Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №5

По дисциплине «Методы защиты информации»

По теме «Хэш-функции»

Выполнил:

Студент гр. 653501

Никитинская А. С.

Проверил:

Артемьев В. С.

Минск 2019

Содержание

[1. Постановка задачи 3](#_Toc26720055)

[2. Краткие теоретические сведения 4](#_Toc26720056)

[3. Блок-схема алгоритма 7](#_Toc26720058)

[Вывод 8](#_Toc26720059)

[Приложение 1. Исходный код программы 9](#_Toc26720060)

[Приложение 2. Скриншот работы программы 12](#_Toc26720061)

## 1. Постановка задачи

1) Изучить теоретические сведения.

2) Реализовать программное средство контроля целостности сообщений с помощью вычисления хэш-функции алгоритма НМАС.

## Краткие теоретические сведения

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.

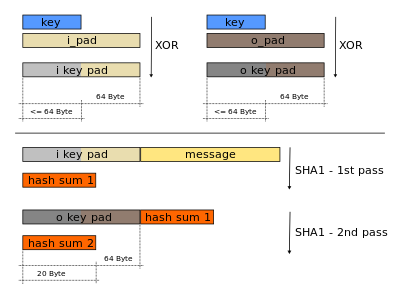


Рисунок 2.1. Генерация ключей в алгоритме HMAC

### Принцип работы алгоритма

* Размер ключа выравнивается с размером блока используемого алгоритма хэширования;
* Если ключ key длиннее блока, он укорачивается путем применения к нему используемого алгоритма хэширования hash:

key = hash(key);

Размер выходного значения алгоритма хэширования обычно много меньше размера блока хэшируемых данных – например, соответственно 128 и 512 [битов](http://cryptowiki.net/index.php?title=Битов) для алгоритма MD4; рекомендуется минимальный размер ключа, равный размеру выходного значения алгоритма хэширования;

* Если размер ключа меньше размера блока, то выровненный ключ k получается путем дополнения до размера блока нулевыми битами исходного (или укороченного) ключа key.
* Выровненный ключ k складывается по модулю 2 с константой C1, которая представляет собой блок данных, заполненный байтами с шестнадцатеричным значением 36; аналогичным образом ключ k также складывается с константой C2, которая представляет собой блок данных, заполненный байтами с шестнадцатеричным значением 5С:

ki = k ⊕ C1; ko = k ⊕ C2.

* Вычисляется хэш-значение от результата конкатенации модифицированного ключа ki и сообщения m:

t = hash(ki || m).

* Выходным значением алгоритма HMAC является хэш-значение от результата конкатенации модифицированного ключа ko и полученного на предыдущем шаге значения t:

hmac = hash(ko || t).

Таким образом, при вычислении HMAC используемый алгоритм хэширования применяется дважды; каждый раз с участием модифицированного ключа. Размер выходного значения алгоритма HMAC равен размеру выходного значения алгоритма хэширования, а общая формула вычисления HMAC (без учета выравнивания ключа) выглядит следующим образом:

HMAC(k, m) = hash((k ⊕ C2) || hash((k ⊕ C1) || m)).

Криптостойкость

[Криптостойкость](http://cryptowiki.net/index.php?title=Криптостойкость) HMAC зависит, прежде всего, от следующих факторов:

* криптостойкость используемого алгоритма хэширования;
* размер выходного значения алгоритма;
* размер и качество ключа.

Контексты атак на HMAC и «классическое» использование алгоритма хэширования заметно различаются. Атакующий алгоритм HMAC может преследовать следующие цели:

* подобрать сообщение, HMAC которого равен эталонному или равен HMAC эталонного сообщения;
* это похоже на атаки на классическое применение алгоритма хэширования по поиску первого или второго прообраза;
* найти корректную пару «сообщение – код аутентификации данного сообщения» или вычислить код аутентификации сообщения для заданного сообщения без знания секретного ключа;
* получить ключ, используемый для вычисления HMAC.

