**Федеральное государственное бюджетное образовательное учреждение высшего образования "Белгородский государственный технологический университет им. В.Г. Шухова"**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем.

# Лабораторная работа №2

Шифрование с имитовставкой.

Выполнил:

Студент группы КБ-211

Коренев Д.Н.

Принял:

Смакаев А.В.

Белгород 2024

**Оглавление**

Задание ..... 3
Требования к консольному приложению ..... 4
Разработанная программа..... 5
Код программы..... 6
Вывод..... 9

*Цель работы:* ознакомится с принципами работы шифрования с имитовставкой. Разработать консольное приложение, использующее алгоритм HMAC для генерации имитовставки и осуществляющее шифрование и дешифрование файла с подтверждением целостности данных.

**Задание**

Разработать консольное приложение, реализующее шифрование/расшифрование файла с имитовставкой. В работе использовать:

● симметричный алгоритм шифрования AES-256 в режиме CFB;

● алгоритм генерации имитовставки HMAC;

● хэш-функцию SHA-256.

Алгоритм выполнения шифрования с имитовставкой:

1. На основе хеша полученных от пользователя данных вырабатывается ключ:

a. с помощью случайных данных генерируется “соль”;

b. формируется ключ

i. вариант 1:

1. “соль” конкатенируется с парольной фразой пользователя;

2. берется хэш от полученных данных: SHA256(salt + password).

ii. вариант 2: PBKDF2

2. Случайным образом вырабатывается инициализирующий вектор.

3. Одновременно с шифрованием файла вырабатывается имитовставка по алгоритму HMAC от открытого текста.

4. Все необходимые для расшифрования данные записываются в файл подряд без разделителей в бинарном виде. Формат файла следующий: [соль(32Б)][HMAC(32Б)][инициализирующий вектор(16Б)][шифротекст]

5. Производится расшифрование данных из файла, выводится результат проверки HMAC.

Для генерации “соли” можно использовать свой метод или один из следующих:

● накопление последних бит кодов нажимаемых пользователем кнопок;

● накопление последних бит от координат пикселя, находящегося под курсором пользователя;

● накопление последних бит от значений одного или нескольких цветовых каналов пикселя, находящегося под курсором.

**Требования к консольному приложению**

Консольное приложение должно иметь два режима:

● шифрование файла;

● расшифрование файла с последующим подтверждением целостности данных.

Консольное приложение должно принимать на вход следующие аргументы:

● имя файла для шифрования/расшифрования;

● имя файла для сохранения зашифрованного/расшифрованного файла;

● (опционально) отключение проверки целостности, по умолчанию проверка целостности включена.

Приветствуется использование аргументов командной строки для передачи вышеописанных параметров. Наличие интерактивного режима не обязательно – при его отсутствии или при запуске с неверными аргументами приложение должно показывать инструкцию по использованию.

После успешного запуска в любом из режимов приложение должно сгенерировать ключ шифрования. Для этого необходимы следующие компоненты:

● парольная фраза. Пользователь должен ввести парольную фразу в консоль, при этом необходимо использовать такие методы, как console.readPassword в JAVA, или специальные модули, например read в NodeJS;

● модификатор ключа. Пользователь должен предоставить программе случайные данные для генерации “соли” при шифровании файла. На этом этапе должен происходить сбор данных (пользователь перемещает курсор по экрану, нажимает случайные клавиши на клавиатуре и т.д.) до тех пор, пока “соль” не будет сгенерирована или пользователь не остановит выполнение программы.

При расшифровании соль берется из предоставленного для расшифровки файла.

После того, как все необходимые данные для выполнения операции получены, необходимо зашифровать/расшифровать файл и в случае расшифрования произвести проверку целостности следующим образом:

● сгенерировать на основе имеющихся данных (расшифрованного текста, парольной фразы и модификатора ключа) имитовставку;

● сравнить с имитовставкой, предоставленной в файле.

В случае подтверждения факта искажения данных сохранять расшифрованный файл не обязательно.

