Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы защиты информации

ОТЧЕТ

к лабораторной работе №5

на тему

**Хэш-функции.**

Выполнил Д.С. Шевцова

Проверил

Минск 2024

СОДЕРЖАНИЕ

[Введение 3](#_Toc158758843)

[1 Краткие теоретические сведения 4](#_Toc158758844)

[2 Результаты выполнения лабораторной работы](#_Toc158758845) 6

[Заключение](#_Toc158758846) 8

[Приложение А](#_Toc158758848) [(обязательное)](#_Toc158758849) [Листинг кода](#_Toc158758850) 9

## ВВЕДЕНИЕ

В данной лабораторной работе нужно реализовать программное средство контроля целостности сообщений с помощью вычисления хэш-функции и алгоритма ГОСТ 34.11 и SHA-1.

## 1 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Хеш-функция, или функция свёртки – функция, осуществляющая преобразование массива входных данных произвольной длины в (выходную) битовую строку установленной длины, выполняемое определённым алгоритмом. Преобразование, производимое хеш-функцией, называется хешированием. Исходные данные называются входным массивом, «ключом» или «сообщением». Результат преобразования (выходные данные) называется «хешем», «хеш-кодом», «хеш-суммой», «сводкой сообщения».

Хеш-функции применяются в следующих случаях:

* при построении ассоциативных массивов;
* при поиске дубликатов в сериях наборов данных;
* при построении уникальных идентификаторов для наборов данных;
* при вычислении контрольных сумм от данных (сигнала) для последующего обнаружения в них ошибок (возникших случайно или внесённых намеренно), возникающих при хранении и/или передаче данных;
* при сохранении паролей в системах защиты в виде хеш-кода (для восстановления пароля по хеш-коду требуется функция, являющаяся обратной по отношению к использованной хеш-функции);
* при выработке электронной подписи (на практике часто подписывается не само сообщение, а его «хеш-образ»);

и др.

В общем случае (согласно принципу Дирихле) нет однозначного соответствия между хеш-кодом (выходными данными) и исходными (входными) данными. Возвращаемые хеш-функцией значения (выходные данные) менее разнообразны, чем значения входного массива (входные данные). Случай, при котором хеш-функция преобразует более чем один массив входных данных в одинаковые сводки, называется «коллизией». Вероятность возникновения коллизий используется для оценки качества хеш-функций.

Существует множество алгоритмов хеширования, отличающихся различными свойствами. Примеры свойств:

* разрядность;
* вычислительная сложность;
* криптостойкость.

MD5 (англ. Message Digest 5) — 128-битный алгоритм хеширования, разработанный профессором Рональдом Л. Ривестом из Массачусетского технологического института (Massachusetts Institute of Technology, MIT) в 1991 году. Предназначен для создания «отпечатков» или дайджестов сообщения произвольной длины и последующей проверки их подлинности. Широко применялся для проверки целостности информации и хранения хешей паролей.

HMAC (сокращение от англ. hash-based message authentication code, код аутентификации (проверки подлинности) сообщений, использующий хеш-функции) — в информатике (криптографии), один из механизмов проверки целостности информации, позволяющий гарантировать то, что данные, передаваемые или хранящиеся в ненадёжной среде, не были изменены посторонними лицами.

Преимущества HMAC:

1 Возможность использования хэш-функций, уже имеющихся в программном продукте;

2 Отсутствие необходимости внесения изменений в реализации существующих хэш-функции (внесение изменений может привести к ухудшению производительности и криптостойкости);

3 Возможность замены хэш-функции в случае появления более безопасной или более быстрой хэш-функции.

Механизм HMAC был описан в стандартах организаций ANSI, IETF, ISO и NIST. Реализация HMAC является для протокола IPsec. HMAC используется и в других протоколах интернета, например, TLS.

Ниже на рисунке 1.1 представлена блок схема алгоритма.

