Белорусский государственный университет информатики и радиоэлектроники

Кафедра информатики

Лабораторная работа № 5

Хэш-функции

Выполнила студент гр. 653502: Серебренников В. А.

Проверил ассистент КИ: Артемьев В. С.

Минск, 2019

Введение

HMAC (сокращение от англ. hash-based message authentication code, код аутентификации (проверки подлинности) сообщений, использующий хеш-функции) — в информатике (криптографии), один из механизмов проверки целостности информации, позволяющий гарантировать то, что данные, передаваемые или хранящиеся в ненадёжной среде, не были изменены посторонними лицами.

Преимущества HMAC:

- возможность использования хэш-функций, уже имеющихся в программном продукте;
- отсутствие необходимости внесения изменений в реализации существующих хэш-функции (внесение изменений может привести к ухудшению производительности и криптостойкости);
- возможность замены хэш-функции в случае появления более безопасной или более быстрой хэш-функции.

В рамках лабораторной работы необходимо реализовать программные средства контроля целостности сообщений с помощью вычисления хэш функций и алгоритма HMAC.

Блок-схема алгоритма

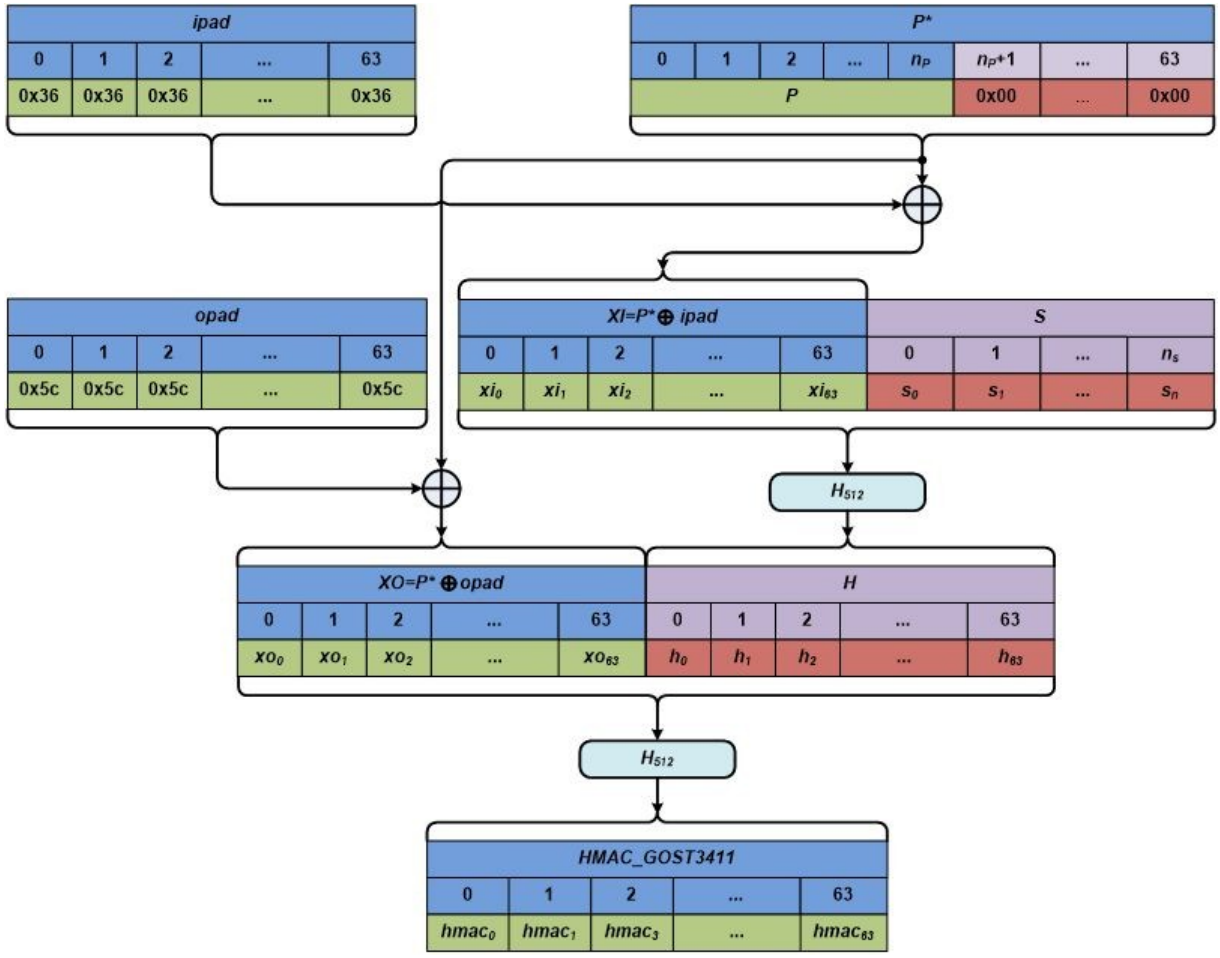

Рис.1. Схема алгоритма

Пример работы программы

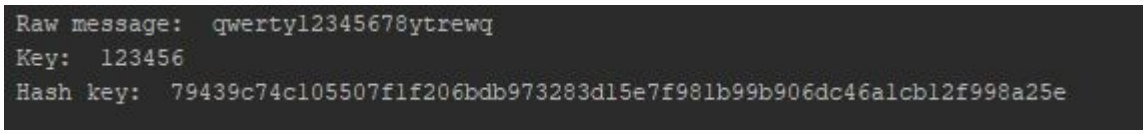

Рис.2. Пример работы

Код программы

```
K = [
    0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1,
0x923f82a4, 0xab1c5ed5,
    0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe,
0x9bdc06a7, 0xc19bf174,
    0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa,
0x5cb0a9dc, 0x76f988da,
    0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147,
0x06ca6351, 0x14292967,
    0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb,
0x81c2c92e, 0x92722c85,
    0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624,
0xf40e3585, 0x106aa070,
    0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a,
0x5b9cca4f, 0x682e6ff3,
    0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb,
0xbef9a3f7, 0xc67178f2
]


def rotate_right(num, shift, size=32):
    return (num >> shift) | (num << size - shift)


def sha256(text):
    text = bytearray(text)
    length = len(text) * 8
    text.append(0x80)

    while (len(text) * 8 + 64) % 512 != 0:
        text.append(0x00)

    text += length.to_bytes(8, 'big')

    blocks = []
    for i in range(0, len(text), 64):
        blocks.append(text[i:i + 64])

    h0 = 0x6a09e667
    h1 = 0xbb67ae85
    h2 = 0x3c6ef372
    h3 = 0xa54ff53a
    h5 = 0x9b05688c
    h4 = 0x510e527f
    h6 = 0x1f83d9ab
    h7 = 0x5be0cd19

    for message_block in blocks:
        message_schedule = []
        for t in range(0, 64):
            if t <= 15:
                message_schedule.append(bytes(message_block[t * 4:(t * 4) + 4]))
            else:
```

```
                term1 = (rotate_right((int.from_bytes(message_schedule[t - 2],
'big')), 17)
                         ^ rotate_right((int.from_bytes(message_schedule[t - 2],
