链滴

大白话讲解 SHA256

作者：Doss

原文链接：https://ld246.com/article/1660940043935

来源网站：链滴

大白话讲解SHA256

文末有js实现和c语言实现的代码。

SHA256简介

SHA-2，名称来自于安全散列算法2（英语：Secure Hash Algorithm 2）的缩写，一种密码散列函算法标准。SHA256属于SHA-2，SHA-2总共包括SHA-224、SHA-256、SHA-384、SHA-512、SH-512/224、SHA-512/256这六个标准。

SHA-256和SHA-512是很新的散列函数，前者以定义一个word为32位，后者则定义一个word为64。它们分别使用了不同的偏移量，或用不同的常量，然而，实际上二者结构是相同的，只在循环执行次数上有所差异。SHA-224以及SHA-384则是前述二种散列函数的截短版，利用不同的初始值做计。

"前者以定义一个word为32位",这里是说SHA256最小运算单位为32位。实现SHA256一定要注意用32位无符号数，溢出时与2^32取余(也就是去除最左边的溢出的位数)

算法

总共有以下几个处理过程。

1. 设置初始化值。
2. 预处理数据。
3. 将数据分段称之为消息块，每段为512位。
4. 遍历消息块，计算出哈希值，最后一个消息块的哈希值为最终的SHA256结果。
5. 将计算得出的8个值，转换为16进制编码格式的字符串，拼起来就是最终的SHA256摘要结果。

简要描述

设置初始化值。

首先有8个初始值和64个常量要预备，这些值怎么得来的呢？取自然数中的前8个质数的平方根小数部的二进制编码前32位数值，前64个质数的立方根小数部分的前32位数值。前8个质数的平方根小数部的二进制编码前32位用来当作计算的初始值，前64个质数的立方根小数部分的二进制编码前32位数用来作遍历消息块每轮计算的一部分。

可能这里就有小朋友好奇了：随便选个数不好吗？为什么非要质数嘞？来咱们看看维基百科对于质数解释：

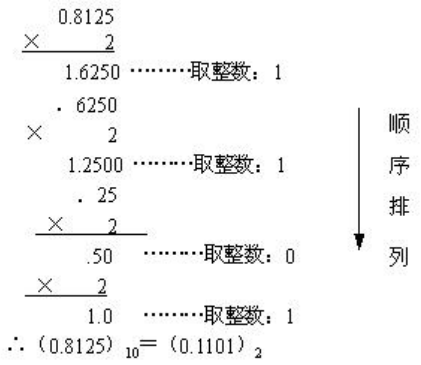
质数（Prime number），又称素数，指在大于1的自然数中，除了1和该数自身外，无法被其他自然整除的数。

什么意思？你看，只能被1和自身整除。不会再有别的因数，这样就能减少防碰撞性（防碰撞性：两不同的输入数据经过hash运算，hash值不应该相同）。

为了更直观一些，来试着算出一个初始值。(前8个质数：2，3，5，7，11，13，17，19)

- 2的平方根等于1.414213562373095048801688724209 7
- 保留小数部分414213562373095048801688724209 7
- 计算小数部分的二进制编码保留前32位。得0110 1010 0000 1001 1110 0110 0110 0111（2进） = 0x6A09E667（16进制）

十进制小数转换成二进制小数采用"乘2取整，顺序排列"法



用2乘十进制小数，可以得到积，将积的整数部分取出，再用2乘余下的小数 部分，又得到一个积，将积的整数部分取出，如此进行，直到积中的小数部分为零，或者达到所要求的精度为止。然后把取的整数部分按顺序排列起来，先取的整数作为二进制小数的高位有效位，后取的整数作为低位有效位例如把（0.8125）转换为二进制小数。

前8个质数平方根小数部分的32位。

h0 := 0x6a09e667
h1 := 0xbb67ae85
h2 := 0x3c6ef372
h3 := 0xa54ff53a
h4 := 0x510e527f
h5 := 0x9b05688c
h6 := 0x1f83d9ab
h7 := 0x5be0cd19

