

**网络空间安全创新创业实践**

**小组成员：李文杰 201900180054**

**Git账户名称： QingYun65**

**仓库地址：https://github.com/QingYun65?tab=repositories**

**项目清单：**

一、SHA256-Benchmark

二、Implement the naïve birthday attack of reduced SM3

三、Implement length extension attack for MD5

四、Optimize SM3 implementation (software)

五、SM3 AESNI 实现

六、SM4 S盒复合域实现方法

七、Impl Merkle Tree following RFC6962

八、Find a key with hash value “sdu\_cst\_20220610” under a message composed of your name followed by your student ID. For example, “San Zhan 202000460001”.

九、Find a 64-byte message under some k fulfilling that their hash value is symmetrical.

十、Forge a signature to pretend that you are Satoshi



**一、SHA256-Benchmark**

Git仓库地址：https://github.com/QingYun65/SHA256-Benchmark.git

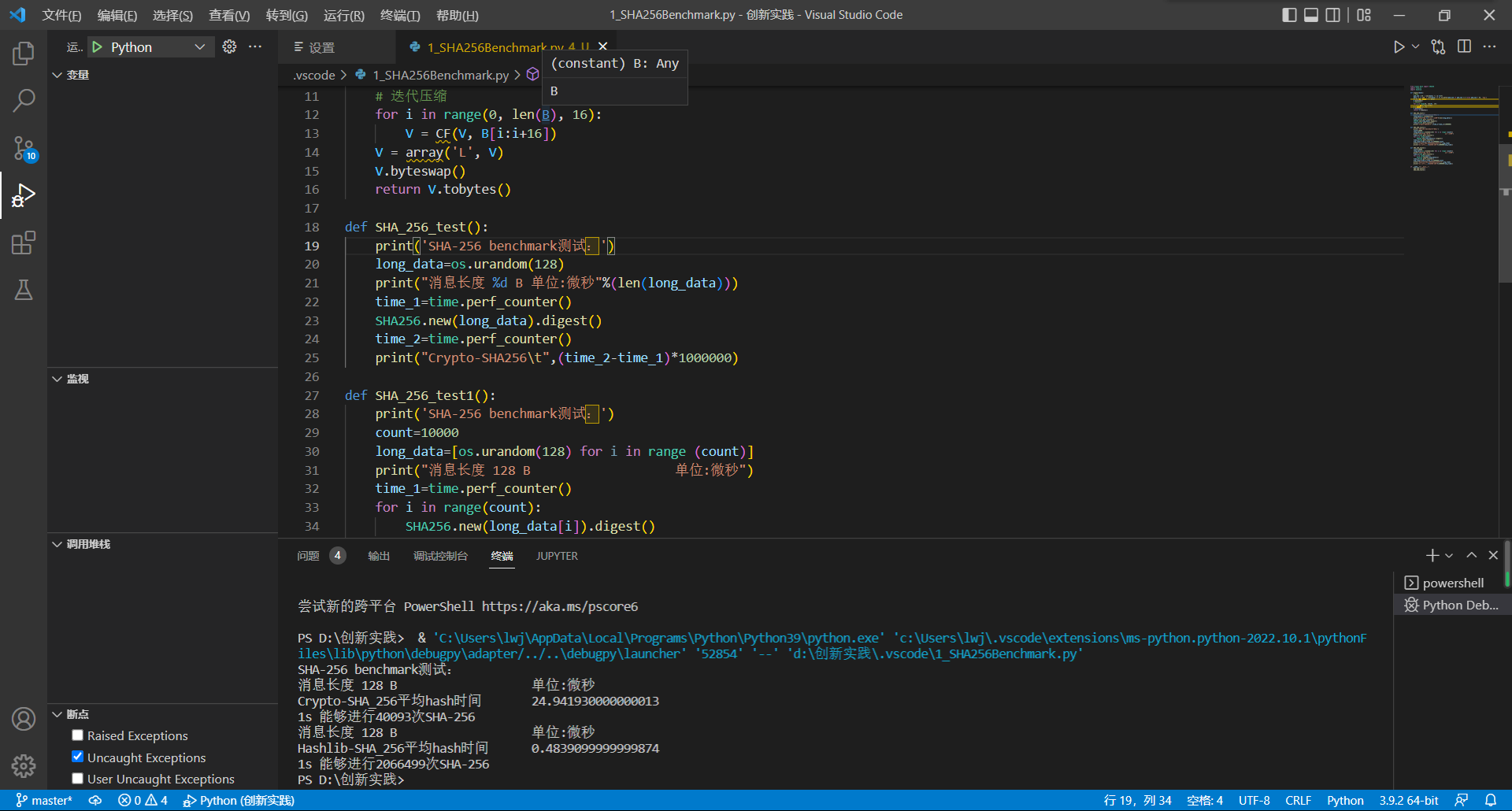
1、项目简介

通过os.urandom()随机生成10000个128B的数据，计算其平均Hash时间，通过计算可以得到1s内Crypto.Hash和hashlib中SHA256加密次数。其中，hashlib中SHA256每秒加密次数可以达到百万次。

