**Министерство науки и высшего образования Российской Федерации**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Факультет безопасности информационных технологий**

**Дисциплина:**

«Криптографические методы обеспечения информационной безопасности»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6**

«Модель протокола защищенного соединения»

**Выполнили:**

Чу Ван Доан, студент группы номер N3347

**

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

**Проверил:**

Таранов Сергей Владимирович

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(отметка о выполнении)

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

# Содержание

[**Содержание**](#_heading=h.4m0x322iz9pr) **2**

[**Введение**](#_heading=h.3b2hjuxkd832) **3**

[**Ход работы**](#_heading=h.mo1k5b1sfa1u) **4**

[1. Программа](#_heading=h.l877kk81rgrk) 4

[2. Выполнение программы](#_heading=h.l877kk81rgrk) 6

[2.1. Вычисление SHA-256 строки](#_heading=h.bp9f3k7bcgy8) 6

[2.2. Проверка SHA-256](#_heading=h.xpmlkfyjs5qq) 6

[2.3. Создание HMAC-SHA256](#_heading=h.ophi36r82h0f) 6

[2.4. Проверка HMAC-SHA256](#_heading=h.gmp9jljniesq) 7

[**Заключение**](#_heading=h.bj7dv4cpq31f) **8**

# Введение

Цель: изучить подходы к применению криптопримитивов в рамках протоколов для защищенных соединений.

Необходимое программное обеспечение: В рамках задания необходим установленный openssl, который может быть установлен отдельно или как составляющая сборки (например, kali linux)

# Ход работы

## Программа

#!/usr/bin/env python3

import argparse

import hashlib

import hmac

import sys

def sha256\_hash(data: bytes) -> str:

"""Compute SHA-256 of the data and return the hex string."""

return hashlib.sha256(data).hexdigest()

def verify\_sha256\_hash(data: bytes, expected\_hex: str) -> bool:

"""Verify whether SHA-256 of the data matches the expected hex value."""

return sha256\_hash(data) == expected\_hex.lower()

def hmac\_sha256(key: bytes, data: bytes) -> str:

"""Generate HMAC-SHA256 from key and data, return the hex string."""

block\_size = hashlib.sha256().block\_size

if len(key) > block\_size:

key = hashlib.sha256(key).digest()

key = key.ljust(block\_size, b'\x00')

o\_key = bytes((b ^ 0x5C) for b in key)

i\_key = bytes((b ^ 0x36) for b in key)

inner = hashlib.sha256(i\_key + data).digest()

return hashlib.sha256(o\_key + inner).hexdigest()

def verify\_hmac\_sha256(key: bytes, data: bytes, expected\_hex: str) -> bool:

"""Verify whether HMAC-SHA256 of the data with the key matches expected\_hex."""

calc = hmac\_sha256(key, data)

# secure comparison

return hmac.compare\_digest(calc, expected\_hex.lower())

def load\_data(args) -> bytes:

if args.file:

return open(args.file, 'rb').read()

else:

return args.message.encode()

def main():

p = argparse.ArgumentParser(

description="Tool for SHA-256 hashing and HMAC-SHA256 generation/verification")

sub = p.add\_subparsers(dest='cmd', required=True)

# sha256

ph = sub.add\_parser('hash', help='Compute SHA-256')

ph.add\_argument('-m', '--message', help='Input string', default='')

ph.add\_argument('-f', '--file', help='Input file')

pv = sub.add\_parser('verify-hash', help='Verify SHA-256')

pv.add\_argument('-m', '--message', help='Input string', default='')

pv.add\_argument('-f', '--file', help='Input file')

pv.add\_argument('-s', '--sha', help='Expected SHA-256 value', required=True)

# hmac

pm = sub.add\_parser('hmac', help='Generate HMAC-SHA256')

pm.add\_argument('-k', '--key', help='HMAC key', required=True)

pm.add\_argument('-m', '--message', help='Input string', default='')

pm.add\_argument('-f', '--file', help='Input file')

pv2 = sub.add\_parser('verify-hmac', help='Verify HMAC-SHA256')

pv2.add\_argument('-k', '--key', help='HMAC key', required=True)

pv2.add\_argument('-m', '--message', help='Input string', default='')

pv2.add\_argument('-f', '--file', help='Input file')

pv2.add\_argument('-t', '--tag', help='Expected HMAC value', required=True)

args = p.parse\_args()

data = load\_data(args)

if args.cmd == 'hash':

print(sha256\_hash(data))

elif args.cmd == 'verify-hash':

ok = verify\_sha256\_hash(data, args.sha)

print("OK" if ok else "FAIL")

sys.exit(0 if ok else 1)

elif args.cmd == 'hmac':

key = args.key.encode()

print(hmac\_sha256(key, data))

elif args.cmd == 'verify-hmac':

key = args.key.encode()

ok = verify\_hmac\_sha256(key, data, args.tag)

print("OK" if ok else "FAIL")

sys.exit(0 if ok else 1)

if \_\_name\_\_ == "\_\_main\_\_":

main()

## Выполнение программы

### Вычисление SHA-256 строки

python3 lab\_hash\_mac.py hash -m "Hello, world!"

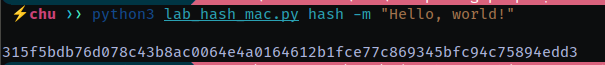


Рисунок 1 - Вычисление SHA-256 строки

### Проверка SHA-256

python3 lab\_hash\_mac.py verify-hash -m "Hello, world!" -s 315f5bdb76d078c43b8ac0064e4a0164612b1fce77c869345bfc94c75894edd3



Рисунок 2 - Проверка SHA-256

### Создание HMAC-SHA256

python3 lab\_hash\_mac.py hmac -k mysecretkey -m "Hello, world!"



Рисунок 3 - Создание HMAC-SHA256

### Проверка **HMAC-SHA256**

python3 lab\_hash\_mac.py verify-hmac -k mysecretkey -m "Hello, world!" -t 9348e20d01015b7c5881cfdd87473e441429e6d716ba0e2b11951e5f7e40c31d



Рисунок 4 - Проверка HMAC-SHA256

# Заключение

В ходе лабораторной работы была реализована утилита для вычисления и проверки криптографических хеш­функций на основе SHA-256, а также для генерации и верификации HMAC-SHA256. В частности:

1. Разработан модуль вычисления SHA-256, обеспечивающий корректное получение и сравнение хеш­значений для произвольных данных (строк или файлов).
2. Реализован алгоритм HMAC-SHA256 с демонстрацией механизма внутренней (ipad) и внешней (opad) подкладки, что полностью соответствует спецификации HMAC и позволяет надёжно аутентифицировать сообщение.
3. Добавлены команды для удобного использования через интерфейс командной строки: режимы hash, verify-hash, hmac, verify-hmac, поддерживающие как ввод через параметр -m/--message, так и посредством указания файла -f/--file.

Проведённые тесты подтвердили корректность работы всех функций: вычисленные хеш­значения совпадают с результатами стандартных инструментов, проверка корректно выявляет поддельные данные или теги.

Данная работа углубила понимание принципов построения криптографических хешей и механизмов аутентификации сообщений. Реализованный код может быть использован в качестве основы для встроенных систем контроля целостности и аутентификации данных в более крупных приложениях.