МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

КРИПТОГРАФИЯ

ОТЧЕТ О ПРАКТИКЕ

студента 4 курса 411 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Вильцева Данила Денисовича

Проверил
Преподаватель _____ А. А. Лобов

Саратов 2023

СОДЕРЖАНИЕ

1 Хеш-функции

1.1 Условие задачи

— a. С помощью OpenSSL посчитайте хеш файла с помощью 3-х алгоритмов

— b. Измените один бит в файле и снова посчитайте хеш-значения теми же алгоритмами. Чтобы изменить бит не обязательно писать программу, можно взять шестнадцатеричный редактор и поменять в нём значения. Взять чётную шестнадцатеричную цифру и увеличить её на один. Или взять текстовый редактор и изменить один символ так, чтобы новый код отличался от предыдущего на один. Лучше всего менять ASCII символы, коды которых можете посмотреть в таблице ASCII.

— c. Сильно ли изменились хеш-значения (можно посмотреть количество отличающихся бит, или количество отличающихся символов).

1.2 Выполнение задачи

```
Last login: Wed Sep 20 15:00:54 on ttys000
(base) danila@MacBook-Pro-Danila ~ % openssl dgst -md5 /Users/danila/IdeaProjects/steganography/src/main/scala/Steganography/test.txt
openssl dgst -sha256 /Users/danila/IdeaProjects/steganography/src/main/scala/Steganography/test.txt
openssl dgst -sha512 /Users/danila/IdeaProjects/steganography/src/main/scala/Steganography/test.txt

openssl dgst -md5 /Users/danila/IdeaProjects/steganography/src/main/scala/Steganography/test2.txt
openssl dgst -sha256 /Users/danila/IdeaProjects/steganography/src/main/scala/Steganography/test2.txt
openssl dgst -sha512 /Users/danila/IdeaProjects/steganography/src/main/scala/Steganography/test2.txt
MD5(/Users/danila/IdeaProjects/steganography/src/main/scala/Steganography/test.txt)= 3adbbad1791fbae3ec908894c4963870
SHA256(/Users/danila/IdeaProjects/steganography/src/main/scala/Steganography/test.txt)= 68e656b251e67e8358bef8483ab0d51c6619f3e7a1a9f0e75838d41ff368f728
SHA512(/Users/danila/IdeaProjects/steganography/src/main/scala/Steganography/test.txt)= 6c2618358da07c830b88c5af8c3535080e8e603c88b891028a259ccdb9ac802d0fc0170c99d58affcf00786ce188fc5d753e8c6628af2071c3270d50445c4b1c
MD5(/Users/danila/IdeaProjects/steganography/src/main/scala/Steganography/test2.txt)= 7ade70847b138e86437ff4ec60af5aa1
SHA256(/Users/danila/IdeaProjects/steganography/src/main/scala/Steganography/test2.txt)= b1f033b77cfdeda3a3ad65176312cf4226cee0255ef606c5878a58b7bb64c8bb
SHA512(/Users/danila/IdeaProjects/steganography/src/main/scala/Steganography/test2.txt)= de0589ebe3e9858150765b1e3c37f2b88168a4ed4113fc66d7bb966d4c7de3f895b78576944c13d9e29caa4618af92973af0ddff7d3694dfd24298db879bf988
```

Рисунок 1 – Пункты a, b

```
MD5 Algorithm
hash1:  3adbbad1791fbae3ec908894c4963870
hash2:  7ade70847b138e86437ff4ec60af5aa1
Количество отличающихся бит: 57
SHA256 Algorithm
hash1:  68e656b251e67e8358bef8483ab0d51c6619f3e7a1a9f0e75838d41ff368f728
hash2:  b1f033b77cfdeda3a3ad65176312cf4226cee0255ef606c5878a58b7bb64c8bb
Количество отличающихся бит: 129
SHA512 Algorithm
hash1:  6c2618358da07c830b88c5af8c3535080e8e603c88b891028a259ccdb9ac802d0fc0170c99d58affcf00786ce188fc5d753e8c6628af2071c3270d50445c4b1c
hash2:  de0589ebe3e9858150765b1e3c37f2b88168a4ed4113fc66d7bb966d4c7de3f895b78576944c13d9e29caa4618af92973af0ddff7d3694dfd24298db879bf988
Количество отличающихся бит: 261
```