2、运行指导

修改SHA\_256\_test1()、SHA\_256\_test2()中count数值可以指定生成测试数据的个数。

3、代码运行过程截图



**二、Implement the naïve birthday attack of reduced SM3**

Git仓库地址：<https://github.com/QingYun65/Birthday-attack-on-SM3.git>

1、项目简介

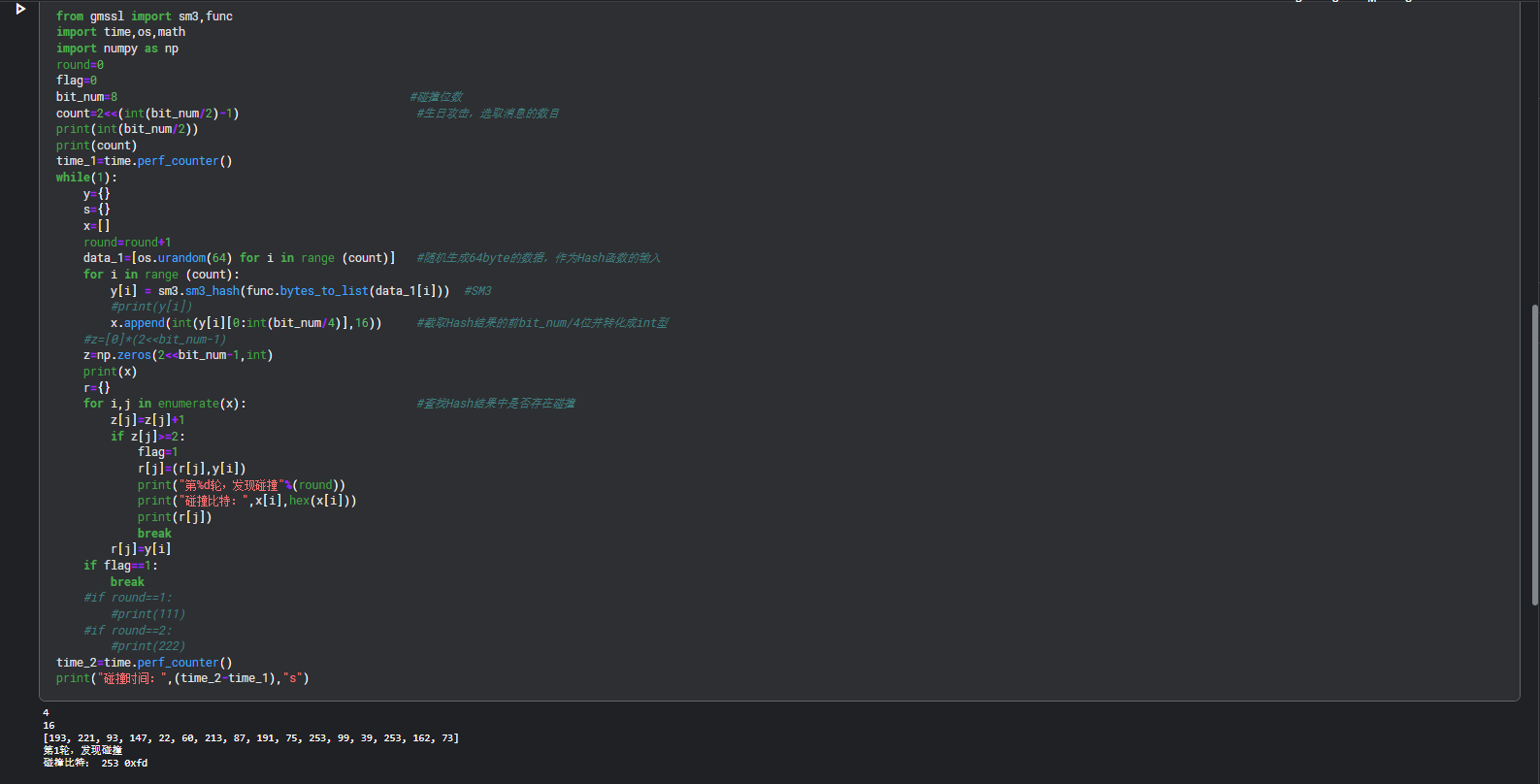
生日攻击可以比穷举更快的找到碰撞，对于n-bit的Hash函数（N=2^n），只要准备t≈1.177sqrt(2^n)个消息，就能以至少p=1/2的概率成功找到一对碰撞，这里对SM3采取生日攻击，当然，并不是结果全部256bit，而仅仅是前8bit、15bit、32bit。

