

**本科实验报告**

|  |  |
| --- | --- |
| 课程名称： | 计算机网络基础 |
| 实验名称： | SHA256算法实践 |
| 姓 名： | 张天涵 |
| 学 院： | 计算机学院 |
| 系： | 计算机系 |
| 专 业： | 软件工程 |
| 学 号： | 3200105746 |
| 指导教师： | 杨小虎 |

2022年 9月 25 日

1. 实验过程
2. SHA的算法实现
   1. 输入的填充：

这一步属于输入的预处理，要对消息进行补码处理，我们目标要达到的结果是输入的最终的长度是512位的倍数。

假设消息的二进制编码长度为L位,首先在消息末尾补上一位"1", 然后再补上k个"0", 其中k为下列方程的最小非负整数。

这一步操作之后，我们需要在操作完成的字符串之后加上（此时是448位）64位的“原消息长度”二进制表示形式。



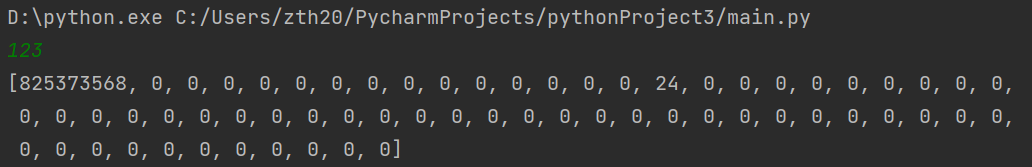
* 1. 内容分块

SHA256以512bits的数据块处理单元，每个512bits的数据会分为16份32bits的数据块，分别为M1，M2，M3,,,M15，之后我们还需要构造48个32bits的数据块，根据以下公式计算：

其中的运算符解释：

我们开辟一个W列表来存储字，一个元素是32bits转化为十进制的值，所以我们的列表总长是64，前16个位之前的数据分块，后面先初始化为0，后续直接带入计算并赋值即可。





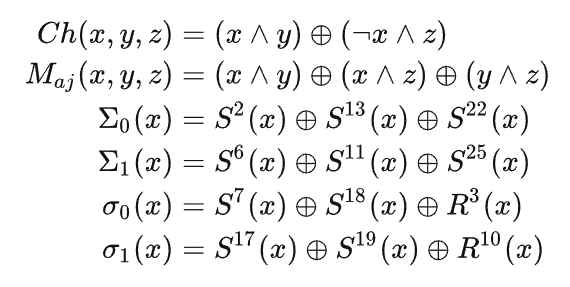
当我们输入123之后，可以看到初始的分块完成之后的64个模块的具体信息，至此，信息预处理已全部完成。

* 1. 循环计算

我们参考网上的sha256伪码介绍，大意就是把a,b,c,d,e,f,g初始化为hash常量（h0 := 0x6a09e667，h1 := 0xbb67ae85，h2 := 0x3c6ef372，h3 := 0xa54ff53a，h4 := 0x510e527f，h5 := 0x9b05688c，h6 := 0x1f83d9ab，h7 := 0x5be0cd19），连同W模块以及64个hash常数一起传入循环，每次进行计算完再重新循环，直到64次算完，其中数据不会超过32位，超过的需要压缩。



其中的逻辑运算方法如图所示



其中循环计算如下，计算64次，具体的计算过程即为伪代码的表述，详见附录代码。



在进行完计算之后，我们把结果与哈希所使用的常数相加，在拼接起来即可得到最终的SHA256结果。伪代码如下：



具体代码如下：



1. Nonce
   1. 初始值和位判断（nonce循环验证

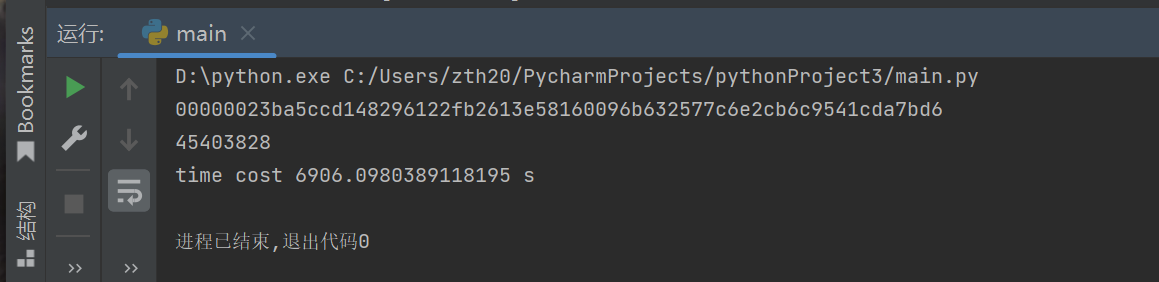
我们让nonce从0开始每次递增1，之后把nonce转化为str类型加在初始信息之后，作为输入的消息并进行sha256加密，之后对得到的256位信息ANS（一个64位16进制字符串）进行验证操作，ANS前n位为0，则意味着答案前4\*n位为0，如果之后需要判断不为4的倍数的位数为0，我们只需要把对应的那一个字转化为4位二进制数，按位来验证即可。

