Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №2

За пятый семестр

По дисциплине: «Криптографические методы защиты информации»

**Тема: «Криптографические хэш-функции»**

Выполнила:

Студент 3 курса

Группы ИИ-21(II)

Кирилович А. А.

Проверила:

Хацкевич А. С.

Брест 2023

**Цель:** изучить существующие алгоритмы вычисления дайджестов сообщений и написать программу, реализующую заданный алгоритм хэширования.

**Ход работы:**

**Вариант 10** (алгоритм SHA–224)

**Задание:**

Реализовать алгоритмы подсчёта значения хэша согласно варианту без использования сторонних средств, исследовать лавинный эффект. Реализовать подсчёт хэша с использованием Openssl.

Листинг(без Openssl):

mod lib;

use lib::{Digest, Ripemd256};

use std::env;

use std::fs;

use std::io::{self, Read};

const BUFFER\_SIZE: usize = 1024;

/// Print digest result as hex string and name pair

fn print\_result(sum: &[u8], name: &str) {

for byte in sum {

print!("{:02x}", byte);

}

println!("\t{}", name);

}

/// Compute digest value for given `Reader` and print it

/// On any error simply return without doing anything

fn process<D: Digest + Default, R: Read>(reader: &mut R, name: &str) {

let mut sh = D::default();

let mut buffer = [0u8; BUFFER\_SIZE];

loop {

let n = match reader.read(&mut buffer) {

Ok(n) => n,

Err(\_) => return,

};

sh.update(&buffer[..n]);

if n == 0 || n < BUFFER\_SIZE {

break;

}

}

print\_result(&sh.finalize(), name);

}

fn main() {

let args: Vec<String> = env::args().collect();

// Process files listed in command line arguments one by one

// If no files provided process input from stdin

if args.len() > 2 {

let path = args[1].clone();

match args[2].clone().parse::<i32>() {

Ok(rev) => {

if let Ok(mut file) = fs::File::open(&path) {

let mut contents = String::new();

let \_ = file.read\_to\_string(&mut contents);

if let Some(byte\_index) = rev.checked\_div(8) {

if let Some(byte\_index\_usize) =

<i32 as TryInto<usize>>::try\_into(byte\_index).ok() {

if byte\_index\_usize < contents.len() {

let mut byte = contents.as\_bytes()[byte\_index\_usize];

let bit\_to\_set = rev % 8;

byte ^= 1 << bit\_to\_set;

unsafe {

contents.as\_bytes\_mut()[byte\_index\_usize] = byte;

}

}

} else {

eprintln!("The bit index is beyond the length of the string.");

}

} else {

eprintln!("Bit index must be greater than or equal to 0.");

}

process::<Ripemd256, \_>(&mut contents.as\_bytes(), &path);

} else {

println!("Failed to open file: {}", path);

}

}

Err(e) => {

eprintln!("{}", e);

}

}

} else if args.len() == 2 {

let path = args[1].clone();

if let Ok(mut file) = fs::File::open(&path) {

process::<Ripemd256, \_>(&mut file, &path);

}

} else {

process::<Ripemd256, \_>(&mut io::stdin(), "-");

}

}

Результат:

f3e98d1a18a7c168c397f043c7497f71eca86420fdb9753013579bde02468ace  
f3e98d1a18a7c168c397f043c7497f717b7bd240b4ac33a6597c43004be6affe  
82f0b26237c8b3d73a9c94917f5f331702f89c29b4ac33a6597c43004be6affe  
82f0b26237c8b3d73a9c94917f5f3317840d57a117f63bd9429b1b059d3bfcfa  
ee63bc8a5740ace91450128c55b4d833840d57a146d7c2e6429b1b059d3bfcfa  
ee63bc8a5740ace91450128c55b4d833b39475584fc7804ef34420d18083b571  
3d61697bf3254f58c625022fc9ed770eb39475584fc7804e0541037d8083b571  
3d61697bf3254f58c625022fc9ed770e3305cbb4a59014ca2e3e25eacfa2b9e7

c13b8c02750b9ec36e252110bb1d41b1eca86420fdb9753013579bde02468ace  
c13b8c02750b9ec36e252110bb1d41b146a515969fe1fae1e30dc5327d939f07  
26ca7a4e0514f056f3602d9b5587e592d04a9b119fe1fae1e30dc5327d939f07  
26ca7a4e0514f056f3602d9b5587e592420f37c3f2f0ca42e342215cb8ce453c  
68fd49293c93f0ba4a1fd90f8952d4ed420f37c31423ff65e342215cb8ce453c  
68fd49293c93f0ba4a1fd90f8952d4edc4088f7ddc124cbb71b6dffa2437f0dd  
524344c9a88016666a9c2f758948004ec4088f7ddc124cbb3b10ca002437f0dd  
524344c9a88016666a9c2f758948004eed794c345bd4e7219e69656e54fc55b9

**Вывод:** в ходе лабораторной работы изучила криптографические хэш-функции.