2、运行指导

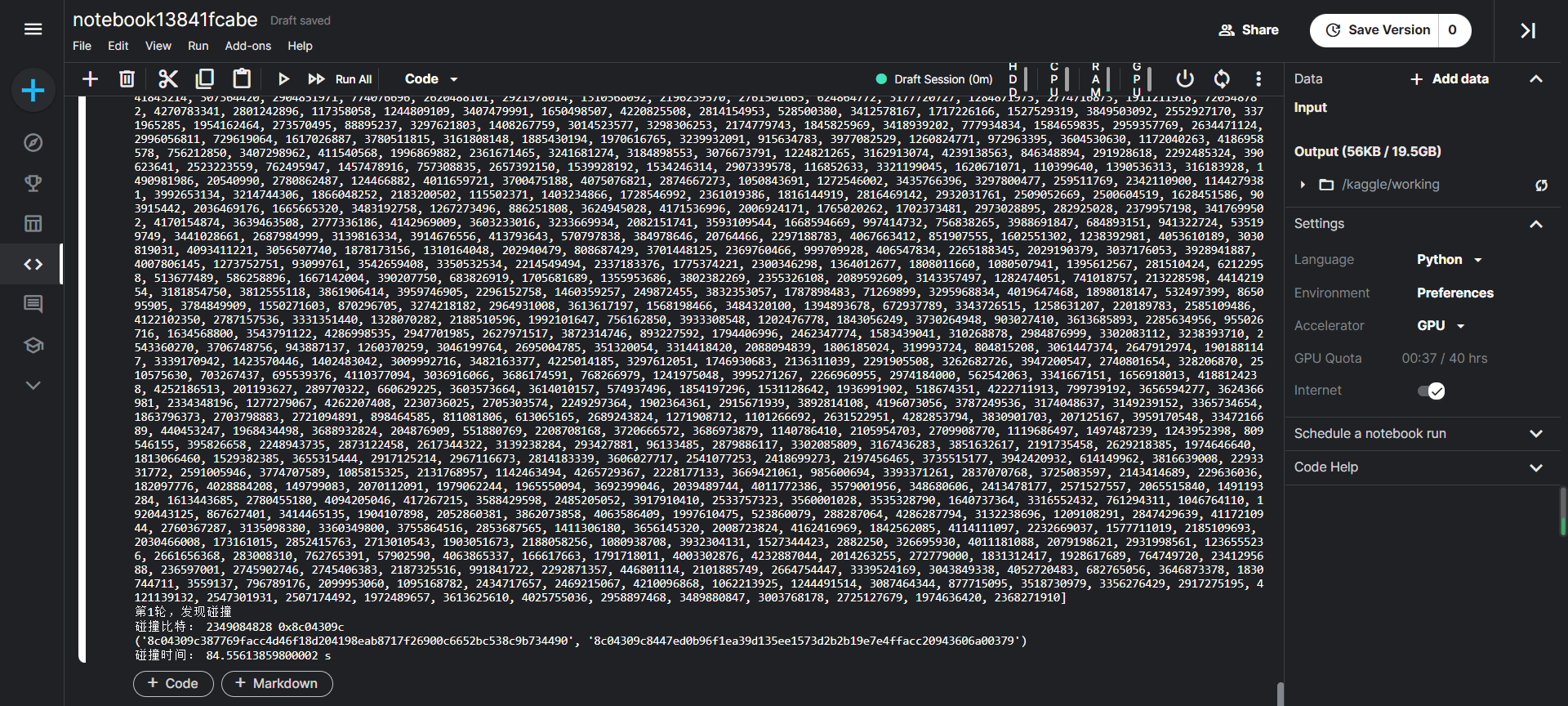
修改bit\_num可以指定碰撞位数，bit\_num需为2的倍数

3、代码运行过程截图

指定碰撞位数是8bit：



指定碰撞位数是32bit：



**三、Implement length extension attack for MD5**

Git仓库地址：<https://github.com/QingYun65/MD5LengthExtentionAttack.git>

1、项目简介

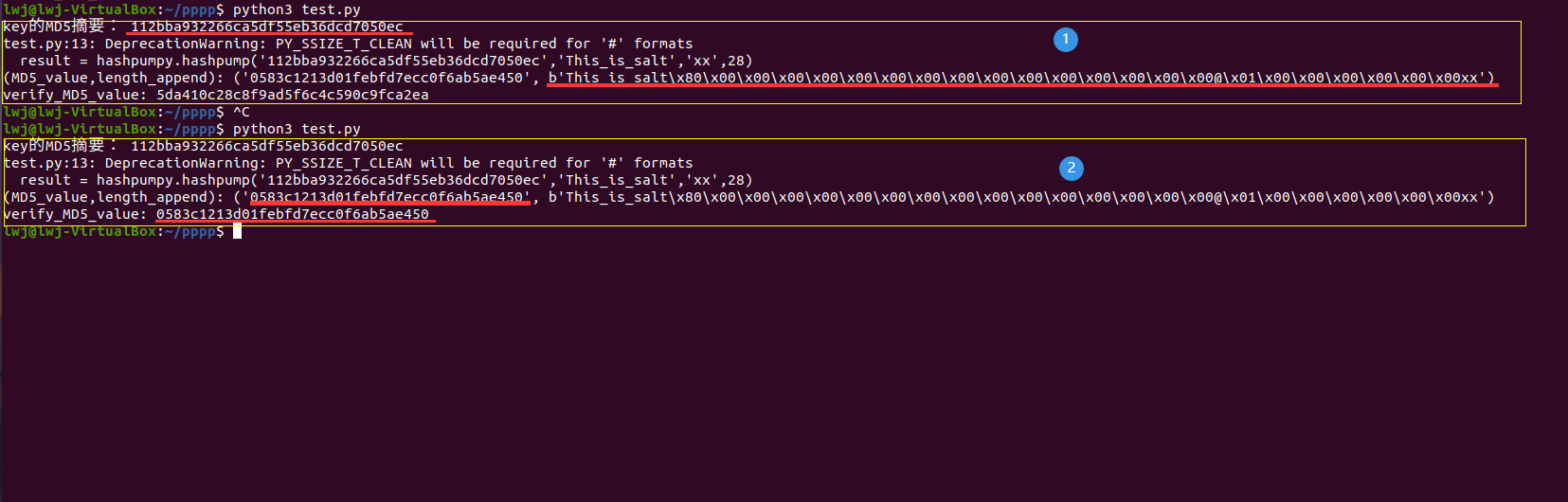
项目包括MD5算法的实现以及对于MD5算法的长度拓展攻击，长度拓展攻击部分，主要思路是：利用Hashpumpy工具，获取secret的MD5值，用该值再添加任意数据data进行第二次MD5，得到K1，根据secret和任意数据data可以重构得到一个新的输入data’，注意此时主要是对data’的长度部分进行修改，对data’进行MD5，得到K2，最后比较K1和K2的值，如果符合预期，则长度拓展攻击成功。

2、运行指导

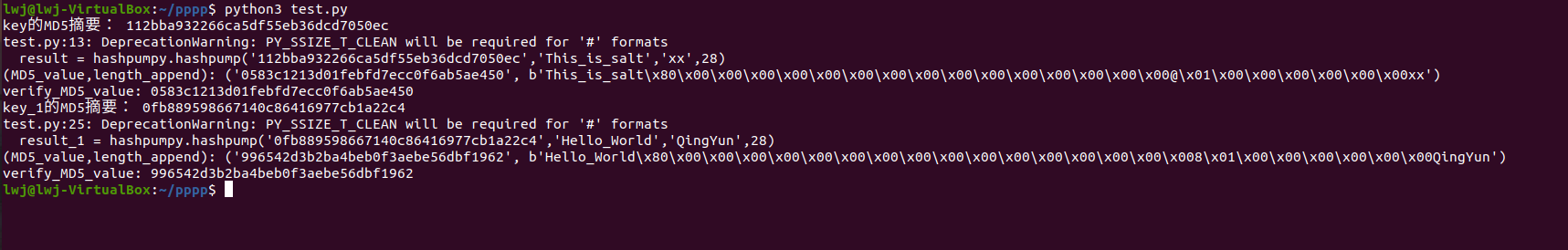
先得出key的MD5值，再将该MD5值、部分已知值‘This\_is\_salt’、要添加的新数据‘xx’和

'flag{1234567890abcdef123456}'的长度传入hashpumpy.hashpump()得到长度拓展部分和长度拓展攻击对应的MD5值。

'flag{1234567890abcdef123456}'级联得到的长度拓展攻击部分如下：'This\_is\_salt\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00@\x01\x00\x00\x00\x00\x00\x00xx'并经过MD5得到拓展攻击以后的值：'0583c1213d01febfd7ecc0f6ab5ae450'，与真实值相同，长度拓展攻击成功。



3、代码运行过程截图



**四、Optimize SM3 implementation (software)**

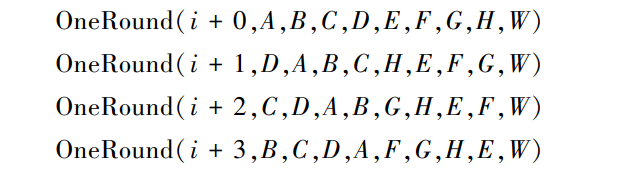
Git仓库地址：https://github.com/QingYun65/SM3-majorization.git

1、项目简介

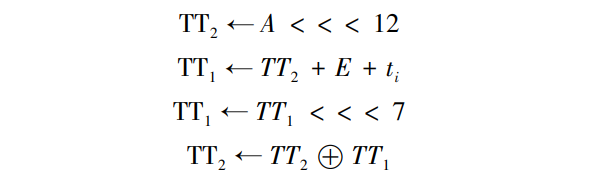
实现SM3算法的优化，从理论上讲，SM3算法中使用最多且最耗时的是64轮压缩函数和消息扩展部分，要加速SM3也就是如何快速实现SM3和消息拓展。

消息拓展的快速实现：为了尽可能减少不必要的数据加载和存储，W0,…,W67 和W’0 ,…,W’67的计算可以调整到压缩函数里执行，具体实施过程是：首先在执行64轮压缩函数前只计算初始的4个字W0, W1, W2, W3； 然后在压缩函数的第i轮生成Wi+4，而W’i则使用代替；经过这样的调整，去掉了字W’0 ,…,W’67，减少了字W0,…,W67 和W’0 ,…,W’67的加载和存储次数，提高了消息扩展的速度。

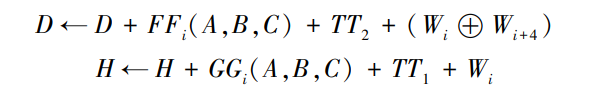
压缩函数的快速实现：压缩函数的快速实现可以从结构调整、流程变更、常数计算等方面着手。(1) 压缩函数的结构可以做适当的调整。 压缩函数每一轮的最末会执行如下所示的循环右移， ，，为了减 少循环移位导致的不必要的赋值运算，可以将字的循环右移变更每轮输入字顺序的变动，且这个顺序变动会在4轮后还原，具体情况如下(用Oneround(**·**))表示一轮压缩）：



(2) 可以优化压缩函数的中间变量的生成流程。 此优化生成流程可以去除不必要的赋值，减少中间变量个数。优化后的执行步骤如下()

