# SHA256在比特币区块链中发挥的作用

## SHA256 实现

local sha256 = {}

local 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

}

-- tool functions

local function initH256(H)

    H[1] = 0x6a09e667

    H[2] = 0xbb67ae85

    H[3] = 0x3c6ef372

    H[4] = 0xa54ff53a

    H[5] = 0x510e527f

    H[6] = 0x9b05688c

    H[7] = 0x1f83d9ab

    H[8] = 0x5be0cd19

    return H

end

local function rrotate(x, n) return ((x >> n) | (x << (32 - n))) end

local function preproc(msg, len)

    local extra = -(len + 1 + 8) % 64

    len = string.pack(">i8", 8 \* len)

    msg = msg .. "\128" .. string.rep("\0", extra) .. len

    assert(#msg % 64 == 0)

    return msg

end

local function digestblock(msg, i, H)

    local w = {string.unpack(">I4I4I4I4I4I4I4I4I4I4I4I4I4I4I4I4", msg, i)}

    for j = 17, 64 do

        local v = w[j - 15]

        local s0 = rrotate(v, 7) ~ rrotate(v, 18) ~ (v >> 3)

        v = w[j - 2]

        local s1 = rrotate(v, 17) ~ rrotate(v, 19) ~ (v >> 10)

        w[j] = (w[j - 16] + s0 + w[j - 7] + s1) & 0xffffffff

    end

    local a, b, c, d, e, f, g, h = H[1], H[2], H[3], H[4], H[5], H[6], H[7],

                                   H[8]

    for i = 1, 64 do

        local s0 = rrotate(a, 2) ~ rrotate(a, 13) ~ rrotate(a, 22)

        local maj = (a & b) ~ (a & c) ~ (b & c)

        local t2 = s0 + maj

        local s1 = rrotate(e, 6) ~ rrotate(e, 11) ~ rrotate(e, 25)

        local ch = (e & f) ~ (~e & g)

        local t1 = h + s1 + ch + k[i] + w[i]

        h = g

        g = f

        f = e

        e = (d + t1) & 0xffffffff

        d = c

        c = b

        b = a

        a = (t1 + t2) & 0xffffffff

    end

    -- Add (mod 2^32) this chunk's hash to result so far:

    H[1] = (H[1] + a) & 0xffffffff

    H[2] = (H[2] + b) & 0xffffffff

    H[3] = (H[3] + c) & 0xffffffff

    H[4] = (H[4] + d) & 0xffffffff

    H[5] = (H[5] + e) & 0xffffffff

    H[6] = (H[6] + f) & 0xffffffff

    H[7] = (H[7] + g) & 0xffffffff

    H[8] = (H[8] + h) & 0xffffffff

end

local function str2hexa(s)

    local h = string.gsub(s, ".", function(c)

        return string.format("%02x", string.byte(c))

    end)

    return h

end

local function finalresult256(H)

    -- Produce the final hash value (big-endian):

    return str2hexa(string.pack("> I4 I4 I4 I4 I4 I4 I4 I4", H[1], H[2], H[3],

                                H[4], H[5], H[6], H[7], H[8]))

end

-- logic functions

function sha256.sha256(msg)

    msg = preproc(msg, #msg)

    local H = {};

    H = initH256(H);

    for i = 1, #msg, 64 do digestblock(msg, i, H) end

    return finalresult256(H)

end

return sha256

## 比特币区块链的区块结构

SHA256在比特币区块链中主要的作用是用于保证区块链的不可篡改性。首先要对比特币区块链中区块的结构做一个描述。

### 比特币区块的总体结构

|  |  |  |
| --- | --- | --- |
| 字段 | 大小 | 描述 |
| 区块大小 | 4 bytes | 区块的大小 |
| 区块头 | 80 bytes | 区块的关键字段信息 |
| 交易计数器 | 1-9 byes | 区块中包含交易的数量 |
| 交易列表 | 可变大小 | 交易数据 |