Рисунок 2 – Пункт c

Программа для сравнения хэш-значений:

```python
def count_different_bits(hash1, hash2):
    # Проверяем, что хеш-значения имеют одинаковую длину
    if len(hash1) != len(hash2):
        raise ValueError("Хеш-значения имеют разную длину")

    # Преобразуем хеш-значения в двоичные строки
    binary_hash1 = bin(int(hash1, 16))[2:]
    binary_hash2 = bin(int(hash2, 16))[2:]
```

```
9
10      # Дополняем строки нулями слева, чтобы они имели одинаковую длину
11      max_len = max(len(binary_hash1), len(binary_hash2))
12      binary_hash1 = binary_hash1.zfill(max_len)
13      binary_hash2 = binary_hash2.zfill(max_len)
14
15      # Считаем количество отличающихся бит
16      different_bits = sum(1 for bit1, bit2 in zip(binary_hash1, binary_hash2) if bit1 != bit2)
17
18      return different_bits
19
20
21  print("MD5 Algorithm")
22  hash1 = "3adbbad1791fbae3ec908894c4963870"
23  hash2 = "7ade70847b138e86437ff4ec60af5aa1"
24  print("hash1: ", hash1)
25  print("hash2: ", hash2)
26  result1 = count_different_bits(hash1, hash2)
27  print("Количество отличающихся бит:", result1)
28
29  print("SHA256 Algorithm")
30  hash1_2 = "68e656b251e67e8358bef8483ab0d51c6619f3e7a1a9f0e75838d41ff368f728"
31  hash2_2 = "b1f033b77cfdeda3a3ad65176312cf4226cee0255ef606c5878a58b7bb64c8bb"
32  print("hash1: ", hash1_2)
33  print("hash2: ", hash2_2)
34  result2 = count_different_bits(hash1_2, hash2_2)
35  print("Количество отличающихся бит:", result2)
36
37  print("SHA512 Algorithm")
38  hash1_3 = "6c2618358da07c830b88c5af8c3535080e8e603c88b891028a259ccdb9ac802d0fc0170c99d58affcf00786ce188fc5d753e8c6628af2071c
39  hash2_3 = "de0589ebe3e9858150765b1e3c37f2b88168a4ed4113fc66d7bb966d4c7de3f895b78576944c13d9e29caa4618af92973af0ddff7d3694dfd
40  print("hash1: ", hash1_3)
41  print("hash2: ", hash2_3)
42  result3 = count_different_bits(hash1_3, hash2_3)
43  print("Количество отличающихся бит:", result3)
```

2 Симметричное шифрование

2.1 Условие задачи

— a. Выберите один файлик размером в 1 килобайт или более:

— b. Зашифруйте его в режиме простой замены (ECB) и в режиме сцепления блоков (CBC). Расшифруйте его. Сравните хеши файла до зашифрования и после расшифрования.

— c. Измените в выбранном файлике первый байт и повторите шифрование с тем же самым ключом. Сравните количество различающихся байтов в файлах, зашифрованных в одинаковых режимах (много или мало, если мало, то сколько различных) (сравниваются 1-ый байт 1-го файла с 1-ым байтом 2-го файла, 2-ой байт 1-го файла с 2-ым байтом 2-го файла и т.д.).