遍历消息块每轮用到的64个常数

k[0..63] :=
0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xa1c5ed5,
0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0c19bf174,
0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x7f988da,
0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x4292967,
0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 092722c85,
0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x06aa070,
0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x82e6ff3,
0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc6178f2

预处理数据

要点：

1. 必须进行补码操作(先补1个1，剩余的全补0)。
2. 最终的数据长度必须是512的倍数。

简单来说，是因为后面要把原始报文处理为每个为512位长的消息块，所以要把原始报文处理为512倍数长度的数据。但这里有个点要注意，最后一个消息块的最后64位我们要用来保存原始报文的bit长度，所以最后一个消息块补到448长度即可。但就算最后一个消息块长度正好是448位或者超过也补码(额外补512位，然后在此基础上把最后的消息块补位至448位)，也就是必须进行补码操作。

我们以字符串“abc”举例，ASCII码值为97，98，99，二进制编码长度为24位。二进制编码为0110 001 0110 0010 0110 0011。

补1，长度等于25。

0110 0001 0110 0010 0110 0011 1

补0，使长度等于448,为了简洁在下面就采用16进制编码显示。1位16进制编码等于4位2进制（0xff = 1111 1111，0x00 = 0000 0000）。

61626380 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000

附加原始报文长度信息。(长度为24 = 0001 1000 = 0x18)

SHA256用一个64位的数据来表示原始消息的长度。因此，通过SHA256计算的消息长度必须要小于264。

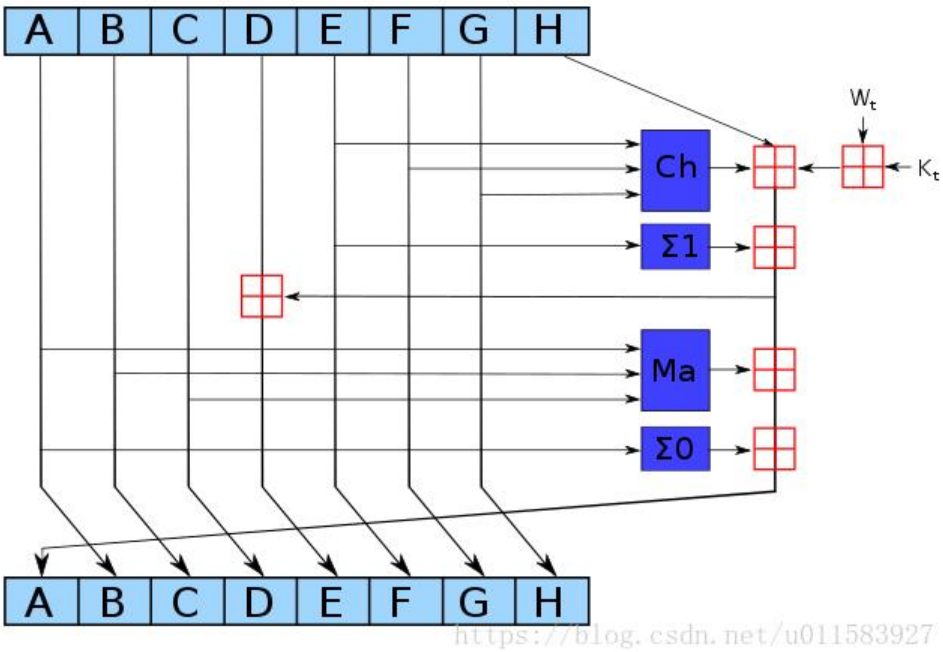
长度信息的编码方式为64-bit big-endian integer(大端序)。

61626380 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000018

数据分段

把处理后的数据分成若干块，每块512位。

主要计算逻辑



以下描述当中所使用到的标记如下：

- \oplus: 按位异或。
- \wedge: 按位与。
- \neg: 按位补（取反）。
- +: 相加以后对2^{32}求余，相当于二进制编码下超出32位直接删除最左边的位数保留32位结果 这里我理解为放弃溢出的数值。
- S^{n}: 环右移n位。
- R^{n}: 右移n位。

