Министерство образования Республики Беларусь
Белорусский государственный университет информатики и радиоэлектроники
Кафедра информатики

ОТЧЕТ
по лабораторной работе №5
Хэш-функции

Выполнил:
студент гр. 653501
Шинкевич Г. С.

Проверил:
Артемьев В. С.

Минск 2019

ЗАДАНИЕ:

Реализовать программное средство контроля целостности сообщений с помощью вычисления хэш-функции и алгоритма HMAC.

MD5

128-битный алгоритм хеширования, предназначенный для создания «отпечатков» или дайджестов сообщения произвольной длины и последующей проверки их подлинности.

На вход алгоритма поступает входной поток данных, хеш которого необходимо найти. Длина сообщения измеряется в битах и может быть любой (в том числе нулевой). Запишем длину сообщения в L. Это число целое и неотрицательное. Кратность каким-либо числам необязательна. После поступления данных идёт процесс подготовки потока к вычислениям.

1. Выравнивание потока:

Сначала к концу потока дописывают единичный бит. Затем добавляют некоторое число нулевых бит такое, чтобы новая длина потока L' стала сравнима с 448 по модулю 512.

2. Добавление длины сообщения:

В конец сообщения дописывают 64-битное представление длины данных (количество бит в сообщении) до выравнивания. Сначала записывают младшие 4 байта, затем старшие.

3. Инициализация буфера:

Для вычислений инициализируются 4 переменных размером по 32 бита и задаются начальные значения шестнадцатеричными числами (порядок байтов little-endian, сначала младший байт):

```
A = 01 23 45 67; // 67452301h
B = 89 AB CD EF; // EFCDAB89h
C = FE DC BA 98; // 98BADCFEh
D = 76 54 32 10. // 10325476h
```

4. Вычисление в цикле:

Для каждого раунда потребуется своя функция. Введём функции от трёх параметров — слов, результатом также будет слово:

1-й раунд: $\mathrm{FunF}(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z)$,
2-й раунд: $\mathrm{FunG}(X, Y, Z) = (X \wedge Z) \vee (\neg Z \wedge Y)$,
3-й раунд: $\mathrm{FunH}(X, Y, Z) = X \oplus Y \oplus Z$,
4-й раунд: $\mathrm{FunI}(X, Y, Z) = Y \oplus (\neg Z \vee X)$,

Определим таблицу констант T[1…64] — 64-элементная таблица данных, построенная следующим образом: T[n] = int($2^{32} \cdot |\sin n|$).

Каждый 512-битный блок проходит 4 этапа вычислений по 16 раундов. Для этого блок представляется в виде массива X из 16 слов по 32 бита. Все раунды однотипны и имеют вид: [abcd k s i], определяемый как a = b + ((a + Fun(b, c, d) + X[k] + T[i]) ⋘ s), где k — номер 32-битного слова из текущего 512-битного блока сообщения, и … ⋘ s — циклический сдвиг влево на s бит полученного 32-битного аргумента. Число s задается отдельно для каждого раунда.

Заносим в блок данных элемент n из массива 512-битных блоков. Сохраняются значения A, B, C и D, оставшиеся после операций над предыдущими блоками (или их начальные значения, если блок первый).

AA = A

BB = B

CC = C

DD = D

Этап 1:

```
/* [abcd k s i] a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD  0 7  1][DABC  1 12  2][CDAB  2 17  3][BCDA  3 22  4]
[ABCD  4 7  5][DABC  5 12  6][CDAB  6 17  7][BCDA  7 22  8]
[ABCD  8 7  9][DABC  9 12 10][CDAB 10 17 11][BCDA 11 22 12]
[ABCD 12 7 13][DABC 13 12 14][CDAB 14 17 15][BCDA 15 22 16]
```

Этап 2:

```
/* [abcd k s i] a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD  1 5 17][DABC  6 9 18][CDAB 11 14 19][BCDA  0 20 20]
[ABCD  5 5 21][DABC 10 9 22][CDAB 15 14 23][BCDA  4 20 24]
[ABCD  9 5 25][DABC 14 9 26][CDAB  3 14 27][BCDA  8 20 28]
[ABCD 13 5 29][DABC  2 9 30][CDAB  7 14 31][BCDA 12 20 32]
```