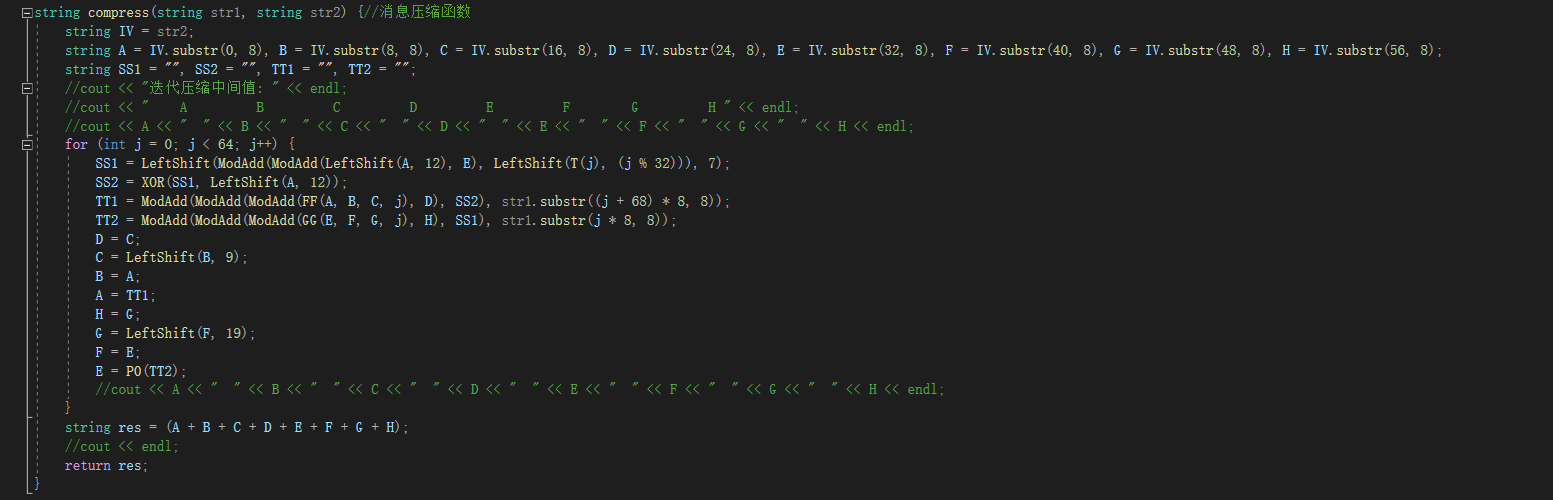


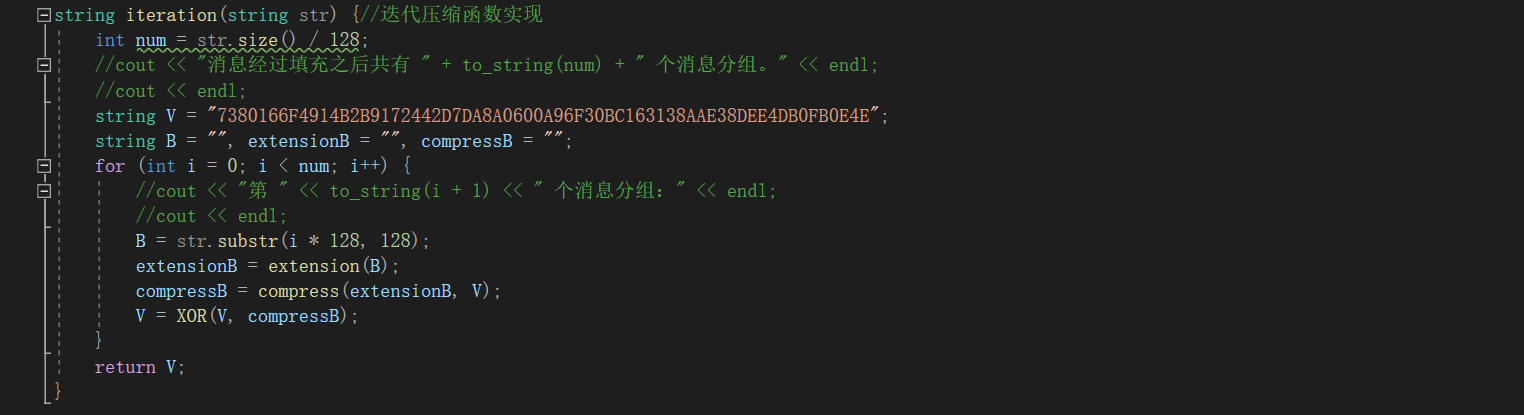
(3) 利用上述调整以及消息扩展部分的调整可 以将原来计算TT1、TT2、Ｄ和H的过程进行如下的进一步简化。



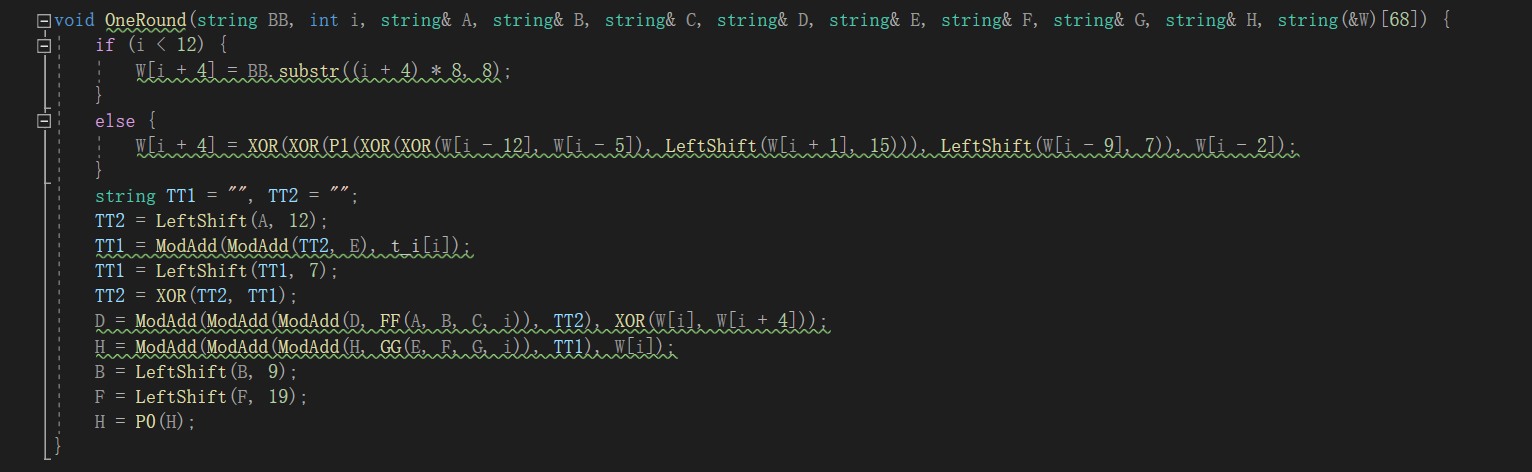
(4) 预先计算并存储常数。 这可以避免每个消息分组都去计算常数，且占用的存储空间也很少，仅256Byte。