计算所需逻辑函数定义：

- Ch(x,y,z)=(x\wedge y)\oplus(\neg x\wedge z)
- M_{aj}(x,y,z)=(x\wedge y)\oplus(x\wedge z)\oplus (y \wedge z)
- \Sigma_{0}(x) = S^{2}(x)\oplus S^{13}\oplus S^{22}(x)
- \Sigma_{1}(x) = S^{6}(x)\oplus S^{11}\oplus S^{25}(x)
- \sigma_{0}(x) = S^{7}(x)\oplus S^{18}\oplus R^{3}(x)
- \sigma_{1}(x) = S^{17}(x)\oplus S^{19}\oplus R^{10}(x)

看不懂公式没关系，下面跟着一步一步来你就能明白。

遍历消息块：

1. 建立一个64个word的缓存数组，称之为w[]。
2. 将当前消息块分成16个word, 每个word是32位，存入w[]。
3. 以这16个word为基础计算剩余的48个word，存入w[]。

计算方式：

- For\:t = 16 \:\rightarrow\: 63
    - W(t)=\sigma_{1}(W_{t-2})+W_{t-7}+\sigma_{0}(W_{t-15})+W_{t-16}

```
// C语言示例
#define rightrotate(w, n) ((w >> n) | (w) << (32-(n))) // 循环右移宏

for (int i = 16; i < 64; i++) {
  // σ0(x)=S7(x)⊕S18(x)⊕R3(x)
  uint32_t s0 = rightrotate(w[i - 15], 7) ^ rightrotate(w[i - 15], 18) ^ (w[i - 15] >> 3);
  // σ1(x)=S17(x)⊕S19(x)⊕R10(x)
  uint32_t s1 = rightrotate(w[i - 2], 17) ^ rightrotate(w[i - 2], 19) ^ (w[i - 2] >> 10);
  // Wj = σ1 + Wj-7 + σ0 + Wj-16
  w[i] = s1 + w[i - 7] + s0 + w[i - 16];
}
```

4. 建立8个缓存变量（a、b、c、d、e、f、g、h），其值为上一个消息块的这8个缓存变量，若当前息块为第一个消息块，则将其值设为开头所提的初始化值（h0, h1, h2, h3, h4, h5, h6, h7）。

```
a := h0
b := h1
c := h2
d := h3
e := h4
f := h5
g := h6
h := h7
```

5. 遍历w[]，每次都已上一个遍历的结果来以一定规则计算本次遍历得出结果，赋值给8个缓存变量。

应用上面提到的函数来更新a，b，c，d，e，f，g，h。

- For\: j = 0\:\rightarrow\:63
    - T_{1}\leftarrow h\,+\Sigma_{1}(e)+Ch(e,f,g)+K_{j}+W_{j}
    - T_{2}\leftarrow \Sigma_{0}(a)+M_{aj}(a,b,c)
    - h\leftarrow g
    - g\leftarrow f
    - f\leftarrow e
    - e\leftarrow d+T_{1}
    - d\leftarrow c
    - c\leftarrow b
    - b\leftarrow a
    - a\leftarrow T_{1}+T_{2}
- 将这一轮计算得出的值追加到h0, h1, h2, h3, h4, h5, h6, h7。

```
h0 := a
h1 := b
h2 := c
h3 := d
h4 := e
```

h5 := f
h6 := g
h7 := h

拼接结果。

最后一个消息块的最后一轮遍历的结果被保存在h0, h1, h2, h3, h4, h5, h6, h7。

将这八个word转换为16进制编码，再合起来，就是最终的摘要结果。

digest := hash := h0 append h1 append h2 append h3 append h4 append h5 append h6 app
nd h7

补充

大端和小端（Big endian and Little endian）

对于整型、长整型等数据类型，都存在字节排列的高低位顺序问题。

Big endian 认为第一个字节是最高位字节（按照从低地址到高地址的顺序存放数据的高位字节到低字节）

而 Little endian 则相反，它认为第一个字节是最低位字节（按照从低地址到高地址的顺序存放据的位字节到高位字节）。

代码仓库地址

https://github.com/Parker-ad/SHA256-implement