How does sha256 works

sha256的实现

源代码：

```python
init_hash = [0x6a09e667, 0xbb67ae85, 0x3c6ef372, 0xa54ff53a, 0x510e527f, 0x9b05688c, 0x1f83d9ab, 0x5be0cd19]

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 ror(x, k):
    return 0xffffffff & (((x & 0xffffffff) >> (k & 31)) | (x << (32 - (k & 31))))

def shr(x, k):
    return (x & 0xffffffff) >> k

Ch = lambda x, y, z: (x & y) ^ (~x & z)
Maj = lambda x, y, z: (x & y) ^ (x & z) ^ (y & z)
Sigma0 = lambda x: ror(x, 2) ^ ror(x, 13) ^ ror(x, 22)
Sigma1 = lambda x: ror(x, 6) ^ ror(x, 11) ^ ror(x, 25)

def pre_process(bits:str):
    l = len(bits)
    k = 0
    while (l + 1 + k) % 512 != 448:
        k += 1
    bits = bits + '1' + '0'*k + str(format(l, '064b'))
    chunks = []
    while len(bits) > 0:
        chunks.append(bits[:512])
        bits = bits[512:]
    return chunks

def loop(chunks:list):
    H = [_ for _ in init_hash]
    for chunk in chunks:
        words = [int(chunk[i:i+32],base=2) for i in range(0, 512, 32)]
        for i in range(16, 64):
            s0 = ror(words[i-15], 7) ^ ror(words[i-15], 18) ^ shr(words[i-15], 3)
            s1 = ror(words[i-2], 17) ^ ror(words[i-2], 19) ^ shr(words[i-2], 10)
            words.append((words[i-16] + s0 + words[i-7] + s1) & 0xffffffff)
        a, b, c, d, e, f, g, h = (H[i] for i in range(8))
        for i in range(64):
            t1 = h + Sigma1(e) + Ch(e,f,g) + K[i] + words[i]
            t2 = Sigma0(a) + Maj(a,b,c)
            h, g, f, e, d, c, b, a = g, f, e, (d+t1)&0xffffffff, c, b, a, (t1+t2)&0xffffffff
        for i in range(8):
            H[i] = (H[i] + [a, b, c, d, e, f, g, h][i]) & 0xffffffff
    digest = ''.join([str(format(H[i], '032b')) for i in range(8)])
    return digest

def my_sha256(bs:bytes):
    bits = ''.join([str(format(b, '08b')) for b in bs])
    chunks = pre_process(bits)
    digest = loop(chunks)
    hexdigest = hex(int(digest, base=2))[2:]
    return hexdigest

def test():
    s = 'abc'
    h = my_sha256(s.encode())
    print(h)
    # 0x13b332010c37792371b684711ea30e0a35267e2f520ac032132f25c9c9d76c2c

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

理想的hash函数应该满足:

- 确定性, 即相同的输入总产生相同的输出
- 高效性, 即可以快速计算任何消息的hash值
- 不可逆性, 即由散列值不能反推原消息
- 很小的改动也会引起hash值的很大变化
- 很难碰撞

下面对sha256验证这几个性质:

1. 确定性

随机生成一个消息，重复计算hash值并判断是否相同，重复1000次

测试代码及结果

```python
def test1():
    result = True
    for i in range(1000):
        s = bytes([random.randint(0, 0xff) for _ in range(random.randint(1, 1000))])
        if my_sha256(s) != my_sha256(s):
            result = False
    print(result) # True
```

可以看到， 多次计算同一个消息的hash，得到的结果必定是一样的

其实这也可以由算法的实现过程直接得出，对于每一个确定的输入，sha256的每一步操作都是确定的，所以产生的结果也是确定的

2. 高效性

随机生成多个消息并计算hash值，记录计算所用的时间

测试代码及结果

```python
def test2():
    print(time.process_time()) # 0.078125
    for i in range(10000):
        s = bytes([random.randint(0, 0xff) for _ in range(random.randint(1, 1000))])
        h = my_sha256(s)
    print(time.process_time()) # 32.03125
```

可以看到，即使是在很一般的机器上，做10000次hash计算也只需要30秒左右，这也是因为sha256采用的基本是比较快的加法和异或运算

3. 不可逆性

这一条性质可以直接由sha256的实现过程本身得出，sha256的输出是256位，而输入是任意长的消息，输入域远远大于输出域，所以sha256是不可逆的

4. 很小的改动引起很大的变化

随机生成一个字符串，每次改变一个字符，比较hash值的变化

测试代码及结果

```python
def test4():
    s = bytes([random.randint(0, 0xff) for _ in range(random.randint(1, 100))])
    old_h = my_sha256(s)
    diffs = []
    for i in range(len(s)):
        s = s[:i] + bytes(random.randint(0, 0xff)) + s[i+1:]
        h = my_sha256(s)
        diffs.append(sum([[0, 1][x!=y] for x, y in zip(h, old_h)]))
        old_h = h
    print(diffs)
# [62, 59, 59, 59, 60, 62, 60, 55, 61, 62, 57, 63, 58, 57, 61, 59, 59, 59, 58, 62, 62, 61, 60, 59, 59, 62, 61, 55, 63]
# [58, 54, 61, 63, 60, 60, 60, 62, 0, 57, 61, 60, 64, 63, 54, 60, 62, 60, 61, 57, 61, 59, 60, 63, 57, 60, 59, 61, 63, 60, 63, 61, 61, 59, 62, 57, 63]
# [58, 61, 61, 60, 63, 58, 58, 57, 62, 59, 61, 61, 58, 61, 59, 60, 61, 62, 60, 63, 62, 59, 61, 57, 55, 62, 61, 60, 62, 60, 64, 60, 62, 56, 60, 61, 61, 61, 60, 59, 57, 57, 52, 62, 59]
```

多次测试，可以看到即使是只改动了一个字符，hash值几乎每一位都变了

5. 很难碰撞

多次随机产生不同的字符串，看hash值是否有碰撞

```python
def test5():
    hs = []
    ss = []
    collided = 0
    for _ in range(100000):
        s = bytes([random.randint(0, 0xff) for _ in range(random.randint(1, 1000))])
        if s in ss:
            continue
        ss.append(s)
        h = my_sha256(s)
        if h in hs:
            collided += 1
            print('collided!', collided)
            continue
        hs.append(h)
    if collided == 0:
        print('no collision')
# no collision
# no collision
# no collision
```

多次尝试均无碰撞出现，可以看到sha256是很难碰撞的，一般情况下可以认为几乎不可能碰撞

经过验证可以看到，sha256作为一个hash算法有很好的表现