优化前（压缩、迭代模块）：





优化后（OneRound、ProcessBlock模块）：





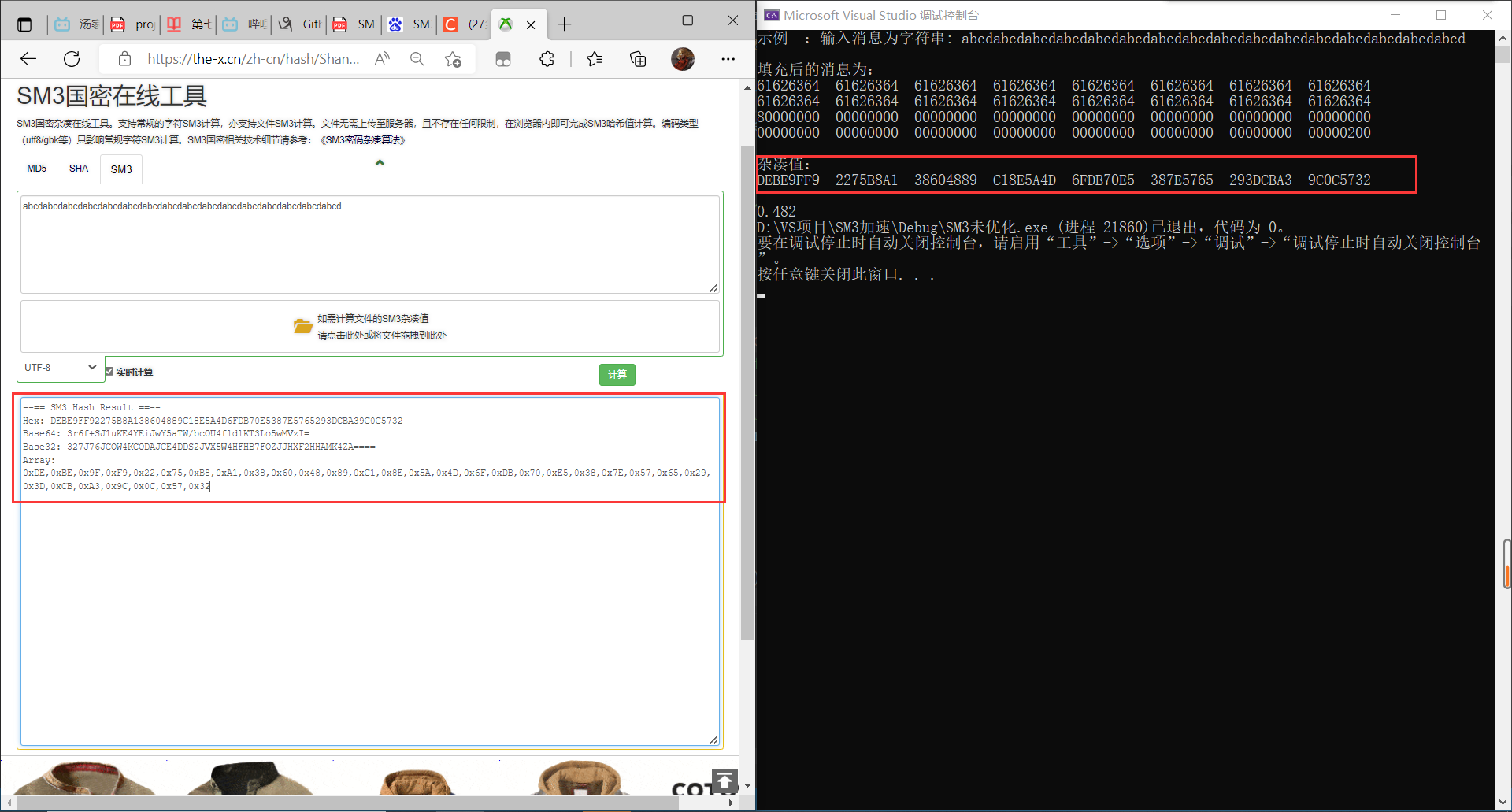
2、运行指导

选取想要测试的str\_example取消注释，运行程序，可以得到对应的SM3加密时间；

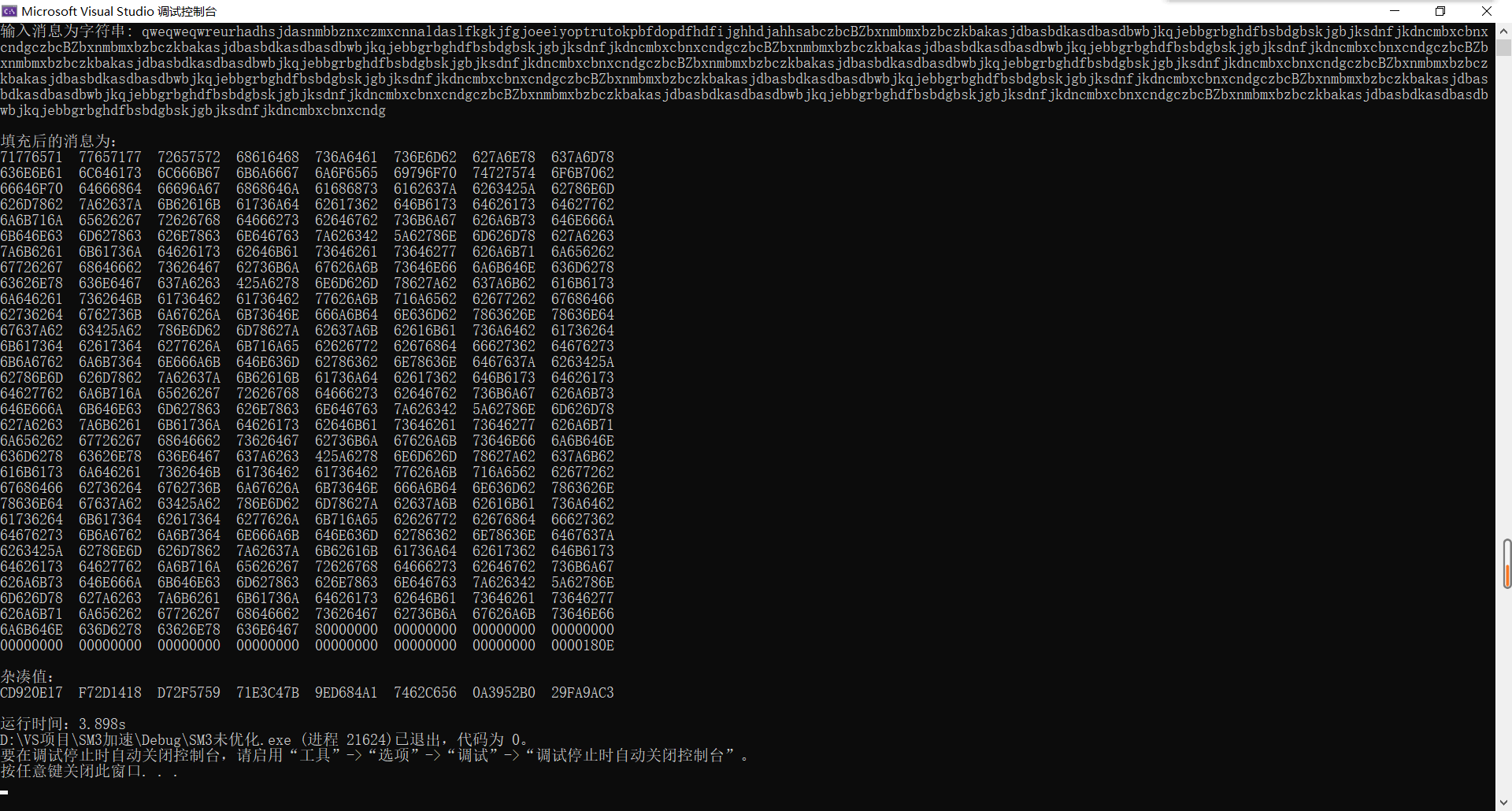
取消注释Speedtest()可以测试total\_num组特定长度随机字符串SM3加密时间；

