Белорусский государственный университет информатики и радиоэлектроники

Кафедра информатики

Лабораторная работа № 5

Хэш-функции

Выполнила студентка гр. 653502: Сулима М.Ф.

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

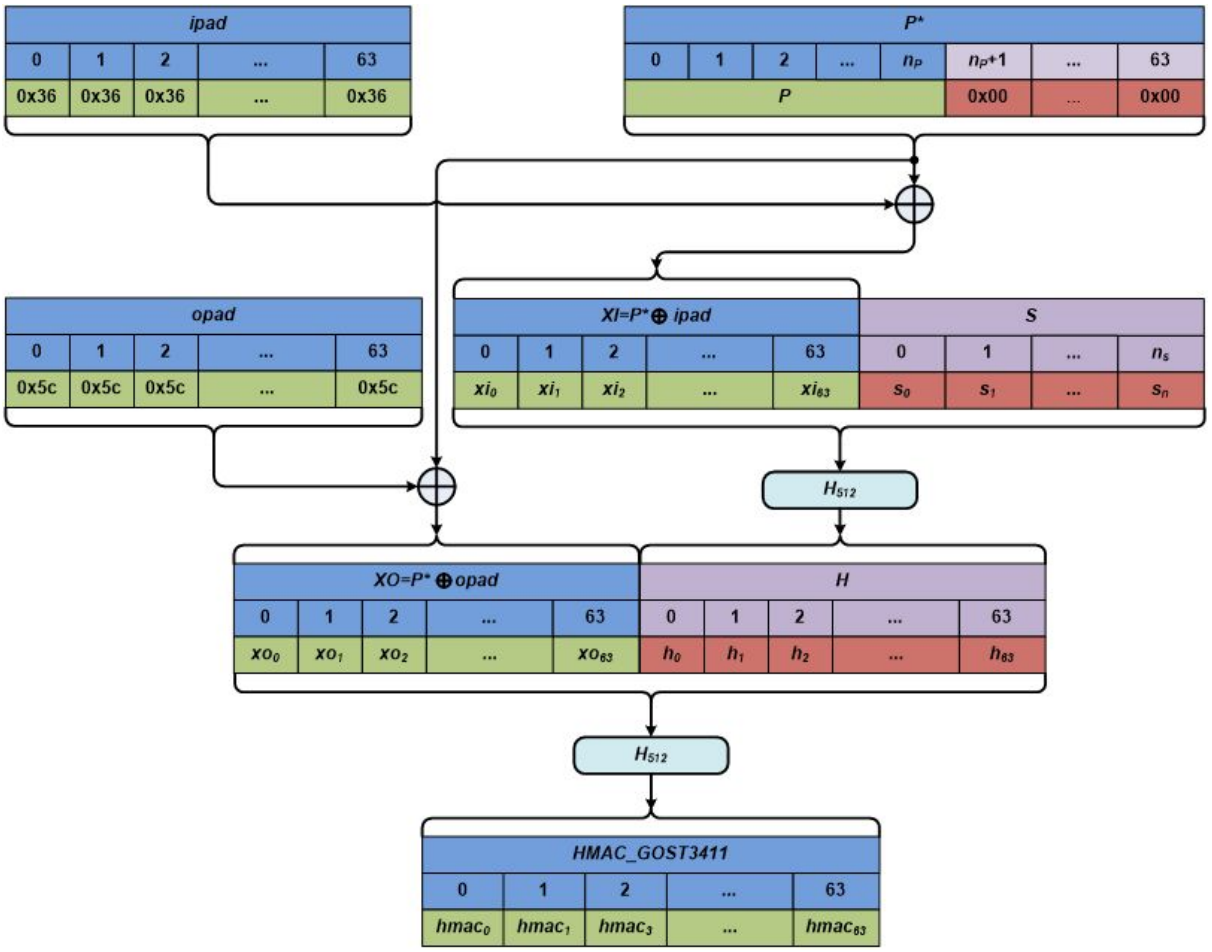

Рис.1. Схема алгоритма

Пример работы программы

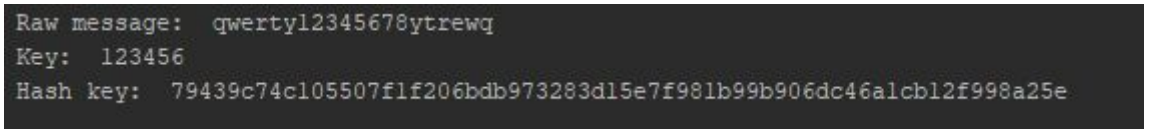

Рис.2. Пример работы

Код программы

```
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:
              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'))

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)
```

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.