2.2 Выполнение задачи

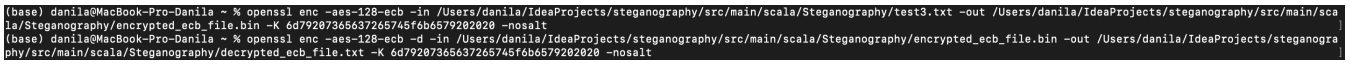

Рисунок 3 – Зашифровка и расшифровка в режиме ECB

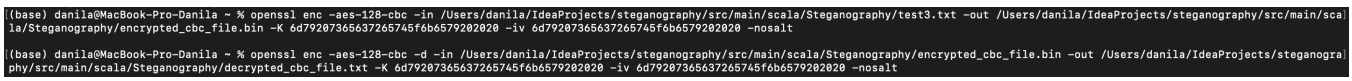

Рисунок 4 – Зашифровка и расшифровка в режиме CBC

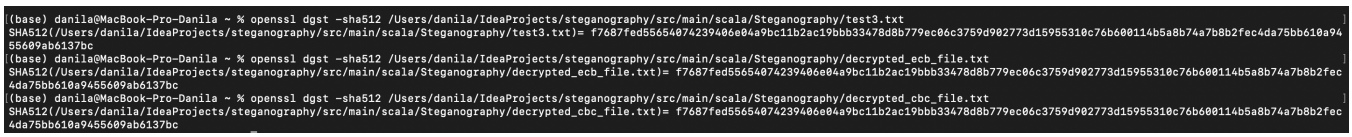

Рисунок 5 – Сравнение хэшей файла до зашифрования и после расшифрования

Далее я изменил в файле первый байт и повторил шифрование:

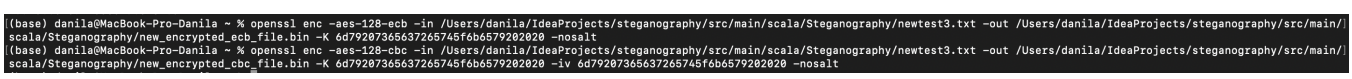

Рисунок 6 – Шифрование с измененном байтом

Рисунок 7 – Сравнение количества различающихся байтов в файлах

Программа для сравнения количества различающихся байтов:

```python
def compare_files(file1_path, file2_path):
    with open(file1_path, "rb") as file1, open(file2_path, "rb") as file2:
        data1 = file1.read()
        data2 = file2.read()

    # Сравниваем данные и определяем количество различающихся байтов
    differences = 0
    min_length = min(len(data1), len(data2))
    for i in range(min_length):
        if data1[i] != data2[i]:
            differences += 1

    return differences

original_file_path_ecb = "/Users/danila/IdeaProjects/steganography/src/main/scala/Steganography/encrypted_ecb_file.bin"
modified_file_path_ecb = "/Users/danila/IdeaProjects/steganography/src/main/scala/Steganography/new_encrypted_ecb_file.bin"

original_file_path_cbc = "/Users/danila/IdeaProjects/steganography/src/main/scala/Steganography/encrypted_cbc_file.bin"
modified_file_path_cbc = "/Users/danila/IdeaProjects/steganography/src/main/scala/Steganography/new_encrypted_cbc_file.bin"

# Вычисляем количество различающихся байтов
print("ECB")
num_differences = compare_files(original_file_path_ecb, modified_file_path_ecb)
print(f"Количество различающихся байтов: {num_differences}")

print("CBC")
num_differences2 = compare_files(original_file_path_cbc, modified_file_path_cbc)
print(f"Количество различающихся байтов: {num_differences2}")
```

3 Ассиметричное шифрование

3.1 Условие задачи

— a. Сгенерируйте ключи RSA
— b. Зашифруйте файл шифром RSA с помощью открытого ключа.
— c. Расшифруйте файл шифром RSA с помощью закрытого ключа.
— d. Сравните хеши оригинального файла и расшифрованного.

3.2 Выполнение задачи

```
Last login: Wed Sep 20 16:36:51 on ttys000
(base) danila@MacBook-Pro-Danila ~ % openssl genpkey -algorithm RSA -out private_key.pem
openssl rsa -pubout -in private_key.pem -out public_key.pem

............+++++
..............+++++
writing RSA key
(base) danila@MacBook-Pro-Danila ~ % openssl rsautl -encrypt -pubin -inkey public_key.pem -in /Users/danila/Desktop/studying/7\ sem/InfoSec/rsakeys/file_to_encrypt.txt -out /Users/danila/Desktop/studying/
7\ sem/InfoSec/rsakeys/encrypted_file.enc

(base) danila@MacBook-Pro-Danila ~ % openssl rsautl -decrypt -inkey private_key.pem -in /Users/danila/Desktop/studying/7\ sem/InfoSec/rsakeys/encrypted_file.enc -out /Users/danila/Desktop/studying/7\ sem/
InfoSec/rsakeys/decrypted_file.txt
```