3、代码运行过程截图

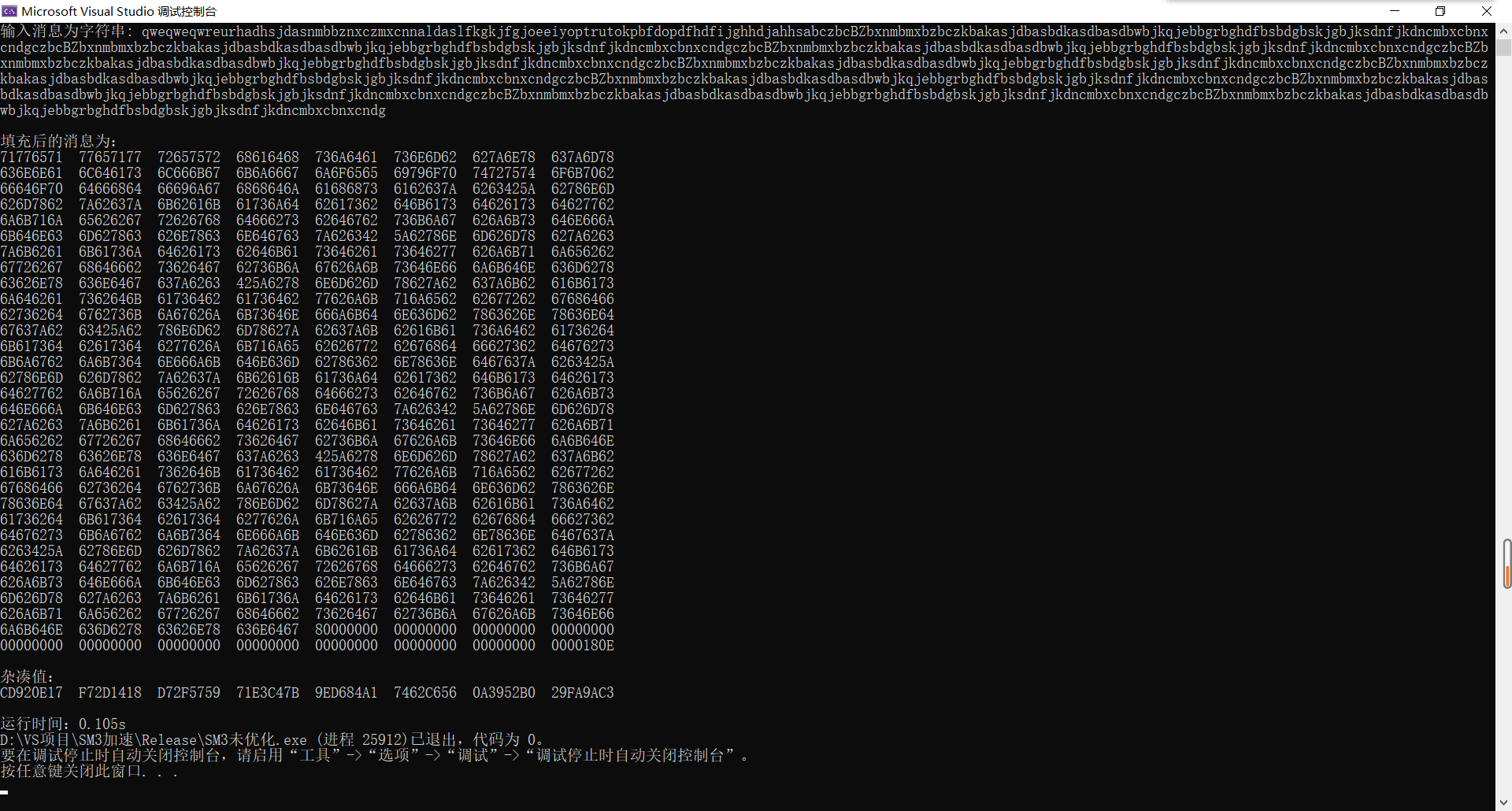
正确性测试：



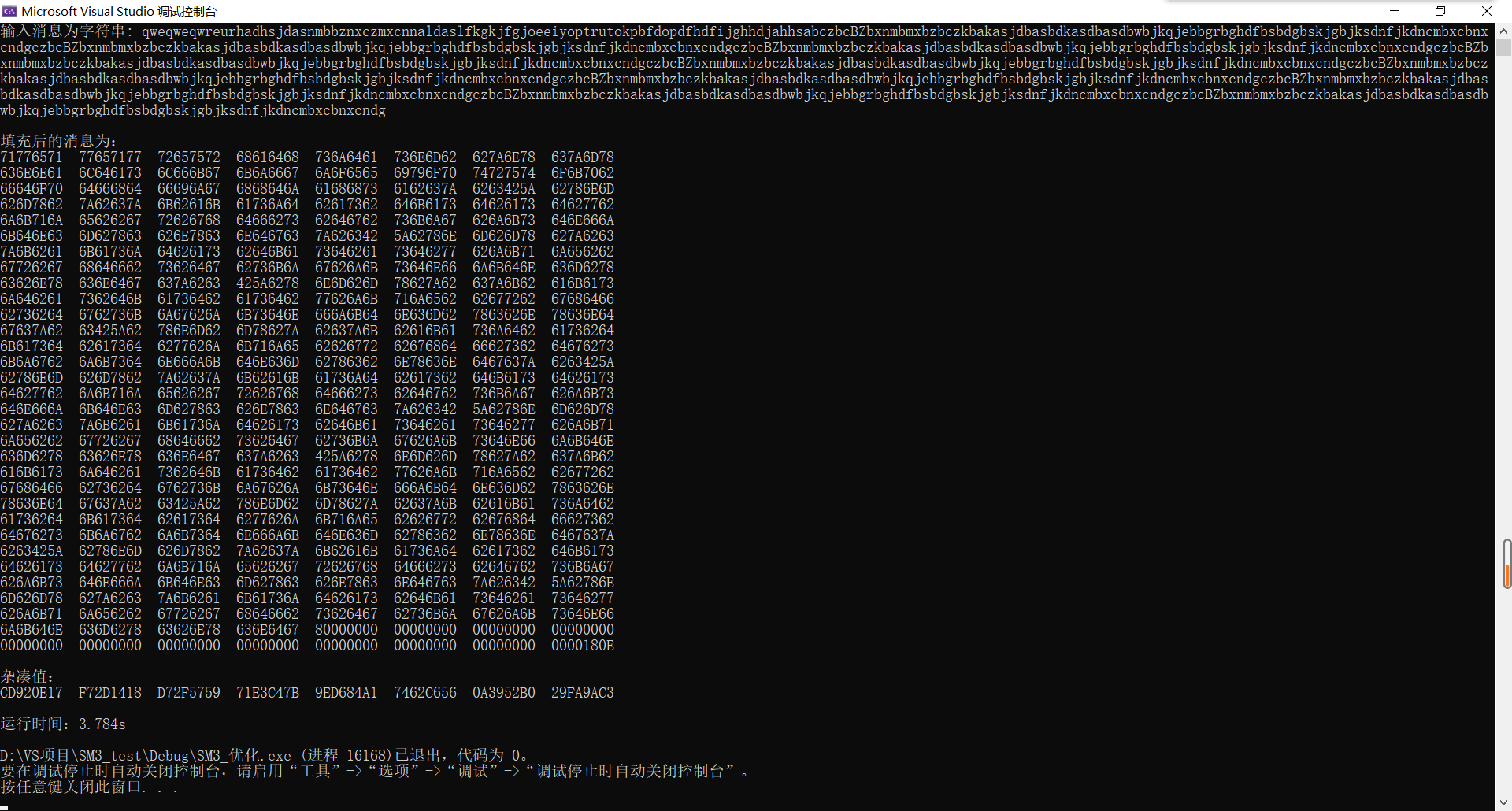
未优化（Debug）：



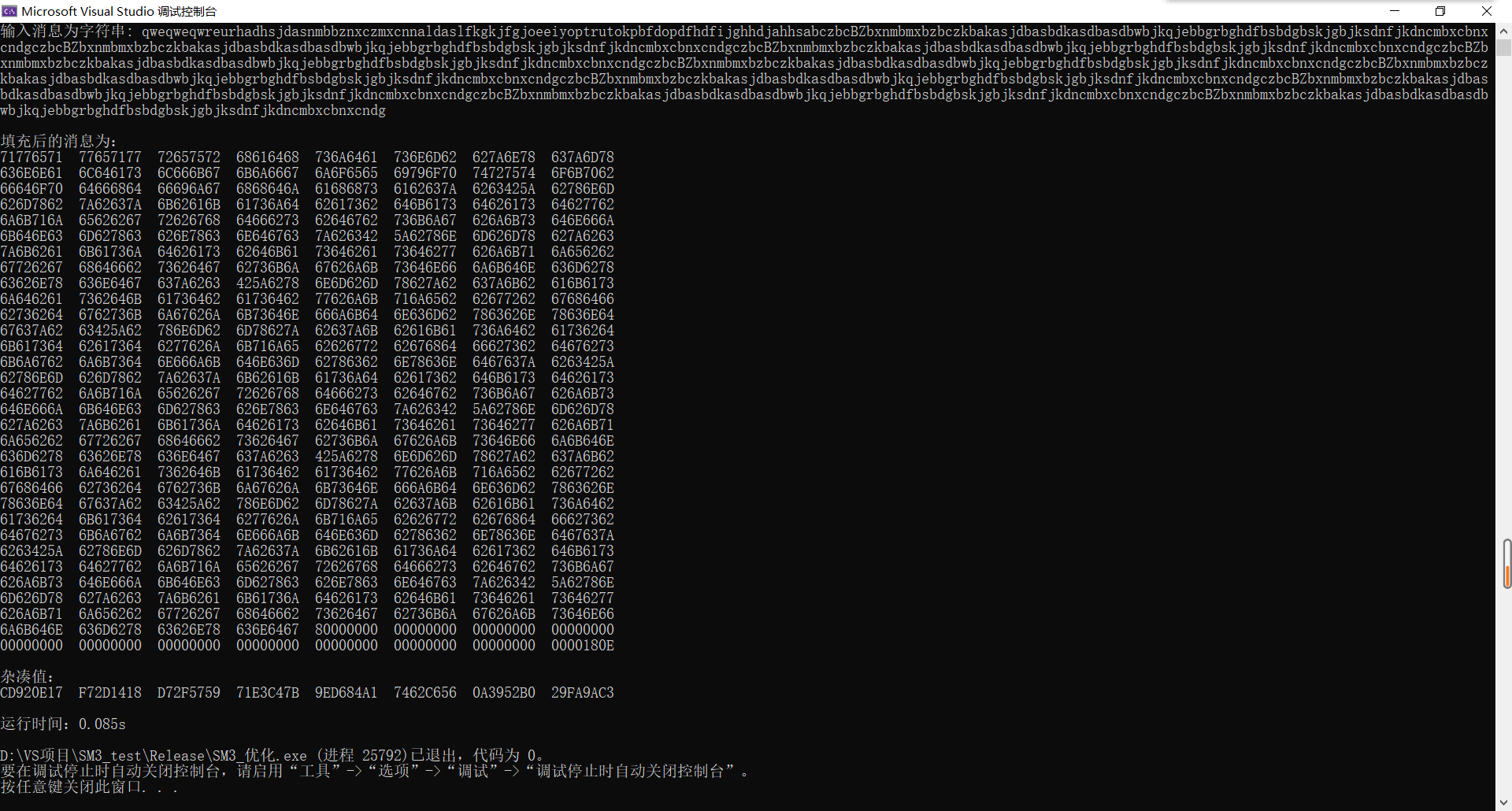
未优化（Release）：



优化后（Debug）：



优化后（Release）：



未优化1000组随机字符串（字符串长度1024）SM3加密：



优化后100组随机字符串（字符串长度1024）SM3加密：



**五、SM3 AESNI 实现**

Git仓库地址：<https://github.com/QingYun65/SM4-AESNI.git>

1、项目简介

基本思想是利用SM4与AES中S盒结构的相似性，借助Intel的AESNI指令完成S盒操作；  
（核心指令是AESNI指令集中的\_mm\_aesenclast\_si128指令）

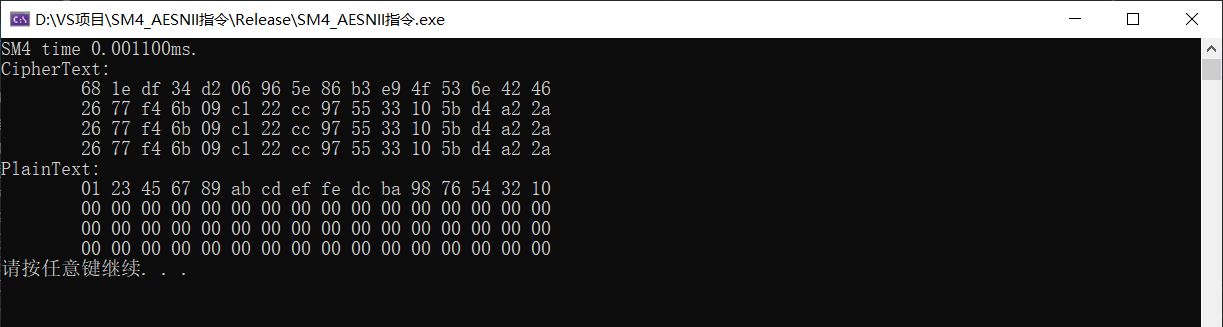
AES的S盒结构形如*Sa (x)=Aa ·Ia (x) + Ca*  ；  
SM4的S盒结构形如*Ss (x)=As ·Is (Asx+Cs) + Cs* ；  
AES使用的不可约多项式为x8 +x4 +x3 +x1 +1 ；  