Этап 3:

```
/* [abcd k s i] a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD  5 4 33][DABC  8 11 34][CDAB 11 16 35][BCDA 14 23 36]
[ABCD  1 4 37][DABC  4 11 38][CDAB  7 16 39][BCDA 10 23 40]
[ABCD 13 4 41][DABC  0 11 42][CDAB  3 16 43][BCDA  6 23 44]
[ABCD  9 4 45][DABC 12 11 46][CDAB 15 16 47][BCDA  2 23 48]
```

Этап 4:

```
/* [abcd k s i] a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD  0 6 49][DABC  7 10 50][CDAB 14 15 51][BCDA  5 21 52]
[ABCD 12 6 53][DABC  3 10 54][CDAB 10 15 55][BCDA  1 21 56]
[ABCD  8 6 57][DABC 15 10 58][CDAB  6 15 59][BCDA 13 21 60]
[ABCD  4 6 61][DABC 11 10 62][CDAB  2 15 63][BCDA  9 21 64]
```

Суммируем с результатом предыдущего цикла:

A = AA + A
B = BB + B
C = CC + C
D = DD + D

После окончания цикла необходимо проверить, есть ли ещё блоки для вычислений. Если да, то переходим к следующему элементу массива (n + 1) и повторяем цикл.

5. Результат вычислений:

Результат вычислений находится в буфере ABCD, это и есть хеш. Если выводить побайтово, начиная с младшего байта A и закончив старшим байтом D, то мы получим MD5-хеш.

