

**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)

Санкт-Петербург

2025 г.

СОДЕРЖАНИЕ

Содержание.....	2
Введение.....	3
Ход работы.....	4
1. Программа.....	4
2. Выполнение программы.....	6
2.1. Вычисление SHA-256 строки.....	6
2.2. Проверка SHA-256.....	6
2.3. Создание HMAC-SHA256.....	6
2.4. Проверка HMAC-SHA256.....	7
Заключение.....	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

1. 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)

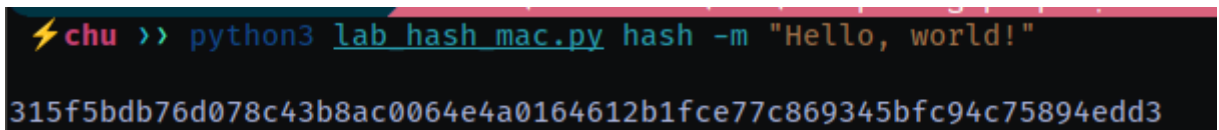
if __name__ == "__main__":
    main()

```

2. Выполнение программы

2.1. Вычисление SHA-256 строки

```
python3 lab_hash_mac.py hash -m "Hello, world!"
```



```

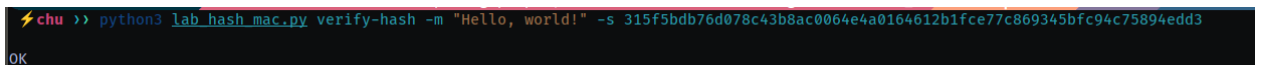
chu >> python3 lab_hash_mac.py hash -m "Hello, world!"
315f5bdb76d078c43b8ac0064e4a0164612b1fce77c869345bfc94c75894edd3

```

Рисунок 1 - Вычисление SHA-256 строки

2.2. Проверка SHA-256

```
python3 lab_hash_mac.py verify-hash -m "Hello, world!" -s 315f5bdb76d078c43b8ac0064e4a0164612b1fce77c869345bfc94c75894edd3
```



```

chu >> python3 lab_hash_mac.py verify-hash -m "Hello, world!" -s 315f5bdb76d078c43b8ac0064e4a0164612b1fce77c869345bfc94c75894edd3
OK

```

Рисунок 2 - Проверка SHA-256

2.3. Создание HMAC-SHA256

```
python3 lab_hash_mac.py hmac -k mysecretkey -m "Hello, world!"
```

```
chu >> python3 lab_hash_mac.py hmac -k mysecretkey -m "Hello, world!"  
9348e20d01015b7c5881cfdd87473e441429e6d716ba0e2b11951e5f7e40c31d
```

Рисунок 3 - Создание HMAC-SHA256

2.4. Проверка HMAC-SHA256

```
python3 lab_hash_mac.py verify-hmac -k mysecretkey -m "Hello, world!" -t  
9348e20d01015b7c5881cfdd87473e441429e6d716ba0e2b11951e5f7e40c31d
```

```
chu >> python3 lab_hash_mac.py verify-hmac -k mysecretkey -m "Hello, world!" -t 9348e20d01015b7c5881cfdd87473e441429e6d716ba0e2b11951e5f7e40c31d  
OK
```

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.