**Разработанная программа**

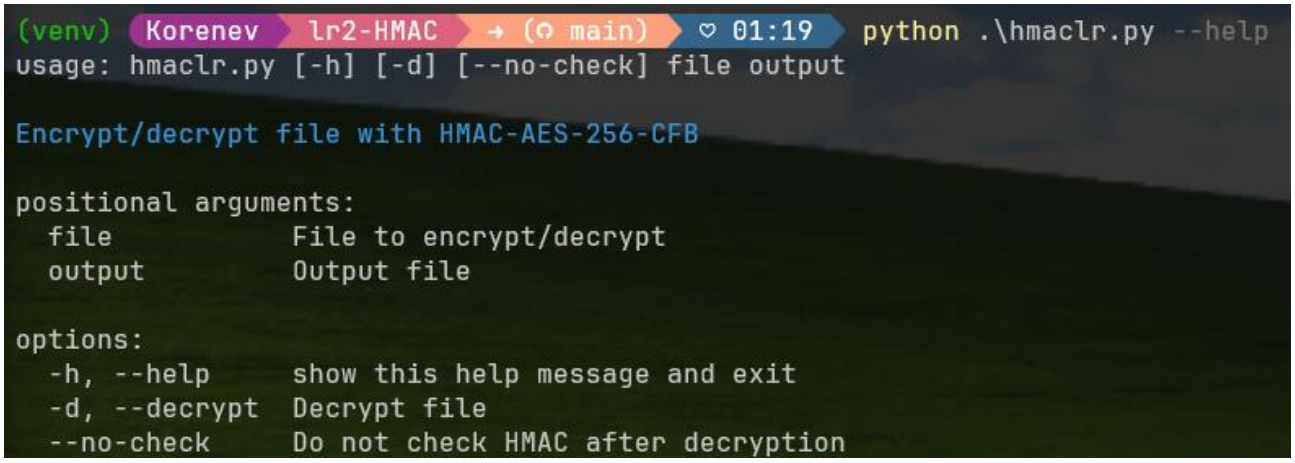


Рисунок 1. Помощь программы.

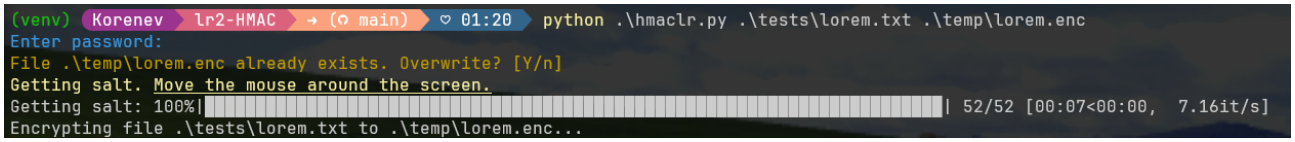


Рисунок 2. Шифрование файла.

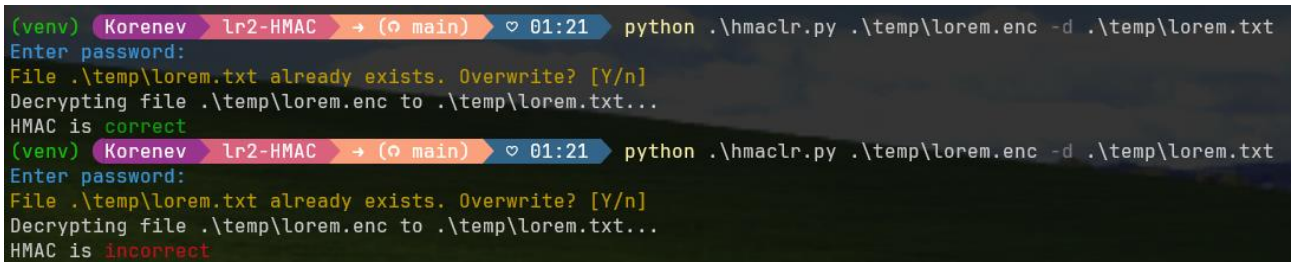


Рисунок 3. Расшифровка с указанием верного (сверху) и неверного (снизу) пароля.