SM4使用的不可约多项式为x8 +x7 +x6 +x5 x4 +x2 +1 ；  
借助于有限域同构，将SM4对应的有限域元素映射到AES对应的有限域元素中，再借助指令求逆，最后再逆映射。  
（实现时要特别注意8bit数与矩阵的对应关系，这在SM4与AES中的结果有所不同）

2、运行指导

运行main.cpp即可。

3、代码运行过程截图

先加密再解密：



**六、SM4 S盒复合域实现方法**

Git仓库地址：https://github.com/QingYun65/SM4-SBox-CompositeField.git

1、项目简介

AES 和 SM4 的 S 盒生成都是基于GF(28)进行构造的，利用逆运算和仿射变换。仿射变换本身就能表示成逻辑运算，故重点关注求逆运算。

AES 和 SM4 的表达都是基于多项式基，以 AES 的有限域为例，假设A为多项式x8+x4+x3+x+1的根，即A8+A4+A3+A+1=0,那么任何一个元素x可以表示为x=x7A7+x6A6+x5A5+x4A4+x3A3+x2A2+x1A+x0。 这种做法是将GF(28)直接看作GF(2)的8次扩域。我们也可以不这么看，将GF(28)看成GF(24)的2次扩域，GF(24)可以进一步看作GF(22)的2次扩域，再进一步GF(22)可以看作GF(2)的2次扩域。而GF(28)的求逆运算，可以通过数学表达式，转换为GF(24)的求逆和一些乘、加操作。这些操作可以进一步向下转化。