Рисунок 8 – Пункты a,b,c

```
(base) danila@MacBook-Pro-Danila ~ % openssl dgst -sha256 /Users/danila/Desktop/studying/7\ sem/InfoSec/rsakeys/file_to_encrypt.txt
openssl dgst -sha256 /Users/danila/Desktop/studying/7\ sem/InfoSec/rsakeys/decrypted_file.txt

SHA256(/Users/danila/Desktop/studying/7 sem/InfoSec/rsakeys/file_to_encrypt.txt)= 09ca7e4eaa6e8ae9c7d26116712918488 3644d07dfba7cbfbc4c8a2e08360d5b
SHA256(/Users/danila/Desktop/studying/7 sem/InfoSec/rsakeys/decrypted_file.txt)= 09ca7e4eaa6e8ae9c7d26116712918488 3644d07dfba7cbfbc4c8a2e08360d5b
(base) danila@MacBook-Pro-Danila ~ %
```

Рисунок 9 – Сравнение хэшей оригинального файла и расшифрованного

4 Электронная цифровая подпись (ЭЦП)

4.1 Условие задачи

Базовая схема генерации подписи сообщения составляется так: считается хеш-значение сообщения (файла) и оно зашифровывается с закрытым ключом. Проверка подписи заключается в расшифровании полученной подписи с помощью открытого ключа и сравнении с хеш-значением сообщения (файла). Если они равны, то подпись принимается.

- a. Выбрать файл.
- b. Подписать его с помощью закрытого ключа RSA.
- c. Проверить подпись файла с помощью открытого ключа RSA.
- d. Изменить в файле один бит (HEX-редактор в помощь)
- e. Проверить подпись файла с помощью открытого ключа RSA.
- f. Сгенерировать параметры DSA.
- g. Сгенерировать закрытый ключ DSA.
- h. Сгенерировать открытый ключ DSA.
- i. Подписать выбранный файл закрытым ключом DSA.
- j. Проверить подпись открытым ключом DSA.
- k. Изменить один бит в файле.
- l. Проверить изменённую подпись открытым ключом DSA.

4.2 Выполнение задачи

```
Last login: Wed Sep 20 16:46:39 on ttys000
(base) danila@MacBook-Pro-Danila ~ % openssl dgst -sha256 -sign private_key.pem -out /Users/danila/Desktop/studying/7\ sem/InfoSec/esign/signature.bin /Users/danila/Desktop/studying/7\ sem/InfoSec/esign/f
ile.txt
```

Рисунок 10 – Пункт b

```
(base) danila@MacBook-Pro-Danila ~ % openssl dgst -sha256 -verify public_key.pem -signature /Users/danila/Desktop/studying/7\ sem/InfoSec/esign/signature.bin /Users/danila/Desktop/studying/7\ sem/InfoSec/
esign/file.txt

Verified OK
```

Рисунок 11 – Пункт c

Далее изменил в файле один бит (пункт d).

Рисунок 12 – Пункт e

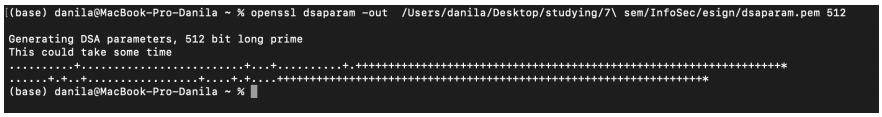

Рисунок 13 – Пункт f

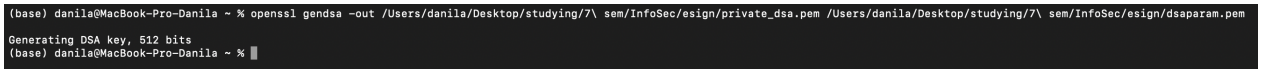

Рисунок 14 – Пункт g

Рисунок 15 – Пункт h

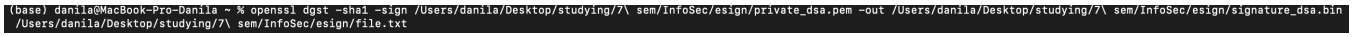

Рисунок 16 – Пункт i

Рисунок 17 – Пункт j

Далее снова изменил один бит в файле (пункт k).

Рисунок 18 – Пункт l