## 3. Блок-схема алгоритма

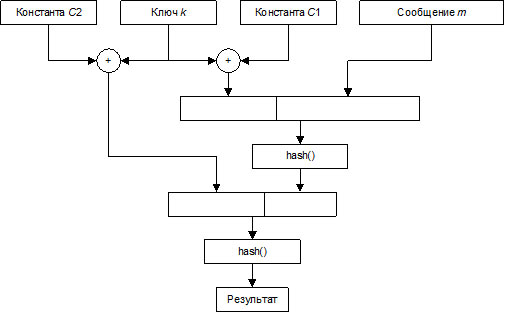


Рисунок 3.1 Блох-схема алгоритма HMAC

## Вывод

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

К таким данным могут относиться, например, данные, передаваемые в запросах API, когда критически важна целостность передаваемой информации, или же при передаче данных из Web-форм.

Была получена простая реализация, позволяющая подписывать любые данные и проверять переданные подписанные данные. Теперь можно подписывать данные, передаваемые через HTTP/REST API, или же создавать продвинутые CSRF-токены для форм и быть уверенными в том, что получаемые данные оригинальны и консистентны.

## Приложение 1. Исходный код программы

import sys

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 calc\_sigma0(num):

num = (calc\_rotate\_right(num, 7) ^

calc\_rotate\_right(num, 18) ^

(num >> 3))

return num

def calc\_sigma1(num):

num = (calc\_rotate\_right(num, 17) ^

calc\_rotate\_right(num, 19) ^

(num >> 10))

return num

def calc\_capsigma0(num):

num = (calc\_rotate\_right(num, 2) ^

calc\_rotate\_right(num, 13) ^

calc\_rotate\_right(num, 22))

return num

def calc\_capsigma1(num):

num = (calc\_rotate\_right(num, 6) ^

calc\_rotate\_right(num, 11) ^

calc\_rotate\_right(num, 25))

return num

def calc\_ch(x, y, z):

return (x & y) ^ (~x & z)

def calc\_maj(x, y, z):

return (x & y) ^ (x & z) ^ (y & z)

def calc\_rotate\_right(num, shift, size=32):

return (num >> shift) | (num << size - shift)

def sha256(message):

if isinstance(message, str):

message = bytearray(message, 'ascii')

elif isinstance(message, bytes):

message = bytearray(message)

elif not isinstance(message, bytearray):

raise TypeError

length = len(message) \* 8

message.append(0x80)

while (len(message) \* 8 + 64) % 512 != 0:

message.append(0x00)

message += length.to\_bytes(8, 'big')

blocks = []

for i in range(0, len(message), 64):

blocks.append(message[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 = calc\_sigma1(int.from\_bytes(message\_schedule[t - 2], 'big'))

term2 = int.from\_bytes(message\_schedule[t - 7], 'big')

term3 = calc\_sigma0(int.from\_bytes(message\_schedule[t - 15], 'big'))

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 + calc\_capsigma1(e) + calc\_ch(e, f, g) + K[t] +

int.from\_bytes(message\_schedule[t], 'big')) % 2 \*\* 32)

t2 = (calc\_capsigma0(a) + calc\_maj(a, 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\_md5(key, message):

i\_key = bytearray()

o\_key = bytearray()

key = key.encode()

message = message.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])

return sha256(bytes(o\_key) + sha256(bytes(i\_key) + message)).hex()

if \_\_name\_\_ == "\_\_main\_\_":

k = "123123"

if len(sys.argv) < 2:

exit(1)

file\_in = sys.argv[1]

file\_out = sys.argv[2]

with open(file\_in, "r+") as f:

m = f.read()

print("Raw message: ", m)

print("Key: ", k)

h\_key = hmac\_md5(k, m)

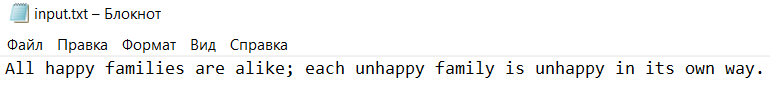
print("Hash key: ", h\_key)

with open(file\_out, "w+") as f:

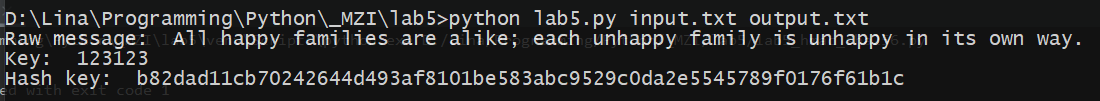
f.write(h\_key)

## Приложение 2. Скриншот работы программы

**input.txt**



**k = "123123"**



**output.txt**

