Науменко Д.

КН 922.а

Лабораторна робота №1

Гешування

Мета : Дослідити принципи роботи гешування

Завдання : Дослідити існуючі механізми гешування. Реалізувати алгоритм гешування SHA (будь-якої версії). Довести коректність роботи реалізованого алгоритму шляхом порівняння результатів з існуючими реалізаціями.

Хід роботи

Найпопулярніші геш-функції на пайтоні є MD5 та SHA.

MD5: Алгоритм виробляє хеш зі значенням 128 бітів. Широко використовується для перевірки цілісності даних. Не підходить для використання в інших областях через вразливість безпеки MD5.

SHA: Група алгоритмів, розроблених NSA Сполучених Штатів. Вони є частиною Федерального стандарту обробки інформації США. Ці алгоритми широко використовуються у кількох криптографічних додатках. Довжина повідомлення варіюється від 160 до 512 біт.

```
import struct

SHA_BLOCKSIZE = 64
SHA_DIGESTSIZE = 32


def new_shaobject():
    return {
        'digest': [0] * 8,
        'count_lo': 0,
        'count_hi': 0,
        'data': [0] * SHA_BLOCKSIZE,
        'local': 0,
        'digestsize': 0
    }


ROR = lambda x, y: (((x & 0xffffffff) >> (y & 31)) | (x << (32 - (y & 31)))) &
0xffffffff
Ch = lambda x, y, z: (z ^ (x & (y ^ z)))
Maj = lambda x, y, z: (((x | y) & z) | (x & y))
S = lambda x, n: ROR(x, n)
R = lambda x, n: (x & 0xffffffff) >> n
Sigma0 = lambda x: (S(x, 2) ^ S(x, 13) ^ S(x, 22))
Sigma1 = lambda x: (S(x, 6) ^ S(x, 11) ^ S(x, 25))
Gamma0 = lambda x: (S(x, 7) ^ S(x, 18) ^ R(x, 3))
Gamma1 = lambda x: (S(x, 17) ^ S(x, 19) ^ R(x, 10))
```

```
def sha_transform(sha_info):
    W = []

    d = sha_info['data']
    for i in xrange(0, 16):
        W.append((d[4 * i] << 24) + (d[4 * i + 1] << 16) + (d[4 * i + 2] << 8) +
d[4 * i + 3])

    for i in xrange(16, 64):
        W.append((Gamma1(W[i - 2]) + W[i - 7] + Gamma0(W[i - 15]) + W[i - 16]) &
0xffffffff)

    ss = sha_info['digest'][:]

    def RND(a, b, c, d, e, f, g, h, i, ki):
        t0 = h + Sigma1(e) + Ch(e, f, g) + ki + W[i];
        t1 = Sigma0(a) + Maj(a, b, c);
        d += t0;
        h = t0 + t1;
        return d & 0xffffffff, h & 0xffffffff

    ss[3], ss[7] = RND(ss[0], ss[1], ss[2], ss[3], ss[4], ss[5], ss[6], ss[7],
0, 0x428a2f98);
    ss[2], ss[6] = RND(ss[7], ss[0], ss[1], ss[2], ss[3], ss[4], ss[5], ss[6],
1, 0x71374491);
    ss[1], ss[5] = RND(ss[6], ss[7], ss[0], ss[1], ss[2], ss[3], ss[4], ss[5],
2, 0xb5c0fbcf);
    ss[0], ss[4] = RND(ss[5], ss[6], ss[7], ss[0], ss[1], ss[2], ss[3], ss[4],
3, 0xe9b5dba5);
    ss[7], ss[3] = RND(ss[4], ss[5], ss[6], ss[7], ss[0], ss[1], ss[2], ss[3],
4, 0x3956c25b);
    ss[6], ss[2] = RND(ss[3], ss[4], ss[5], ss[6], ss[7], ss[0], ss[1], ss[2],
5, 0x59f111f1);
    ss[5], ss[1] = RND(ss[2], ss[3], ss[4], ss[5], ss[6], ss[7], ss[0], ss[1],
6, 0x923f82a4);
    ss[4], ss[0] = RND(ss[1], ss[2], ss[3], ss[4], ss[5], ss[6], ss[7], ss[0],
7, 0xab1c5ed5);
    ss[3], ss[7] = RND(ss[0], ss[1], ss[2], ss[3], ss[4], ss[5], ss[6], ss[7],
8, 0xd807aa98);
    ss[2], ss[6] = RND(ss[7], ss[0], ss[1], ss[2], ss[3], ss[4], ss[5], ss[6],
9, 0x12835b01);
    ss[1], ss[5] = RND(ss[6], ss[7], ss[0], ss[1], ss[2], ss[3], ss[4], ss[5],
10, 0x243185be);
    ss[0], ss[4] = RND(ss[5], ss[6], ss[7], ss[0], ss[1], ss[2], ss[3], ss[4],
11, 0x550c7dc3);
    ss[7], ss[3] = RND(ss[4], ss[5], ss[6], ss[7], ss[0], ss[1], ss[2], ss[3],