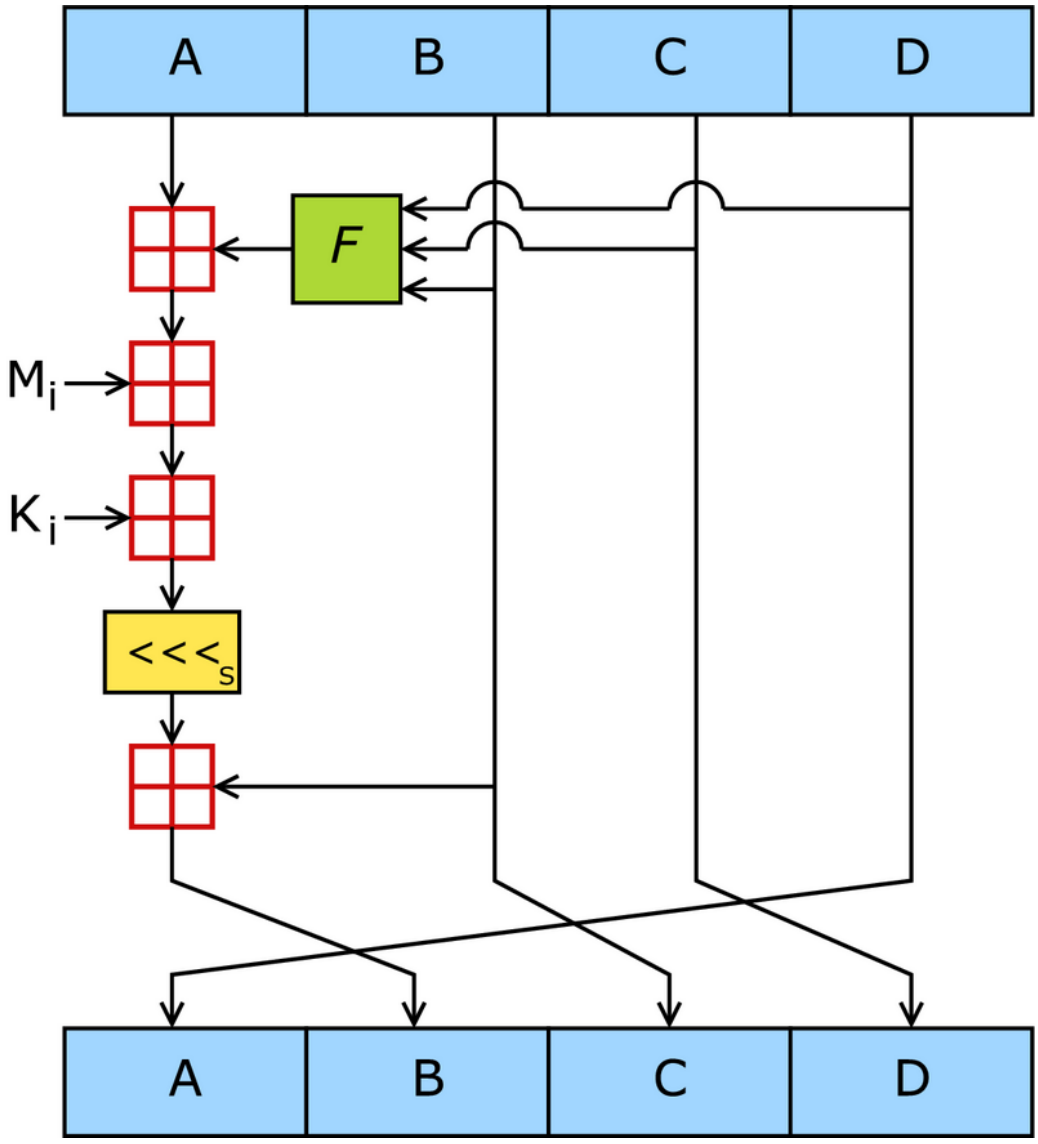

Рисунок 1. Алгоритм MD5

HMAC

Механизм проверки целостности информации, позволяющий гарантировать то, что данные, передаваемые или хранящиеся в ненадёжной среде, не были изменены посторонними. Механизм HMAC использует MAC. MAC — стандарт, описывающий способ обмена данными и способ проверки целостности передаваемых данных с использованием секретного ключа. Два клиента, использующие HMAC, как правило, разделяют общий секретный ключ. HMAC — надстройка над MAC; механизм обмена данными с использованием секретного ключа (как в MAC) и хеш-функций.

Алгоритм

Алгоритм HMAC можно записать в виде одной формулы:

$$\mathrm{HMAC}_K(text) = \mathrm{H}\left((K \oplus opad) \| \mathrm{H}\left((K \oplus ipad) \| text\right)\right)$$

opad — блок вида (0x5c 0x5c 0x5c ... 0x5c), где байт 0x5c повторяется b раз.
ipad — блок вида (0x36 0x36 0x36 ... 0x36), где байт 0x36 повторяется b раз.

1. Получить K0 путём уменьшения или увеличения ключа K до размера блока.

2. Получить блок Si размером в b байт с помощью операции «побитовое исключающее ИЛИ»:

Si = xor(K0, ipad)

3. Получить блок S0 размером в b байт с помощью операции «побитовое исключающее ИЛИ»:

S0 = xor(K0, opad)

4. Разбить сообщение text на блоки размером b байт.

5. Склеить строку (последовательность байт) Si с каждым блоком сообщения M.

6. К строке, полученной на прошлом шаге, применить хеш-функцию H.

7. Склеить строку S0 со строкой, полученной от хеш-функции H на прошлом шаге.

8. К строке, полученной на прошлом шаге, применить хеш-функцию H.