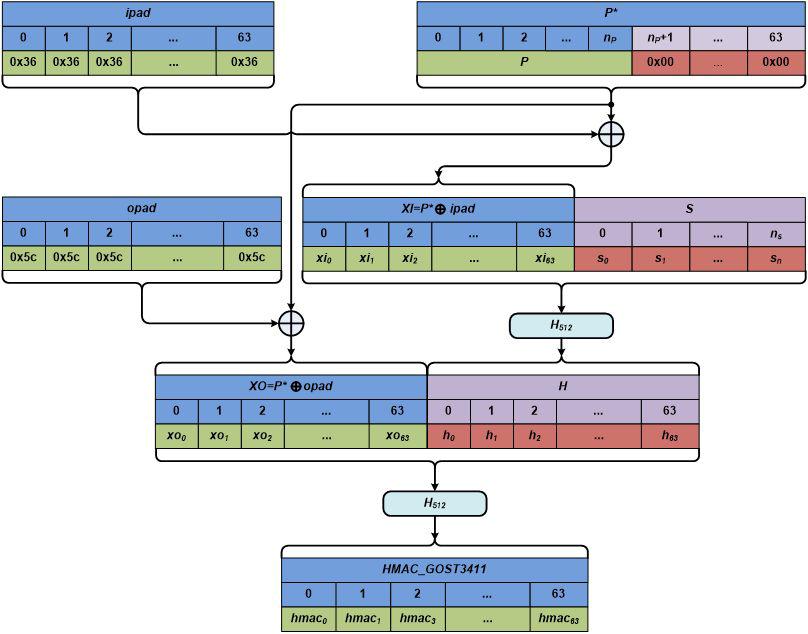


Рисунок 1.1 – Блок схема алгоритма

# 2 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

Для шифрования и дешифрования файлом используется графический интерфейс, где указывается режим программы, а именно ГОСТ или SHA-1. На рисунке 2.1 указаны данные для ГОСТ, а на рисунке 2.2 указаны данные для работы с SHA-1.

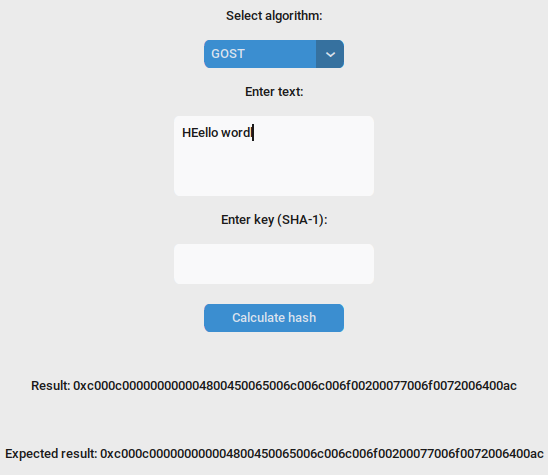


Рисунок 2.1 – Работа в режиме ГОСТ



Рисунок 2.2 – Работа в режиме SHA-1

## ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы была написана реализация программного средства контроля целостности сообщений с помощью вычисления хэш-функции и алгоритма ГОСТ 34.11 и SHA-1 с графическим интерфейсом.

# **ПРИЛОЖЕНИЕ А**

## (обязательное)

## Листинг кода

gost\_file.py

CHANGE\_TABLE = [

[4, 10, 9, 2, 13, 8, 0, 14, 6, 11, 1, 12, 7, 15, 5, 3],

[14, 11, 4, 12, 6, 13, 15, 10, 2, 3, 8, 1, 0, 7, 5, 9],

[5, 8, 1, 13, 10, 3, 4, 2, 14, 15, 12, 7, 6, 0, 9, 11],

[7, 13, 10, 1, 0, 8, 9, 15, 14, 4, 6, 12, 11, 2, 5, 3],

[6, 12, 7, 1, 5, 15, 13, 8, 4, 10, 9, 14, 0, 3, 11, 2],

[4, 11, 10, 0, 7, 2, 1, 13, 3, 6, 8, 5, 9, 12, 15, 14],

[13, 11, 4, 1, 3, 15, 5, 9, 0, 10, 14, 7, 6, 8, 2, 12],

[1, 15, 13, 0, 5, 7, 10, 4, 9, 2, 3, 14, 6, 11, 8, 12],

]

def a\_func(block):