12, 0x72be5d74);
    ss[6], ss[2] = RND(ss[3], ss[4], ss[5], ss[6], ss[7], ss[0], ss[1], ss[2],
13, 0x80deb1fe);
    ss[5], ss[1] = RND(ss[2], ss[3], ss[4], ss[5], ss[6], ss[7], ss[0], ss[1],
14, 0x9bdc06a7);
    ss[4], ss[0] = RND(ss[1], ss[2], ss[3], ss[4], ss[5], ss[6], ss[7], ss[0],
15, 0xc19bf174);
    ss[3], ss[7] = RND(ss[0], ss[1], ss[2], ss[3], ss[4], ss[5], ss[6], ss[7],
16, 0xe49b69c1);
    ss[2], ss[6] = RND(ss[7], ss[0], ss[1], ss[2], ss[3], ss[4], ss[5], ss[6],
17, 0xefbe4786);
    ss[1], ss[5] = RND(ss[6], ss[7], ss[0], ss[1], ss[2], ss[3], ss[4], ss[5],
18, 0x0fc19dc6);
    ss[0], ss[4] = RND(ss[5], ss[6], ss[7], ss[0], ss[1], ss[2], ss[3], ss[4],
19, 0x240ca1cc);
    ss[7], ss[3] = RND(ss[4], ss[5], ss[6], ss[7], ss[0], ss[1], ss[2], ss[3],
20, 0x2de92c6f);
    ss[6], ss[2] = RND(ss[3], ss[4], ss[5], ss[6], ss[7], ss[0], ss[1], ss[2],
21, 0x4a7484aa);
    ss[5], ss[1] = RND(ss[2], ss[3], ss[4], ss[5], ss[6], ss[7], ss[0], ss[1],
22, 0x5cb0a9dc);
```

```
    ss[4], ss[0] = RND(ss[1], ss[2], ss[3], ss[4], ss[5], ss[6], ss[7], ss[0],
23, 0x76f988da);
    ss[3], ss[7] = RND(ss[0], ss[1], ss[2], ss[3], ss[4], ss[5], ss[6], ss[7],
24, 0x983e5152);
    ss[2], ss[6] = RND(ss[7], ss[0], ss[1], ss[2], ss[3], ss[4], ss[5], ss[6],
25, 0xa831c66d);
    ss[1], ss[5] = RND(ss[6], ss[7], ss[0], ss[1], ss[2], ss[3], ss[4], ss[5],
26, 0xb00327c8);
    ss[0], ss[4] = RND(ss[5], ss[6], ss[7], ss[0], ss[1], ss[2], ss[3], ss[4],
27, 0xbf597fc7);
    ss[7], ss[3] = RND(ss[4], ss[5], ss[6], ss[7], ss[0], ss[1], ss[2], ss[3],
28, 0xc6e00bf3);
    ss[6], ss[2] = RND(ss[3], ss[4], ss[5], ss[6], ss[7], ss[0], ss[1], ss[2],
29, 0xd5a79147);
    ss[5], ss[1] = RND(ss[2], ss[3], ss[4], ss[5], ss[6], ss[7], ss[0], ss[1],
30, 0x06ca6351);
    ss[4], ss[0] = RND(ss[1], ss[2], ss[3], ss[4], ss[5], ss[6], ss[7], ss[0],
31, 0x14292967);
    ss[3], ss[7] = RND(ss[0], ss[1], ss[2], ss[3], ss[4], ss[5], ss[6], ss[7],
32, 0x27b70a85);
    ss[2], ss[6] = RND(ss[7], ss[0], ss[1], ss[2], ss[3], ss[4], ss[5], ss[6],
33, 0x2e1b2138);
    ss[1], ss[5] = RND(ss[6], ss[7], ss[0], ss[1], ss[2], ss[3], ss[4], ss[5],
34, 0x4d2c6dfc);
    ss[0], ss[4] = RND(ss[5], ss[6], ss[7], ss[0], ss[1], ss[2], ss[3], ss[4],
35, 0x53380d13);
    ss[7], ss[3] = RND(ss[4], ss[5], ss[6], ss[7], ss[0], ss[1], ss[2], ss[3],
36, 0x650a7354);
    ss[6], ss[2] = RND(ss[3], ss[4], ss[5], ss[6], ss[7], ss[0], ss[1], ss[2],
37, 0x766a0abb);
    ss[5], ss[1] = RND(ss[2], ss[3], ss[4], ss[5], ss[6], ss[7], ss[0], ss[1],
38, 0x81c2c92e);
    ss[4], ss[0] = RND(ss[1], ss[2], ss[3], ss[4], ss[5], ss[6], ss[7], ss[0],
39, 0x92722c85);
    ss[3], ss[7] = RND(ss[0], ss[1], ss[2], ss[3], ss[4], ss[5], ss[6], ss[7],
40, 0xa2bfe8a1);
    ss[2], ss[6] = RND(ss[7], ss[0], ss[1], ss[2], ss[3], ss[4], ss[5], ss[6],
41, 0xa81a664b);
    ss[1], ss[5] = RND(ss[6], ss[7], ss[0], ss[1], ss[2], ss[3], ss[4], ss[5],
42, 0xc24b8b70);
    ss[0], ss[4] = RND(ss[5], ss[6], ss[7], ss[0], ss[1], ss[2], ss[3], ss[4],
43, 0xc76c51a3);
    ss[7], ss[3] = RND(ss[4], ss[5], ss[6], ss[7], ss[0], ss[1], ss[2], ss[3],
44, 0xd192e819);
    ss[6], ss[2] = RND(ss[3], ss[4], ss[5], ss[6], ss[7], ss[0], ss[1], ss[2],
45, 0xd6990624);
    ss[5], ss[1] = RND(ss[2], ss[3], ss[4], ss[5], ss[6], ss[7], ss[0], ss[1],
