1. 代码实现

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include<stdint.h>

#define maxlen 1000000 //可转化的消息最大长度

typedef unsigned int uint;

typedef unsigned char ubyte;

typedef unsigned long long ull;

char instr[maxlen];

const uint 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

};

inline uint rrot(uint val, int pos) {

pos %= 32;

return ( val >> pos ) | ( val << (32 - pos) );

}

void sha\_256(char \*digest, char \*str) {

// 提取字符跟预处理

uint orilen = strlen(str); // 以byte计算长度

uint chunks\_count = (orilen+9)/64; // 块数

if( (orilen+9)%64!=0 ) ++chunks\_count; // 块数

ull total\_count = orilen \* 8; // 总字节数

uint \*filldata = new uint[ chunks\_count\*16 ]; // 将要把数据填充到这一块内存

memset(filldata, 0, sizeof(uint)\*chunks\_count\*16); // 初始化

ubyte \*pdata = (ubyte \*)filldata;

uint idx;

for (idx = 0; idx < orilen; ++idx) {

pdata[idx+3-idx%4-idx%4] = str[idx]; // idx:[0..11] --> idx+3-idx%4-idx%4: [3,2,1,0, 7,6,5,4, 11,10,9,8]

}

pdata[idx+3-idx%4-idx%4] = 0x80; // 追加1跟7个0

++idx;

// 追加流的长度

ubyte \*pbyte = (ubyte \*)&total\_count;

for (uint i = 0, j = 64\*chunks\_count - 4; i < 8; ++i) {

if (i==4) j -= 8;

pdata[j + i] = pbyte[i];

}

// 对每块进行轮换迭代

uint H[8],a,b,c,d,e,f,g,h, w[80], s[2];

H[0] = 0x6a09e667;

H[1] = 0xbb67ae85;

H[2] = 0x3c6ef372;

H[3] = 0xa54ff53a;

H[4] = 0x510e527f;

H[5] = 0x9b05688c;

H[6] = 0x1f83d9ab;

H[7] = 0x5be0cd19;

for (uint i = 0; i < chunks\_count\*16; i += 16) {

uint \*puint = &filldata[i];

for (uint j = 0; j < 16; ++j) {

w[j] = puint[j];

}

for (uint j = 16; j < 64; ++j) {

s[0] = ( rrot(w[j - 15], 7) ) ^ ( rrot(w[j - 15], 18) ) ^ ( w[j - 15]>>3 );

s[1] = ( rrot(w[j - 2], 17) ) ^ ( rrot(w[j - 2], 19) ) ^ ( w[j - 2]>>10 );

w[j] = w[j - 16] + s[0] + w[j - 7] + s[1];

}

a = H[0], b = H[1], c = H[2], d = H[3], e = H[4], f = H[5], g = H[6], h = H[7];

for (uint j = 0; j < 64; ++j) {

uint maj, t[3], ch;

s[0] = ( rrot(a,2) ) ^ (rrot(a,13)) ^ (rrot(a,22));

maj = (a & b) ^ (a & c) ^ (b & c);

t[2] = s[0] + maj;

s[1] = rrot(e, 6) ^ rrot(e, 11) ^ rrot(e, 25);

ch = (e & f) ^ ((~e) & g);

t[1] = h + s[1] + ch + k[j] + w[j];

h = g, g = f, f = e, e = d + t[1], d = c, c = b, b = a, a = t[1] + t[2];

}

H[0] += a, H[1] += b, H[2] += c, H[3] += d, H[4] += e, H[5] += f, H[6] += g, H[7] += h;

}

for (int i = 0; i < 8; ++i) {

sprintf(digest+i\*8, "%08x", H[i]);

}

delete[] filldata;

}

int main () {

puts("请输入要转化的内容:");

char digest[65];

scanf("%s", instr);

sha\_256(digest, instr);

puts(digest);

return 0;

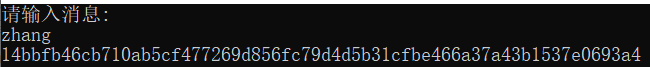
}

1. 实验数据

测试用例一：纯字母，较短字符串

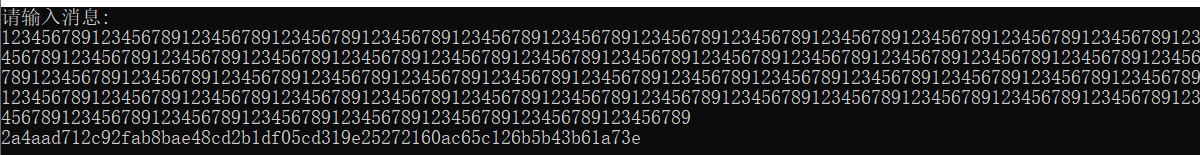
输入要加密的字符串：zhang，输出结果为

14bbfb46cb710ab5cf477269d856fc79d4d5b31cfbe466a37a43b1537e0693a4



测试用例二：纯数字，较长字符串

输入要加密的字符串：60组123456789，输出结果为： 2a4aad712c92fab8bae48cd2b1df05cd319e25272160ac65c126b5b43b61a73e



1. SHA256用于区块链的作用

由于SHA256是基于哈希进行的链接，且对于SHA256加密算法，对于相同的输入，会产生相同的输出。基于哈希值进行的链接可以保证区块链中的中间数据无法被篡改或删除，因为一旦中间节点的数据被更改会导致之后链条的数据都会异常，进而被发现。