**Код программы**

Полный код программы доступен ниже и в репозитории по ссылке: https://github.com/Kseen715/crypto-io-lr/tree/main/lr2-HMAC

```Python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# Python 3.12.6

import hmac
import os
import argparse
from hashlib import sha256
import getpass

from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
import tqdm
import pyautogui
from PIL import ImageGrab
import colorama as clr

def get_mouse_pixel_data():
    # Get the current mouse position
    x, y = pyautogui.position()

    # Capture the screen at the mouse position
    screen = ImageGrab.grab(bbox=(x, y, x+1, y+1))

    # Get the RGB value of the pixel
    pixel = screen.load()
    r, g, b = pixel[0, 0]

    return x, y, r, g, b

def get_salt(byte_len) -> int:
    salt = 0
    bitlen = byte_len * 8
    prev_x, prev_y, prev_r, prev_g, prev_b = None, None, None, None, None
    print(f'{clr.Fore.LIGHTYELLOW_EX}Getting salt. {
        '\033[4m'}Move the mouse around the screen.{clr.Style.RESET_ALL}')
    # for _ in range((bitlen // 5) + 1):
    for _ in tqdm.tqdm(range((bitlen // 5) + 1), desc='Getting salt'):
        while True:
            x, y, r, g, b = get_mouse_pixel_data()
            if (x, y, r, g, b) != (prev_x, prev_y, prev_r, prev_g, prev_b):
                prev_x, prev_y, prev_r, prev_g, prev_b = x, y, r, g, b
                salt = (salt << 1) | (x & 0x01)
                salt = (salt << 1) | (y & 0x01)
                salt = (salt << 1) | (r & 0x01)
                salt = (salt << 1) | (g & 0x01)
                salt = (salt << 1) | (b & 0x01)
                # print last 5 bits in binary, with leading zeros
                # print(f'{salt & 0x1F:05b}', end='\r')
                bitlen -= 5
                break
    # cut the excessive bits
    salt = salt >> (bitlen * -1)
    return salt

def count_zeros_ones_binary(num, bytelength=32):
    zeros = 0
```

```
    ones = 0
    while num:
        if num & 1:
            ones += 1
        else:
            pass
        num >>= 1
    bitlen = bytelength * 8
    zeros = bitlen - ones
    return zeros, ones

def get_password_salted_hash(salt, password):
    # invert bits in password
    # repr every char in password as bytes, then invert bits in every byte
    password = int.from_bytes(
        bytes([(~char + 256) % 256
               for char in password.encode('utf-8')]), 'big')
    # if password longer than 32 bytes, fold it recursively
    # to 32 bytes using XOR
    while password.bit_length() > 256:
        password = (password >> 256) ^ (
            password & 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF)

    # add salt to password and hash
    salted_password = salt.to_bytes(32, 'big') + password.to_bytes(32, 'big')
    return sha256(salted_password).digest()

def get_iv(byte_len):
    return get_random_bytes(byte_len)

def get_hmac(key, data):
    return hmac.new(key, data, sha256).digest()

def cypher_file(file_path, passw, output_path=None):
    # [salt(32B)][HMAC(32B)][IV(16B)][cipher_text]
    assert os.path.exists(file_path), f'File {file_path} does not exist'
    assert os.path.isfile(file_path), f'{file_path} is not a file'
    assert output_path is not None, 'Output path is not provided'
    if os.path.exists(output_path):
        response = input(f'{clr.Fore.YELLOW}File {
                         output_path} already exists. Overwrite? [Y/n]{
                             clr.Style.RESET_ALL}')
        if response.lower() != 'y' and response.lower() != '':
            return
    if not os.path.exists(os.path.dirname(output_path)):
        os.makedirs(os.path.dirname(output_path))
    salt = get_salt(32)
    iv = get_iv(16)
    key = get_password_salted_hash(salt, passw)
    hmac = None
    with open(file_path, 'rb') as file:
        hmac = get_hmac(key, file.read())
    # print(hmac)
    cipher = AES.new(key, AES.MODE_CFB, iv)
    print(f'Encrypting file {file_path} to {output_path}...')
    with open(file_path, 'rb') as file:
        with open(output_path, 'wb') as enc_file:
            enc_file.write(salt.to_bytes(32, 'big'))
            enc_file.write(hmac)
            enc_file.write(iv)
            while chunk := file.read(16 * 1024):
                enc_file.write(cipher.encrypt(chunk))

def decypher_file(file_path, passw, output_path=None, check=True):
    # [salt(32B)][HMAC(32B)][IV(16B)][cipher_text]
```

```
    assert os.path.exists(file_path), f'File {file_path} does not exist'
    assert os.path.isfile(file_path), f'{file_path} is not a file'
    assert output_path is not None, 'Output path is not provided'
    if os.path.exists(output_path):
        response = input(f'{clr.Fore.YELLOW}File {
                         output_path} already exists. Overwrite? [Y/n]{
                             clr.Style.RESET_ALL}')
        if response.lower() != 'y' and response.lower() != '':
            return
    if not os.path.exists(os.path.dirname(output_path)):
        os.makedirs(os.path.dirname(output_path))
    print(f'Decrypting file {file_path} to {output_path}...')
    checkres = None
    with open(file_path, 'rb') as file:
        salt = int.from_bytes(file.read(32), 'big')
        hmac = file.read(32)
        iv = file.read(16)
        key = get_password_salted_hash(salt, passw)
        cipher = AES.new(key, AES.MODE_CFB, iv)
        decrypted_data = b''
        while chunk := file.read(16 * 1024):
            decrypted_data += cipher.decrypt(chunk)
        checkres = get_hmac(key, decrypted_data)
        if hmac == checkres:
            with open(output_path, 'wb') as dec_file:
                dec_file.write(decrypted_data)
    if check:
        if hmac == checkres:
            print(f'HMAC is {clr.Fore.GREEN}correct{clr.Style.RESET_ALL}')
        else:
            print(f'HMAC is {clr.Fore.RED}incorrect{clr.Style.RESET_ALL}')

def main():
    parser = argparse.ArgumentParser(
        description=f'{clr.Fore.CYAN
                       }Encrypt/decrypt file with HMAC-AES-256-CFB{
                           clr.Style.RESET_ALL}')
    parser.add_argument('file', type=str, help='File to encrypt/decrypt')
    parser.add_argument('output', type=str, help='Output file')
    parser.add_argument('-d', '--decrypt',
                        action='store_true', help='Decrypt file')
    # do not check HMAC option
    parser.add_argument('--no-check', dest='check', action='store_false',
                        help='Do not check HMAC after decryption')

    args = parser.parse_args()

    # show help if no arguments provided
    if not vars(args) or not args.file or not args.output:
        parser.print_help()
        parser.exit()

    password = getpass.getpass(prompt=f'{clr.Fore.CYAN}Enter password: {
                               clr.Style.RESET_ALL}')
    if args.decrypt:
        decypher_file(args.file, password, args.output, args.check)
    else:
        cypher_file(args.file, password, args.output)

if __name__ == "__main__":
    main()
```

**Вывод**

В ходе выполнения лабораторной работы мы ознакомились с принципами работы шифрования с использованием имитовставки. Нам удалось разработать консольное приложение, которое реализует шифрование и расшифрование файлов с использованием алгоритма AES-256 в режиме CFB и HMAC для генерации имитовставки, обеспечивающей контроль целостности данных. Для генерации ключа шифрования применялись различные методы, включая использование случайных данных для создания "соли" и парольных фраз. Одновременно с процессом шифрования происходила генерация имитовставки на основе открытого текста с использованием HMAC и хэш-функции SHA-256. В ходе выполнения задачи мы успешно записали все необходимые для расшифровки данные в единый файл, соблюдая требуемый формат. В процессе расшифрования были проведены проверки целостности данных, что подтверждало корректность их обработки. Таким образом, работа достигла своей цели: разработанное приложение успешно выполнило функции шифрования, расшифрования и контроля целостности данных, демонстрируя корректность и надежность алгоритмов.