46, 0xf40e3585);
    ss[4], ss[0] = RND(ss[1], ss[2], ss[3], ss[4], ss[5], ss[6], ss[7], ss[0],
47, 0x106aa070);
    ss[3], ss[7] = RND(ss[0], ss[1], ss[2], ss[3], ss[4], ss[5], ss[6], ss[7],
48, 0x19a4c116);
    ss[2], ss[6] = RND(ss[7], ss[0], ss[1], ss[2], ss[3], ss[4], ss[5], ss[6],
49, 0x1e376c08);
    ss[1], ss[5] = RND(ss[6], ss[7], ss[0], ss[1], ss[2], ss[3], ss[4], ss[5],
50, 0x2748774c);
    ss[0], ss[4] = RND(ss[5], ss[6], ss[7], ss[0], ss[1], ss[2], ss[3], ss[4],
51, 0x34b0bcb5);
    ss[7], ss[3] = RND(ss[4], ss[5], ss[6], ss[7], ss[0], ss[1], ss[2], ss[3],
52, 0x391c0cb3);
    ss[6], ss[2] = RND(ss[3], ss[4], ss[5], ss[6], ss[7], ss[0], ss[1], ss[2],
53, 0x4ed8aa4a);
    ss[5], ss[1] = RND(ss[2], ss[3], ss[4], ss[5], ss[6], ss[7], ss[0], ss[1],
54, 0x5b9cca4f);
    ss[4], ss[0] = RND(ss[1], ss[2], ss[3], ss[4], ss[5], ss[6], ss[7], ss[0],
55, 0x682e6ff3);
    ss[3], ss[7] = RND(ss[0], ss[1], ss[2], ss[3], ss[4], ss[5], ss[6], ss[7],
56, 0x748f82ee);
```

```
    ss[2], ss[6] = RND(ss[7], ss[0], ss[1], ss[2], ss[3], ss[4], ss[5], ss[6],
57, 0x78a5636f);
    ss[1], ss[5] = RND(ss[6], ss[7], ss[0], ss[1], ss[2], ss[3], ss[4], ss[5],
58, 0x84c87814);
    ss[0], ss[4] = RND(ss[5], ss[6], ss[7], ss[0], ss[1], ss[2], ss[3], ss[4],
59, 0x8cc70208);
    ss[7], ss[3] = RND(ss[4], ss[5], ss[6], ss[7], ss[0], ss[1], ss[2], ss[3],
60, 0x90befffa);
    ss[6], ss[2] = RND(ss[3], ss[4], ss[5], ss[6], ss[7], ss[0], ss[1], ss[2],
61, 0xa4506ceb);
    ss[5], ss[1] = RND(ss[2], ss[3], ss[4], ss[5], ss[6], ss[7], ss[0], ss[1],
62, 0xbef9a3f7);
    ss[4], ss[0] = RND(ss[1], ss[2], ss[3], ss[4], ss[5], ss[6], ss[7], ss[0],
63, 0xc67178f2);

    dig = []
    for i, x in enumerate(sha_info['digest']):
        dig.append((x + ss[i]) & 0xffffffff)
    sha_info['digest'] = dig


def sha_init():
    sha_info = new_shaobject()
    sha_info['digest'] = [0x6A09E667, 0xBB67AE85, 0x3C6EF372, 0xA54FF53A,
0x510E527F, 0x9B05688C, 0x1F83D9AB,
                          0x5BE0CD19]
    sha_info['count_lo'] = 0
    sha_info['count_hi'] = 0
    sha_info['local'] = 0
    sha_info['digestsize'] = 32
    return sha_info


def sha224_init():
    sha_info = new_shaobject()
    sha_info['digest'] = [0xc1059ed8, 0x367cd507, 0x3070dd17, 0xf70e5939,
0xffc00b31, 0x68581511, 0x64f98fa7,
                          0xbefa4fa4]
    sha_info['count_lo'] = 0
    sha_info['count_hi'] = 0
    sha_info['local'] = 0
    sha_info['digestsize'] = 28
    return sha_info


def getbuf(s):
    if isinstance(s, str):
        return s
    elif isinstance(s, unicode):
        return str(s)
    else:
        return buffer(s)


def sha_update(sha_info, buffer):
    count = len(buffer)
    buffer_idx = 0
    clo = (sha_info['count_lo'] + (count << 3)) & 0xffffffff
    if clo < sha_info['count_lo']:
        sha_info['count_hi'] += 1
    sha_info['count_lo'] = clo

    sha_info['count_hi'] += (count >> 29)

    if sha_info['local']:
        i = SHA_BLOCKSIZE - sha_info['local']
        if i > count:
            i = count
```

```
        # copy buffer
        for x in enumerate(buffer[buffer_idx:buffer_idx + i]):
            sha_info['data'][sha_info['local'] + x[0]] = struct.unpack('B',
x[1])[0]

        count -= i
        buffer_idx += i

        sha_info['local'] += i