x4 = (block >> 192) & 0xFFFFFFFFFFFFFFFF

x3 = (block >> 128) & 0xFFFFFFFFFFFFFFFF

x2 = (block >> 64) & 0xFFFFFFFFFFFFFFFF

x1 = block & 0xFFFFFFFFFFFFFFFF

fin\_block = ((x1 ^ x2) << 192) | (x4 << 128) | (x3 << 64) | x2

return fin\_block

def p\_func(block):

def fi\_func(arg):

i = arg & 0x03

k = (arg >> 2) + 1

fi = (i << 3) + k - 1

return fi

block\_list = []

for i in range(32):

byte\_value = (block >> (i \* 8)) & 0xFF

block\_list.append(byte\_value)

fin\_block = 0

for i in range(31, -1, -1):

fin\_block = (fin\_block << 8) | block\_list[fi\_func(i)]

return fin\_block

u = h\_in

v = block

w = u ^ v

keys.append(p\_func(w))

for i in range(2, 5):

u = a\_func(u) ^ cs[i - 2]

v = a\_func(a\_func(v))

w = u ^ v

keys.append(p\_func(w)

return blocks, len(current\_block)

def basic\_crypt\_step(part\_key, block, change\_table):

module = 4294967296

lower\_32\_bits = block & 0xFFFFFFFF

upper\_32\_bits = (block >> 32) & 0xFFFFFFFF

s = (lower\_32\_bits + part\_key) % module

sn = [(s >> (4 \* i)) & 0xF for i in range(8)]

for i in range(8):

row = i

col = sn[i]

sn[i] = change\_table[row][col]

s = 0

for i in range(8):

s = (s << 4) | sn[i]

s = ((s << 11) | (s >> (32 - 11))) & 0xFFFFFFFF

s = s ^ upper\_32\_bits

upper\_32\_bits = lower\_32\_bits

lower\_32\_bits = s

result = (lower\_32\_bits << 32) | upper\_32\_bits

return result

def encrypt\_cycle(key, block, change\_table):

keys = [(key >> (32 \* i)) & 0xFFFFFFFF for i in range(8)]

for i in range(24):

key\_index = i % 8

block = basic\_crypt\_step(keys[key\_index], block, change\_table)

for i in range(7, -1, -1):

block = basic\_crypt\_step(keys[i], block, change\_table)

return block

def decrypt\_cycle(key, block, change\_table):

keys = [(key >> (32 \* i)) & 0xFFFFFFFF for i in range(8)]

for i in range(8):

block = basic\_crypt\_step(keys[i], block, change\_table)

for \_ in range(3):

for i in range(7, -1, -1):

block = basic\_crypt\_step(keys[i], block, change\_table)

return block

def compute\_gost\_hash(input\_string):

bin\_text = [hex(ord(elem)) for elem in input\_string]

blocks, last\_len = split\_string\_into\_blocks(bin\_text, 256)

return hex(hash\_gost(blocks, CHANGE\_TABLE, last\_len))

main.py

def calculate\_hash():

algorithm = algo\_var.get()

text = text\_entry.get("1.0", "end-1c")

key = key\_entry.get("1.0", "end-1c") if algorithm == "SHA-1" else None

if algorithm == "GOST":

result = compute\_gost\_hash(text)

expected\_result = compute\_gost\_hash(text)

elif algorithm == "SHA-1":

result = compute\_sha1\_hash(text, key)

expected\_result = HMAC(key.encode(), msg=text.encode(), digestmod=sha1).hexdigest()

else:

result = "Unknown Algorithm"

expected\_result = None

result\_label.configure(text=f"Result: {result}")

if expected\_result:

expected\_label.configure(text=f"Expected result: {expected\_result}")

def update\_key\_entry\_state(\*args):

if algo\_var.get() == "GOST":

key\_entry.configure(state='disabled')

else:

key\_entry.configure(state='normal')

app = ctk.CTk()

app.title("Hash Calculator")

app.geometry("400x400")

hash\_button = ctk.CTkButton(app, text="Calculate hash", command=calculate\_hash)

hash\_button.pack(pady=20)

result\_label = ctk.CTkLabel(app, text="Result:")

result\_label.pack(pady=20)

expected\_label = ctk.CTkLabel(app, text="Expected result:")

expected\_label.pack(pady=20)

update\_key\_entry\_state() # Initial state of the key entry

app.mainloop()