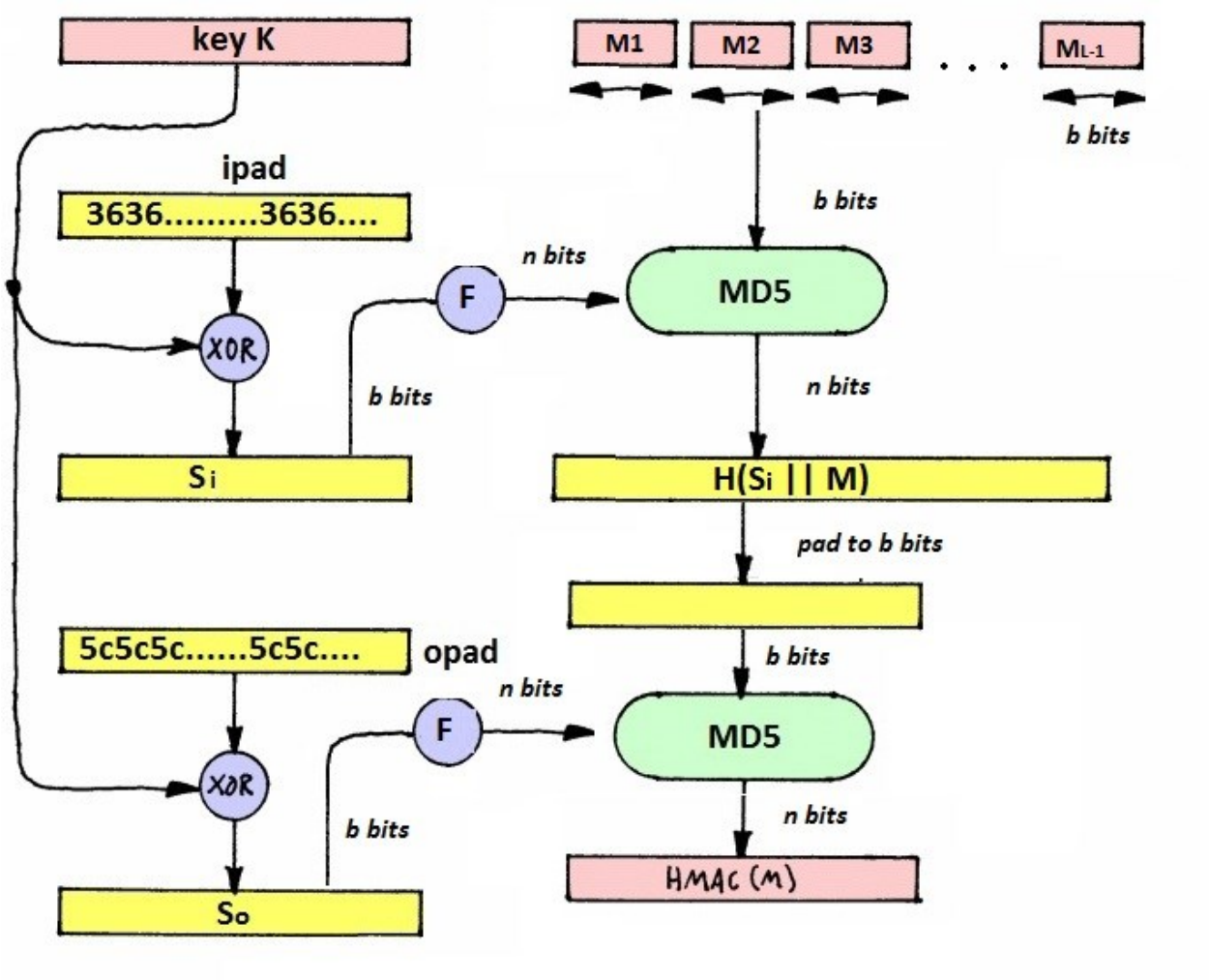

Рисунок 2. Алгоритм HMAC

ДЕМОНСТРАЦИЯ РАБОТЫ

```
MD5('The quick brown fox jumps over the lazy dog') = 9E107D9D372BB6826BD81D3542A419D6
MD5('The quick brown fox jumps over the lazy dog.') = E4D909C290D0FB1CA068FFADDF22CBD0
MD5('md5') = 1BC29B36F623BA82AAF6724FD3B16718
MD5('') = D41D8CD98F00B204E9800998ECF8427E
```

```
HMAC('key', 'The quick brown fox jumps over the lazy dog') = 80070713463E7749B90C2DC24911E275
HMAC('key', 'The quick brown fox jumps over the lazy dog.') = 120A17985A1E97BF8F0E38A52FB9FE79
HMAC('key', 'md5') = 3710C1151691936D1DE2B8FA3A76F083
HMAC('key', '') = 63530468A04E386459855DA0063B6596
```

ВЫВОД

В результате лабораторной работы была написана программа для контроля целостности сообщений, использующая алгоритм HMAC-MD5.