        if sha_info['local'] == SHA_BLOCKSIZE:
            sha_transform(sha_info)
            sha_info['local'] = 0
        else:
            return

    while count >= SHA_BLOCKSIZE:
        # copy buffer
        sha_info['data'] = [struct.unpack('B', c)[0] for c in
buffer[buffer_idx:buffer_idx + SHA_BLOCKSIZE]]
        count -= SHA_BLOCKSIZE
        buffer_idx += SHA_BLOCKSIZE
        sha_transform(sha_info)

    # copy buffer
    pos = sha_info['local']
    sha_info['data'][pos:pos + count] = [struct.unpack('B', c)[0] for c in
buffer[buffer_idx:buffer_idx + count]]
    sha_info['local'] = count


def sha_final(sha_info):
    lo_bit_count = sha_info['count_lo']
    hi_bit_count = sha_info['count_hi']
    count = (lo_bit_count >> 3) & 0x3f
    sha_info['data'][count] = 0x80;
    count += 1
    if count > SHA_BLOCKSIZE - 8:
        # zero the bytes in data after the count
        sha_info['data'] = sha_info['data'][:count] + ([0] * (SHA_BLOCKSIZE -
count))
        sha_transform(sha_info)
        # zero bytes in data
        sha_info['data'] = [0] * SHA_BLOCKSIZE
    else:
        sha_info['data'] = sha_info['data'][:count] + ([0] * (SHA_BLOCKSIZE -
count))

    sha_info['data'][56] = (hi_bit_count >> 24) & 0xff
    sha_info['data'][57] = (hi_bit_count >> 16) & 0xff
    sha_info['data'][58] = (hi_bit_count >> 8) & 0xff
    sha_info['data'][59] = (hi_bit_count >> 0) & 0xff
    sha_info['data'][60] = (lo_bit_count >> 24) & 0xff
    sha_info['data'][61] = (lo_bit_count >> 16) & 0xff
    sha_info['data'][62] = (lo_bit_count >> 8) & 0xff
    sha_info['data'][63] = (lo_bit_count >> 0) & 0xff

    sha_transform(sha_info)

    dig = []
    for i in sha_info['digest']:
        dig.extend([((i >> 24) & 0xff), ((i >> 16) & 0xff), ((i >> 8) & 0xff),
(i & 0xff)])
    return ''.join([chr(i) for i in dig])


class sha256(object):
    digest_size = digestsize = SHA_DIGESTSIZE
```

```
    block_size = SHA_BLOCKSIZE

    def __init__(self, s=None):
        self._sha = sha_init()
        if s:
            sha_update(self._sha, getbuf(s))

    def update(self, s):
        sha_update(self._sha, getbuf(s))

    def digest(self):
        return sha_final(self._sha.copy())[:self._sha['digestsize']]

    def hexdigest(self):
        return ''.join(['%.2x' % ord(i) for i in self.digest()])

    def copy(self):
        new = sha256.__new__(sha256)
        new._sha = self._sha.copy()
        return new


class sha224(sha256):
    digest_size = digestsize = 28

    def __init__(self, s=None):
        self._sha = sha224_init()
        if s:
            sha_update(self._sha, getbuf(s))

    def copy(self):
        new = sha224.__new__(sha224)
        new._sha = self._sha.copy()
        return new


def test():
    a_str = "Dmitriy"

    assert 'e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855' ==
sha256().hexdigest()
    assert 'd7b553c6f09ac85d142415f857c5310f3bbbe7cdd787cce4b985acedd585266f' ==
sha256(a_str).hexdigest()
    assert '8113ebf33c97daa9998762aacafe750c7cefc2b2f173c90c59663a57fe626f21' ==
sha256(a_str * 7).hexdigest()

    s = sha256(a_str)
    s.update(a_str)
    assert '03d9963e05a094593190b6fc794cb1a3e1ac7d7883f0b5855268afeccc70d461' ==
s.hexdigest()
    print(a_str)


if __name__ == "__main__":
    test()
```

