作业一

游宁远 3180105261

### 一、实验目的

提交一个SHA256的代码实现，并用实验数据说明SHA256在比特币区块链中发挥的作用。

### 二、源代码与分析

由于该程序主要根据[1]中的算法描述进行实现，此处不进行重复的算法分析。代码解释见注释。

1. #include<iostream>
2. #include<string>
3. #include<cstring>
4. #include<cstdio>
5. #include<set>
6. #include<ctime>
7. #include<cstring>
8. #include<iomanip>
9. **using** **namespace** std;
10. **class** SHA256{
11. **const** unsigned **int** table[64]={
12. 0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5,
13. 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
14. 0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,
15. 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
16. 0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc,
17. 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
18. 0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7,
19. 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
20. 0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,
21. 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
22. 0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3,
23. 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
24. 0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5,
25. 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
26. 0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
27. 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
28. };
29. unsigned **int** h[8]={0x6a09e667,0xbb67ae85,0x3c6ef372,0xa54ff53a,0x510e527f,0x9b05688c,0x1f83d9ab,0x5be0cd19};
30. **void** init(){//初始化
31. h[0] = 0x6a09e667;
32. h[1] = 0xbb67ae85;
33. h[2] = 0x3c6ef372;
34. h[3] = 0xa54ff53a;
35. h[4] = 0x510e527f;
36. h[5] = 0x9b05688c;
37. h[6] = 0x1f83d9ab;
38. h[7] = 0x5be0cd19;
39. **return**;
40. }
41. unsigned **int** Ror(unsigned **int** src,**int** x){//循环右移位
42. **return** (src>>x) | (src<<(32-x));
43. }
44. **void** calc(unsigned **int** a[]){//以指针形式传入512个bit进行计算
45. **char** \*s = (**char**\*)a;
47. **for**(**int** i=0;i<64;i+=4){//为了适应小端存储规则，进行换位
48. swap(s[i],s[i+3]);
49. swap(s[i+1],s[i+2]);
50. }
52. unsigned **int** w[64];
53. **for**(**int** i=0;i<16;i++){//复制前16个word
54. w[i]=a[i];
55. }
56. **for**(**int** i=16;i<64;i++){//生成后48个word
57. w[i] = ((Ror(w[i-2],17) ^ Ror(w[i-2],19) ^ (w[i-2]>>10))) +
58. w[i-7] +
59. (Ror(w[i-15],7) ^ Ror(w[i-15],18) ^ (w[i-15]>>3)) +
60. w[i-16];
61. }
62. unsigned **int** A,B,C,D,E,F,G,H;
63. A = h[0];
64. B = h[1];
65. C = h[2];
66. D = h[3];
67. E = h[4];
68. F = h[5];
69. G = h[6];
70. H = h[7];
71. **for**(**int** i=0;i<64;i++){//64次循环计算
72. //根据规则计算8组32位值
73. unsigned **int** Ch = (E&F)^((~E)&G);
74. unsigned **int** Ma = (A&B)^(A&C)^(B&C);
75. unsigned **int** sigma0 = Ror(A,2)^Ror(A,13)^Ror(A,22);
76. unsigned **int** sigma1 = Ror(E,6)^Ror(E,11)^Ror(E,25);
77. unsigned **int** tmpH = H;
78. H = G;
79. G = F;
80. F = E;
81. E = D + Ch + w[i] + table[i] +sigma1 + tmpH;
82. D = C;
83. C = B;
84. B = A;
85. A = tmpH + w[i] + table[i] + Ch + sigma1 + Ma + sigma0;//自然溢出取模
86. }
87. h[0] += A;
88. h[1] += B;
89. h[2] += C;
90. h[3] += D;
91. h[4] += E;
92. h[5] += F;
93. h[6] += G;
94. h[7] += H;
95. **return**;
96. }
97. **public**:
98. **void** Encode(string str){
99. init();
100. unsigned **long** **long** len = str.length();
101. //padding
102. //填充到 (448/8==56)个byte
103. **int** padlen = ((56 - str.length()%64)+64)%64;
104. **if**(padlen==0) padlen = 64;
105. str.append(1,128);
106. **for**(**int** i=1;i<padlen;i++){
107. str.append(1,0);
108. }
109. //additive
110. //将表示字符串原长度的64位字填充到待加密串的末尾
111. **for**(**int** i=1;i<=8;i++){
112. unsigned **char** tmp = (unsigned **char**)((len\*8) >> (64-i\*8));
113. str.append(1,tmp);
114. }
115. //calc
116. len = str.length();
117. **for**(**int** i=0;i<len;i+=64){//每512bit为一段，进行计算
118. calc((unsigned **int**\*)(str.c\_str()+i));
119. }
120. **return**;
121. }
122. **void** Output(){//输出
123. **for**(**int** i=0;i<8;i++){
124. cout<<setfill('0')<<setw(8)<<hex<<h[i];
125. }
126. cout<<endl;
127. **return**;
128. }
129. string GetRes(){//输出为字符串
130. string res="";
131. **char** s[10];
132. **for**(**int** i=0;i<8;i++){
133. sprintf(s,"%08x",h[i]);
134. res+=s;
135. }
136. **return** res;
137. }
138. };
139. **int** main(){
140. string input;
141. cin>>input;//输入待加密字符串(不含空格)
142. SHA256 res;
143. res.Encode(input);//加密
144. res.Output();//输出
145. /\*
146. srand(time(0));
147. set<unsigned int>num;
148. set<string>st;
149. st.clear();
150. for(int i=0;i<100000;i++){
151. unsigned int x = rand()\*10000 + rand()\*100 + rand();
152. //      unsigned int x = 1000000000 + i;
153. if(num.count(x)>0)continue;//使用不重复的数字进行加密
154. num.insert(x);
155. char ch[50];
156. sprintf(ch,"%d",x);//生成的随机数转为字符串
157. string ss(ch);
158. ss += "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
159. SHA256 res;
160. res.Encode(ss);
161. ss = res.GetRes();
162. if(st.count(ss)>0){//若检测到碰撞，终止
163. cout<<"Collision"<<endl;
164. break;
165. }
166. st.insert(ss);
167. cout<<ss<<endl;
168. }
169. \*/
170. **return** 0;
171. }

