%B8%D1%82%D0%BE%D0%B2%D1%81%D1%82%D0%B0%D0%B2%D0%BA%D0%B0), описан в [RFC 2104](https://tools.ietf.org/html/rfc2104), в стандартах организаций [ANSI](https://ru.wikipedia.org/wiki/ANSI), [IETF](https://ru.wikipedia.org/wiki/IETF), [ISO](https://ru.wikipedia.org/wiki/ISO) и [NIST](https://ru.wikipedia.org/wiki/NIST). MAC — стандарт, описывающий способ обмена данными и способ проверки целостности передаваемых данных с использованием секретного ключа. Два клиента, использующие HMAC, как правило, разделяют общий секретный ключ. HMAC — надстройка над MAC; механизм обмена данными с использованием секретного ключа (как в MAC) и [хеш-функций](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F). В зависимости от используемой хеш-функции выделяют HMAC-[MD5](https://ru.wikipedia.org/wiki/MD5), HMAC-[SHA1](https://ru.wikipedia.org/wiki/SHA-1), HMAC-[RIPEMD128](https://ru.wikipedia.org/wiki/RIPEMD-128), HMAC-[RIPEMD160](https://ru.wikipedia.org/wiki/RIPEMD-160) и т. п.

Реализация HMAC является обязательной ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) mandatory to implement) для протокола [IPsec](https://ru.wikipedia.org/wiki/IPsec).

HMAC используется и в других протоколах [интернета](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82), например, [TLS](https://ru.wikipedia.org/wiki/Transport_Layer_Security). Ожидается, что TLS вскоре заменит [SSL](https://ru.wikipedia.org/wiki/Secure_Sockets_Layer) и [SET](https://ru.wikipedia.org/wiki/Secure_Electronic_Transaction).

**3. Схема алгоритма**

# 

# 4. Результаты выполнения

# 

Рисунок 4.1. Исходные данные

# 

Рисунок 4.2. Результат работы программы

**5. Вывод**

В рамках данной лабораторной работы был рассмотрены механизм проверки целостности информации НМАС и алгоритм хэширования MD5.

Для демонстрации работы описанных выше алгоритмов было разработано и реализовано приложение с использованием указанного выше алгоритма.

# ПРИЛОЖЕНИЕ 1. ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ

import struct

from enum import Enum

from math import (

floor,

sin,

)

from bitarray import bitarray

class MD5Buffer(Enum):

A = 0x67452301

B = 0xEFCDAB89

C = 0x98BADCFE

D = 0x10325476

class MD5(object):

\_string = None

\_buffers = {

MD5Buffer.A: None,

MD5Buffer.B: None,

MD5Buffer.C: None,

MD5Buffer.D: None,

}

@classmethod

def hash(cls, string):

cls.\_string = string

preprocessed\_bit\_array = cls.\_step\_2(cls.\_step\_1())

cls.\_step\_3()

cls.\_step\_4(preprocessed\_bit\_array)

return cls.\_step\_5()

@classmethod

def \_step\_1(cls):

bit\_array = bitarray(endian="big")

bit\_array.frombytes(cls.\_string.encode("utf-8"))

bit\_array.append(1)

while bit\_array.length() % 512 != 448:

bit\_array.append(0)

return bitarray(bit\_array, endian="little")

@classmethod

def \_step\_2(cls, step\_1\_result):

length = (len(cls.\_string) \* 8) % pow(2, 64)

length\_bit\_array = bitarray(endian="little")

length\_bit\_array.frombytes(struct.pack("<Q", length))

result = step\_1\_result.copy()

result.extend(length\_bit\_array)

return result

@classmethod

def \_step\_3(cls):

for buffer\_type in cls.\_buffers.keys():

cls.\_buffers[buffer\_type] = buffer\_type.value

@classmethod

def \_step\_4(cls, step\_2\_result):

F = lambda x, y, z: (x & y) | (~x & z)

G = lambda x, y, z: (x & z) | (y & ~z)

H = lambda x, y, z: x ^ y ^ z

I = lambda x, y, z: y ^ (x | ~z)

rotate\_left = lambda x, n: (x << n) | (x >> (32 - n))

modular\_add = lambda a, b: (a + b) % pow(2, 32)

T = [floor(pow(2, 32) \* abs(sin(i + 1))) for i in range(64)]

N = len(step\_2\_result) // 32

for chunk\_index in range(N // 16):

start = chunk\_index \* 512

X = [step\_2\_result[start + (x \* 32) : start + (x \* 32) + 32] for x in range(16)]

X = [int.from\_bytes(word.tobytes(), byteorder="little") for word in X]

A = cls.\_buffers[MD5Buffer.A]

B = cls.\_buffers[MD5Buffer.B]

C = cls.\_buffers[MD5Buffer.C]

D = cls.\_buffers[MD5Buffer.D]

for i in range(4 \* 16):

if 0 <= i <= 15:

k = i

s = [7, 12, 17, 22]

temp = F(B, C, D)

elif 16 <= i <= 31:

k = ((5 \* i) + 1) % 16

s = [5, 9, 14, 20]

temp = G(B, C, D)

elif 32 <= i <= 47:

k = ((3 \* i) + 5) % 16

s = [4, 11, 16, 23]

temp = H(B, C, D)

elif 48 <= i <= 63:

k = (7 \* i) % 16

s = [6, 10, 15, 21]

temp = I(B, C, D)

temp = modular\_add(temp, X[k])

temp = modular\_add(temp, T[i])

temp = modular\_add(temp, A)

temp = rotate\_left(temp, s[i % 4])

temp = modular\_add(temp, B)

A = D

D = C

C = B

B = temp

cls.\_buffers[MD5Buffer.A] = modular\_add(cls.\_buffers[MD5Buffer.A], A)

cls.\_buffers[MD5Buffer.B] = modular\_add(cls.\_buffers[MD5Buffer.B], B)

cls.\_buffers[MD5Buffer.C] = modular\_add(cls.\_buffers[MD5Buffer.C], C)

cls.\_buffers[MD5Buffer.D] = modular\_add(cls.\_buffers[MD5Buffer.D], D)

@classmethod

def \_step\_5(cls):

A = struct.unpack("<I", struct.pack(">I", cls.\_buffers[MD5Buffer.A]))[0]

B = struct.unpack("<I", struct.pack(">I", cls.\_buffers[MD5Buffer.B]))[0]

C = struct.unpack("<I", struct.pack(">I", cls.\_buffers[MD5Buffer.C]))[0]

D = struct.unpack("<I", struct.pack(">I", cls.\_buffers[MD5Buffer.D]))[0]

return bytes(f"{format(A, '08x')}{format(B, '08x')}{format(C, '08x')}{format(D, '08x')}", 'utf-8')

def hmac(message, key, block\_size=64):

if len(key) > block\_size:

key = MD5.hash(key)

if type(key) is str:

key = bytes(key, 'utf-8')

if len(key) < block\_size:

key += b'\0' \* (block\_size - len(key))

i\_pad = [0x36 \* block\_size]

o\_pad = [0x5c \* block\_size]

i\_key\_pad = ''.join([chr(x ^ y) for x, y in zip(key, i\_pad)])

o\_key\_pad = ''.join([chr(x ^ y) for x, y in zip(key, o\_pad)])

i\_key\_pad\_message = str(MD5.hash(i\_key\_pad + message))

result = MD5.hash(o\_key\_pad + i\_key\_pad\_message)

return result