'big')), 19)
                         ^ ((int.from_bytes(message_schedule[t - 2], 'big')) >> 10))
                term2 = int.from_bytes(message_schedule[t - 7], 'big')
                term3 = (rotate_right((int.from_bytes(message_schedule[t - 15],
'big')), 7)
                         ^ rotate_right((int.from_bytes(message_schedule[t - 15],
'big')), 18)
                         ^ ((int.from_bytes(message_schedule[t - 15], 'big')) >> 3))
                term4 = int.from_bytes(message_schedule[t - 16], 'big')

                schedule = ((term1 + term2 + term3 + term4) % 2 ** 32).to_bytes(4,
'big')
                message_schedule.append(schedule)

        a = h0
        b = h1
        c = h2
        d = h3
        e = h4
        f = h5
        g = h6
        h = h7

        for t in range(64):
            t1 = ((h + (rotate_right(e, 6) ^ rotate_right(e, 11) ^ rotate_right(e,
25))
                   + ((e & f) ^ (~e & g)) + K[t] +
                   int.from_bytes(message_schedule[t], 'big')) % 2 ** 32)

            t2 = ((rotate_right(a, 2) ^ rotate_right(a, 13) ^ rotate_right(a, 22))
                  + ((a & b) ^ (a & c) ^ (b & c))) % 2 ** 32
            h = g
            g = f
            f = e
            e = (d + t1) % 2 ** 32
            d = c
            c = b
            b = a
            a = (t1 + t2) % 2 ** 32

        h0 = (h0 + a) % 2 ** 32
        h1 = (h1 + b) % 2 ** 32
        h2 = (h2 + c) % 2 ** 32
        h3 = (h3 + d) % 2 ** 32
        h4 = (h4 + e) % 2 ** 32
        h5 = (h5 + f) % 2 ** 32
        h6 = (h6 + g) % 2 ** 32
        h7 = (h7 + h) % 2 ** 32

    return ((h0).to_bytes(4, 'big') + (h1).to_bytes(4, 'big') +
            (h2).to_bytes(4, 'big') + (h3).to_bytes(4, 'big') +
            (h4).to_bytes(4, 'big') + (h5).to_bytes(4, 'big') +
            (h6).to_bytes(4, 'big') + (h7).to_bytes(4, 'big'))
```

```python
def hmac(key, text):
    i_key = bytearray()
    o_key = bytearray()

    key = key.encode()
    text = text.encode()
    blocksize = 64

    if len(key) > blocksize:
        key = bytearray(sha256(key))
    elif len(key) < blocksize:
        i = len(key)
        while i < blocksize:
            key += b"\x00"
            i += 1

    for i in range(blocksize):
        i_key.append(0x36 ^ key[i])
        o_key.append(0x5C ^ key[i])
    text = bytes(o_key) + sha256(bytes(i_key) + text)
    return sha256(text).hex()


if __name__ == "__main__":
    key = "123456"
    text = "qwerty12345678ytrewq"
    print("Raw message: ", text)
    print("Key: ", key)

    h_key = hmac(key, text)
    print("Hash key: ", h_key)
```

Вывод

Безопасность любой функции MAC на основе встроенных хеш-функций зависит от криптостойкости базовой хеш-функции. Привлекательность HMAC — в том, что его создатели смогли доказать точное соотношение между стойкостью встроенных хеш-функций и стойкостью HMAC.

В ходе написания лабораторной работы были изучен алгоритм хэширования HMAC, а также написана его программная реализация.