我们设置一个while循环，直到找到符合条件的一个nonce，否则会将程序一直执行下去。最后程序依次输出SHA256结果，nonce值，运行时间。

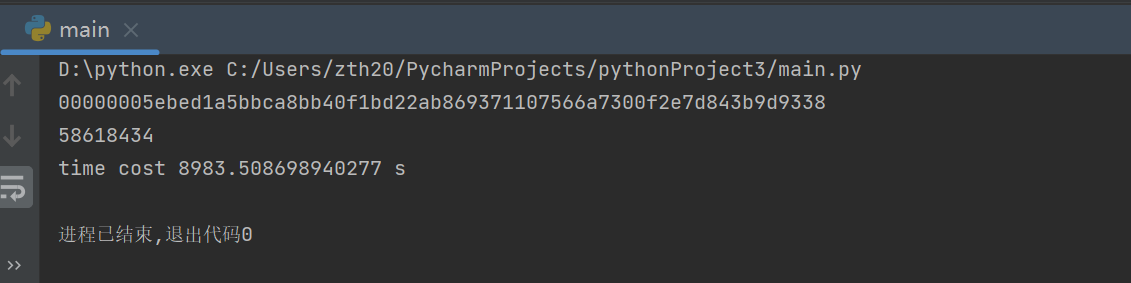


1. 实验结果

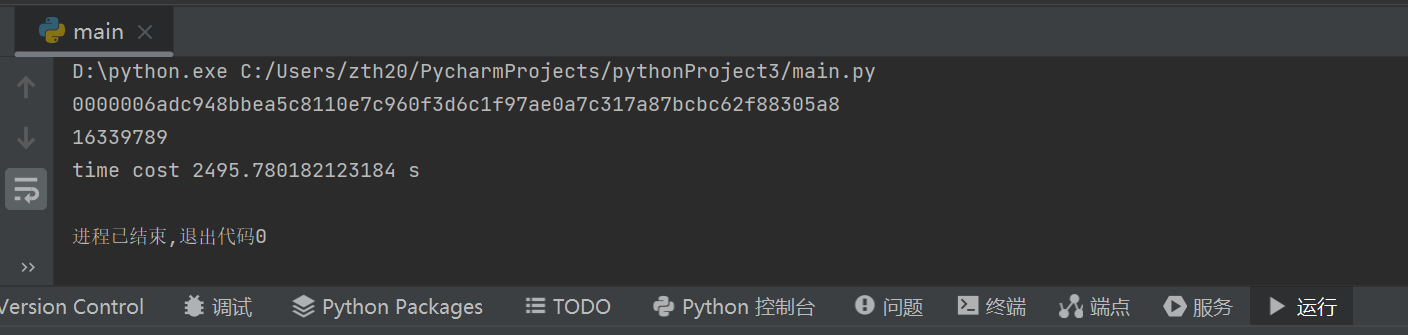
|  |  |
| --- | --- |
| 序号 | 1 |
| 前n位为0 | 26 |
| Nonce | 45403828 |
| 用时 | 6906s |



|  |  |
| --- | --- |
| 序号 | 2 |
| 前n位为0 | 29 |
| Nonce | 58618434 |
| 用时 | 8983s |
| 备注：因为这个29为为0是我在跑“前27位为0”时跑出来的，所以可认为27，28，29的结果都为此。 | |



|  |  |
| --- | --- |
| 序号 | 3 |
| 前n位为0 | 25 |
| Nonce | 16339789 |
| 用时 | 2495s |
| 备注：因为如上一个实验结果可知：29位以上的结果程序跑了一晚上还未出结果，所以我减少了几位bits，计算了24，25的结果。 | |



|  |  |
| --- | --- |
| 序号 | 4 |
| 前n位为0 | 24 |
| Nonce | 1546596 |
| 用时 | 2360s |

我忽略了python是解释性的语言，在程序进行的时候会比c++的速度慢很多，导致我的转换了一下bits的要求。

附录：源码





