# SHA256算法实验报告

### SHA256算法描述

1. 附加填充比特。  
    对报文进行填充使报文长度与448 模512 同余（长度=448 mod 512），填充的比特数范围是1 到512，填充比特串的最高位为1，其余位为0。
2. 附加长度值。将用64-bit 表示的初始报文（填充前）的位长度附加在步骤1 的结果后（低位字节优先）。
3. 初始化缓存。使用一个256-bit 的缓存来存放该散列函数的中间及最终结果。
4. 处理512-bit（16 个字）报文分组序列。该算法使用了六种基本逻辑函数，由64步迭代运算组成。每步都以256-bit 缓存值ABCDEFGH 为输入，然后更新缓存内容。

### 基础实验结果分析

输入：我喜欢比特币

输出：EBFD09F276A23997171DFF058558CB09348C4C9C6FFFD87178E90E0F5D49B88A

输入：我喜欢比特币

输出：EAFFA1BA1C4E5FB72CF0948F983C9AD39A42C2C60A7C313E9ABA998A05AB595B

结论：SHA256算法很适合比特币加密原因有下：  
 每个值都有对应唯一的哈希值，无论是谁来算只要输入一样，无论多少次结果都是一样的。

稍微变动一点结果便大不相同。

结果是单映射的。

### SHA256在PoW中的作用

例如，我们假设区块信息为"Hello！"，我们给出的工作量要求是，可以在这个字符串后面添加一个叫做nonce的整数值，对变更后（添加nonce)的字符串进行SHA256哈希运算，如果得到的哈希结果（以16进制的形式表示）是以"0000"开头的，则验证通过。

我们将”Hello!”的变更为"Hello!+数字 "，整数值取1到50，也就是说，将输入变成一个由1000个值组成的数组："Hello!1、Hello!2……Hello!50"。然后对数组中的每一个输入依次进行上面例子中要求的工作量证明——找到前导为4个0的哈希散列。

根据哈希函数估算，我们大概要进行2^16计算，而实际结果是平均计算次数为66765，符合统计学的规律。

### SHA256如何用于区块数据锁定

在区块链中，假设用msgtx表示一次交易，这个交易结构体里，有txin字段，完整的定义了msgtx的下一个块的信息。这个结构体里，有个previousoutpoint表示这笔输入的来源地址；每一笔交易信息，会先封到一个block结构体里，然后经过验证存到数据库里；每一笔交易都会有一个previousoutpoint，又由于哈希函数具有单映射性，所以previousoutpoint几乎是唯一确定的，通过这个可以溯源至最初的一个区块，很难出现篡改的可能