### 区块头的结构

|  |  |  |
| --- | --- | --- |
| 字段 | 大小 | 描述 |
| Version | 4 bytes | 区块版本 |
| Previous Hash | 32 bytes | 前一个区块的区块头Hash值 |
| Merkle Tree Root Hash | 32 bytes | 区块中所有交易创建的Hash树的根节点的Hash值 |
| Time Stamp | 4 byes | 区块创建的时间 |
| Difficulty | 4 byes | 区块的工作量证明难度目标 |
| Nonce | 4 bytes | 工作量证明算法的扰动输入参数 |

## 交易列表的结构

区块中的交易列表，实际上是区块中交易的Hash列表。类似下面这样：

文本

描述已自动生成

### 哈希算法在比特币区块链中的作用

比特币区块链的这种设计需求一种可靠的哈希算法，SHA256就是一种这样的算法，它满足以下特性：

* 对于任意长度的输入，SHA256会将输入数据转换为固定长度的输出；
* SHA256算法的输出是无规律的，输入数据中任何一个bit的改变都会导致结果输出完全不同；
* SHA256算法的碰撞率是极低的，因为SHA256算法的输出位数有256 bits，所以穷举出两个输出相同的输入值的难度是很高的。

### SHA256保障交易数据的不可篡改性

比如比特币区块头中的Merkle Tree Root Hash字段就为比特币区块中所有的交易建立了一个Merkle Tree，之后自底向上的计算出了根节点的Hash值，如果一个区块链中的一个区块的两个备份的Merkle Tree Root Hash是相同的，那么就可以认为这两个区块中记录的交易记录是相同的。

### SHA256 保障区块的不可篡改性

区块本身用一个Block Hash值来标识，但是区块自身的哈希值并没有记录在区块头，而是通过区块头来计算得出，上一个区块可以通过区块头记录的Previous Hash字段来进行访问。这样每个区块的Previous Hash都指向上一个区块，这些区块串联起来就形成了区块链，第一个区块的Previous Hash值被设置为0。因此如果一个攻击者篡改了区块中的交易数据，那么Merkle Tree Root Hash的值就会被改变，进而导致Block Hash的值发生变化，下一个区块指向它的链接就中断了。因此攻击者必须重新计算这个区块的Block Hash的值，并将之后所有区块的都重新计算和伪造一遍才能修改整个区块链，所以理论上这样的攻击在目前的计算能力下是不可能实现的。

## 实验证明SHA256发挥的作用

设计以下的交易列表生成的Merkle Tree：

图示

描述已自动生成

计算更改交易内容或者交易顺序之后的Merkle Tree Root Hash的值：

local sha = require("homework1")

-- normal treade

local a1 = sha.sha256("trade1")

local a2 = sha.sha256("trade2")

local a3 = sha.sha256("trade3")

local a4 = sha.sha256("trade4")

local b1 = sha.sha256(a1 .. a2)

local b2 = sha.sha256(a3 .. a4)

local merkle = sha.sha256(b1 .. b2)

-- output: 6aafa9d924a1782b638449c6f3fc3283eb44bf28579512b00999e0f100d3674b

print(merkle)

-- edit trade value

a1 = sha.sha256("trade1")

a2 = sha.sha256("faketrade")

a3 = sha.sha256("trade3")

a4 = sha.sha256("trade4")

b1 = sha.sha256(a1 .. a2)

b2 = sha.sha256(a3 .. a4)

merkle = sha.sha256(b1 .. b2)

-- output: 8570c33becafdb08b7462101b257122e3f763b682a9dfcf8143fa9f3a5a421a1

print(merkle)

-- edit trade order

a1 = sha.sha256("trade1")

a2 = sha.sha256("trade3")

a3 = sha.sha256("trade2")

a4 = sha.sha256("trade4")

b1 = sha.sha256(a1 .. a2)

b2 = sha.sha256(a3 .. a4)

merkle = sha.sha256(b1 .. b2)

-- output: 6132794d6f8c62254322259b52b2543db83ec0509e0c3110ff8f91403955d649

print(merkle)

从上述实验可以看出SHA256可以有效保障比特币区块链的不可篡改性。