ЛИСТИНГ ПРОГРАММЫ

md5.py

```
SHIFTS = [
    7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,
    5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20,
    4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,
    6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21
]

CONSTANTS = [
    0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee,
    0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501,
    0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be,
    0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821,
    0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa,
    0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8,
    0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed,
    0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a,
    0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c,
    0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70,
    0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05,
    0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665,
    0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039,
    0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1,
    0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1,
    0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391
]


def hex_to_bin(hex_number, size=32):
    return bin(hex_number)[2:].zfill(size)


def bin_to_hex(binary_string):
    return '%0*X' % ((len(binary_string) + 3) // 4, int(binary_string, 2))
```

```
def str_to_bin(text):
    return ''.join(format(ord(char), 'b').zfill(8) for char in text)


def b_xor(u, v):
    return '{0:b}'.format(int(u, 2) ^ int(v, 2)).zfill(32)


def b_or(u, v):
    return '{0:b}'.format(int(u, 2) | int(v, 2)).zfill(32)


def b_and(u, v):
    return '{0:b}'.format(int(u, 2) & int(v, 2)).zfill(32)


def b_not(u):
    return '{0:b}'.format(int(u, 2) ^ 0xFFFFFFFF).zfill(32)


def plus_32(u, v):
    return '{0:b}'.format((int(u, 2) + int(v, 2)) % 2 ** 32).zfill(32)


def left_rotate(u, r):
    return u[r:] + u[:r]


def chunks(string, size):
    for i in range(0, len(string), size):
        yield string[i:i + size]


def to_le(string, size=8):
    return "".join(list(chunks(string, size))[::-1])


def md5(data):
    message = data
```

```
    orig_len = len(message)
    message += "1"
    while True:
        if len(message) % 512 == 448:
            break
        message += "0"
    r = "{0:b}".format(orig_len % (2 ** 64)).zfill(64)
    r1, r2 = r[:32], r[len(r) - 32:]
    message += to_le(r2) + to_le(r1)

    a0 = hex_to_bin(0x67452301)
    b0 = hex_to_bin(0xefcdab89)
    c0 = hex_to_bin(0x98badcfe)
    d0 = hex_to_bin(0x10325476)

    for chunk in chunks(message, 512):
        M = [to_le(c) for c in chunks(chunk, 32)]

        A, B, C, D = a0, b0, c0, d0

        for i in range(64):
            F = ""
            g = 0
            if 0 <= i <= 15:
                F = b_or(b_and(B, C), b_and(b_not(B), D))
                g = i
            elif 16 <= i <= 31:
                F = b_or(b_and(D, B), b_and(b_not(D), C))
                g = (5 * i + 1) % 16
            elif 32 <= i <= 47:
                F = b_xor(b_xor(B, C), D)
                g = (3 * i + 5) % 16
            elif 48 <= i <= 63:
                F = b_xor(C, b_or(B, b_not(D)))
                g = (7 * i) % 16

            F = plus_32(plus_32(plus_32(A, F), M[g]),
hex_to_bin(CONSTANTS[i]))
            A = D
```

```
            D = C
            C = B
            B = plus_32(B, left_rotate(F, SHIFTS[i]))

        a0 = plus_32(A, a0)
        b0 = plus_32(B, b0)
        c0 = plus_32(C, c0)
        d0 = plus_32(D, d0)

    return to_le(a0) + to_le(b0) + to_le(c0) + to_le(d0)


if __name__ == "__main__":
    text = "The quick brown fox jumps over the lazy dog"
    result = bin_to_hex(md5(str_to_bin(text)))
    print(f"MD5('{text}') = {result}")
    text = "The quick brown fox jumps over the lazy dog."
    result = bin_to_hex(md5(str_to_bin(text)))
    print(f"MD5('{text}') = {result}")
    text = "md5"
    result = bin_to_hex(md5(str_to_bin(text)))
    print(f"MD5('{text}') = {result}")
    text = ""
    result = bin_to_hex(md5(str_to_bin(text)))
    print(f"MD5('{text}') = {result}")
```

hmac.py

```
from md5 import md5

BLOCK_SIZE = 512


def bin_to_hex(binary_string):
    return '%0*X' % ((len(binary_string) + 3) // 4, int(binary_string, 2))


def hex_to_bin(hex_number, size=8):
    return bin(hex_number)[2:].zfill(size)
```

```
def str_to_bin(text):
    return ''.join(format(ord(char), 'b').zfill(8) for char in text)


def b_xor(u, v):
    return '{0:b}'.format(int(u, 2) ^ int(v, 2)).zfill(512)


def hmac(key, message):
    if len(key) > BLOCK_SIZE:
        key = md5(key)
    if len(key) < BLOCK_SIZE:
        key += "0" * (BLOCK_SIZE - len(key))

    ipad = hex_to_bin(0x36) * (BLOCK_SIZE // 8)
    opad = hex_to_bin(0x5c) * (BLOCK_SIZE // 8)

    ikeypad = b_xor(ipad, key)
    okeypad = b_xor(opad, key)

    return md5(okeypad + md5(ikeypad + message))


if __name__ == "__main__":
    key = "key"
    text = "The quick brown fox jumps over the lazy dog"
    result = bin_to_hex(hmac(str_to_bin(key), str_to_bin(text)))
    print(f"HMAC('{key}', '{text}') = {result}")
    text = "The quick brown fox jumps over the lazy dog."
    result = bin_to_hex(hmac(str_to_bin(key), str_to_bin(text)))
    print(f"HMAC('{key}', '{text}') = {result}")
    text = "md5"
    result = bin_to_hex(hmac(str_to_bin(key), str_to_bin(text)))
    print(f"HMAC('{key}', '{text}') = {result}")
    text = ""
    result = bin_to_hex(hmac(str_to_bin(key), str_to_bin(text)))
    print(f"HMAC('{key}', '{text}') = {result}")
```