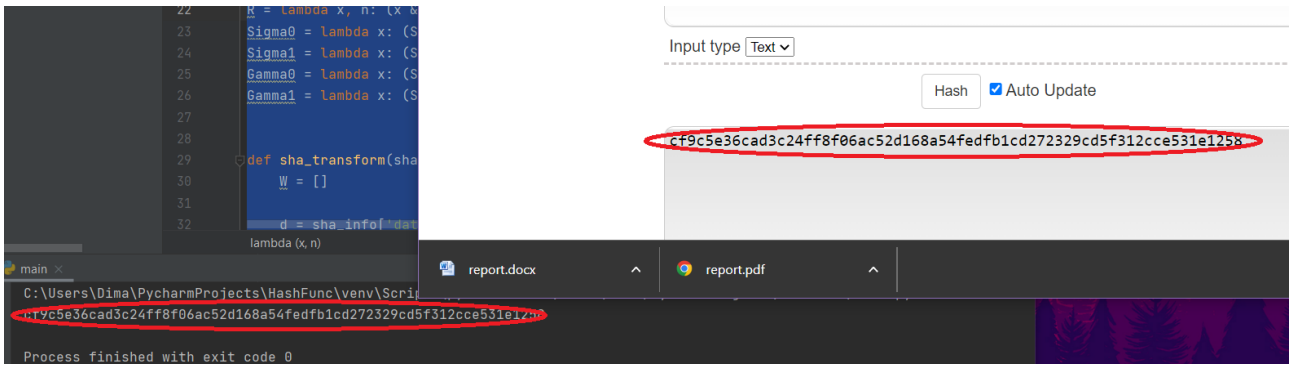

Рис. 1 – Результат.

Як бачимо, гешування виконано правильно. Остаточний результат співпадає с перевірочним сервісом.

Висновок : під час виконання лабораторної роботи я дослідив існуючі механізми гешування. Реалізував алгоритм гешування SHA256. Довів коректність роботи реалізованого алгоритму шляхом порівняння результатів з існуючими реалізаціями.