### 三、结果与分析

1. **正确性验证**

随意选取了一些字符串进行加密，结果与已有工具

(<http://www.ttmd5.com/hash.php?type=9> 网站提供的在线SHA256加密工具)

比对，发现结果正确。

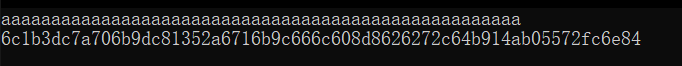
对任意长度的输入数据，SHA256可以生成长为64个byte的不冲突的输出。

（截图第一行为输入字符串，第二行为加密结果）











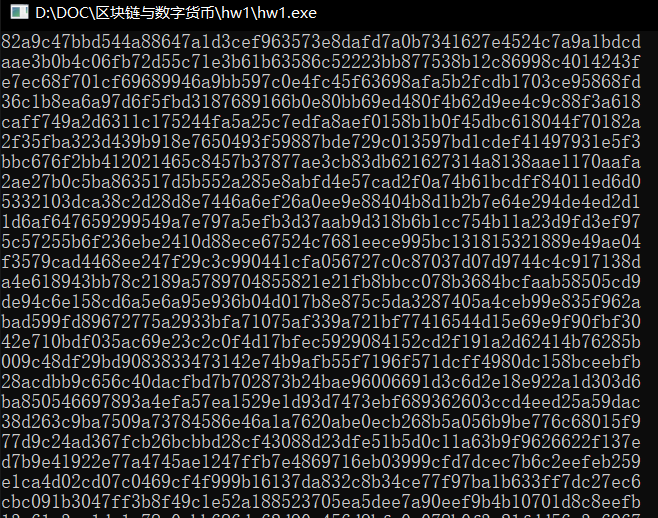
1. **随机数碰撞**

使用以下代码生成随机数并进行加密，判断加密结果是否碰撞：



未发生碰撞。虽然1e5规模的输入数据对于SHA256的hash值域来说确实过小，但也能使我们对其性能有初步的感受。

部分结果：



1. **规律性输入：**

使用以下代码生成有规律的字符串并加密：



发现对于规律性变化的输入数据，输出的加密结果无直观规律可循。

部分结果：



1. **总结**

根据以上结果， SHA256算法在比特币区块链中发挥的重要作用可见一斑：

(1)SHA256的hash值域大，难以产生碰撞，且输出结果定长。因此将比特币区块header的SHA256加密结果作为比特币区块的唯一标识符，既不用担心碰撞，又易于用数据结构存储和表示。

(2)从1中数据局部差异导致的结果差异，3中规律输入数据的无规律输出，可以看出对原文的微小改动会导致SHA256密文无规律改变，且加密结果难以“主动制造”。因此，在比特币的区块链中，对任意区块的任意修改都会导致其后续区块的sha256值发生巨大变化。这样的性质，使得区块链上的信息可以完全公开而无法被恶意篡改、伪造，保证了交易记录的安全性。

(3)另外根据测试2和3的输出结果可以发现，加密结果串的前导零出现得没有规律，利于“矿工”根据算力进行公平竞争。

(4)分析代码结构，SHA256算法的时间和空间复杂度均为O(n)，使得对于任何规模的数据都能以可接受的速度求解。有利于比特币用户快速校验所收到区块链的可靠性。

### 四、参考资料

[1] SHA256算法原理详解,

https://blog.csdn.net/u011583927/article/details/80905740