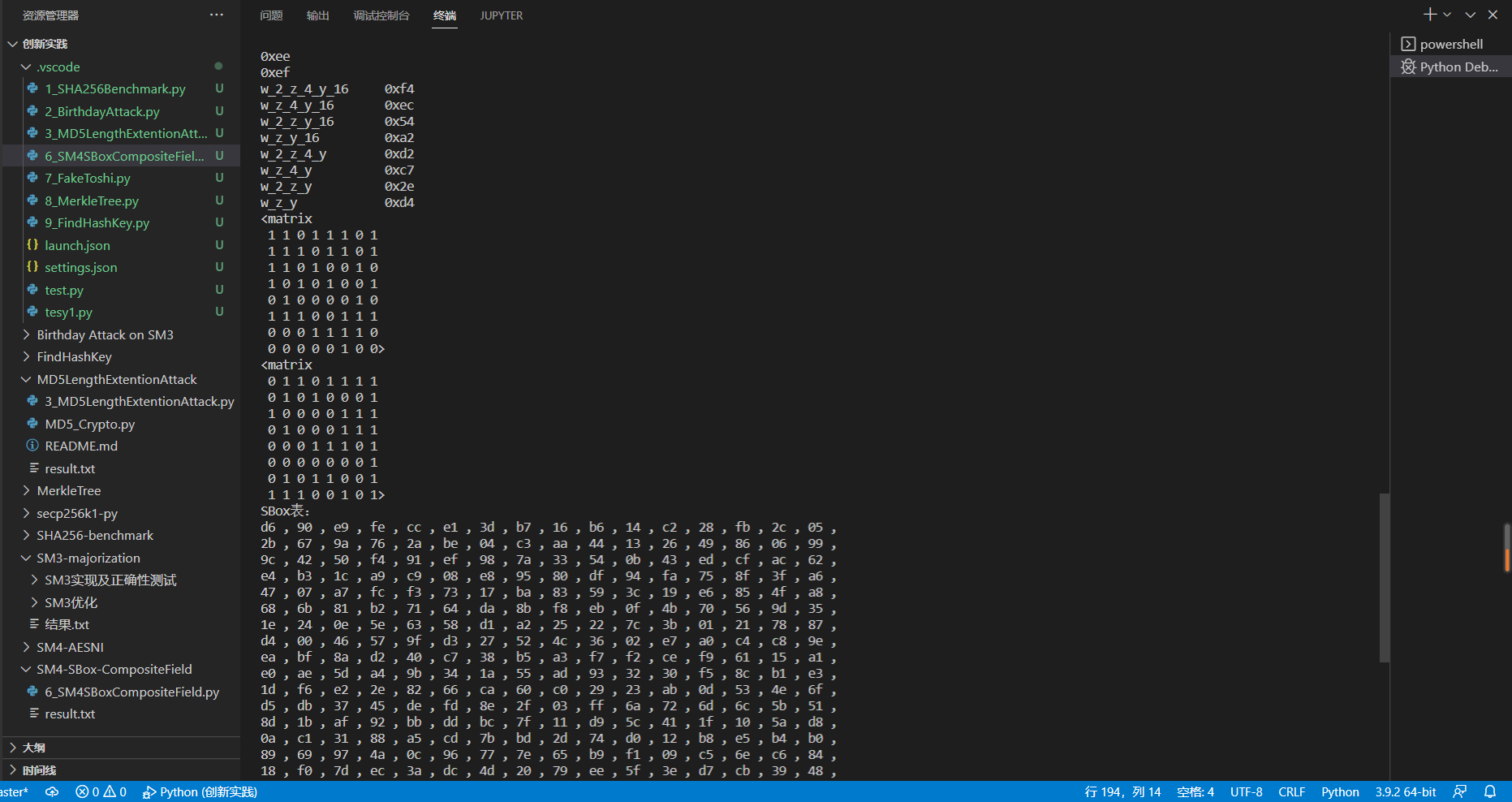
从而实现对AES和SM4的bit直接进行逻辑运算，避免查表操作，同时结合AESNI指令集可以将SM4的S盒运算转化为AES的S盒运算。

2、运行指导

（求解Sbox表项之前的过程都属于准备工作，得到所需结果后代入下一步运算）；

代码直接运行即可，结果包含中间量、矩阵X、矩阵X-1和SM4算法的SBox表。

3、代码运行过程截图



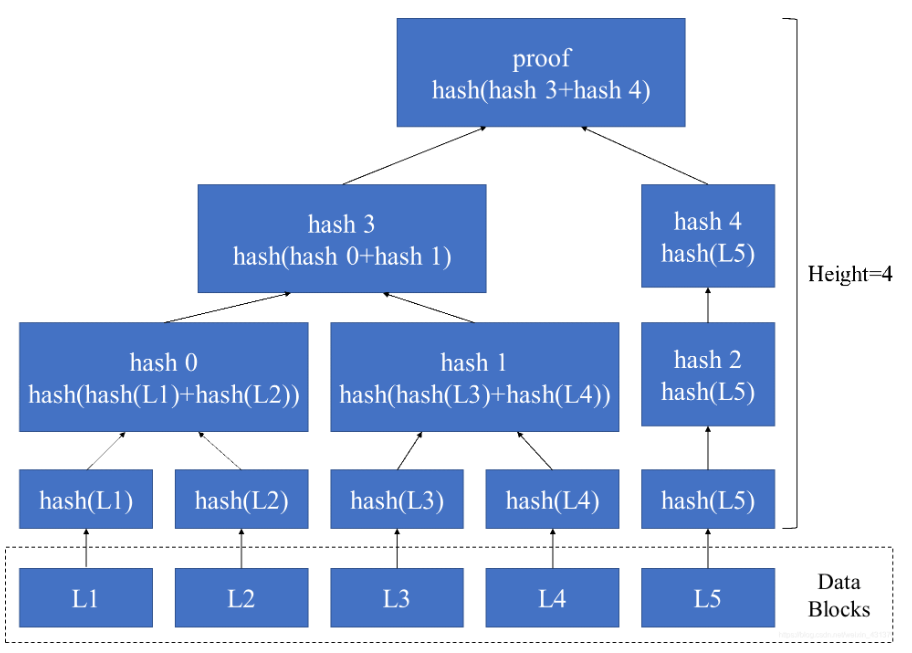
**七、Impl Merkle Tree following RFC6962**

Git仓库地址：https://github.com/QingYun65/MerkleTree-realization.git

1、项目简介

在分布式系统、P2P应用中或者是区块链中，会经常使用一种数据结构Merkle tree，Merkle树看起来非常像二叉树，其叶子节点上的值通常为数据块的哈希值，而非叶子节点上的值，所以有时候Merkle tree也可以表示为Hash tree。

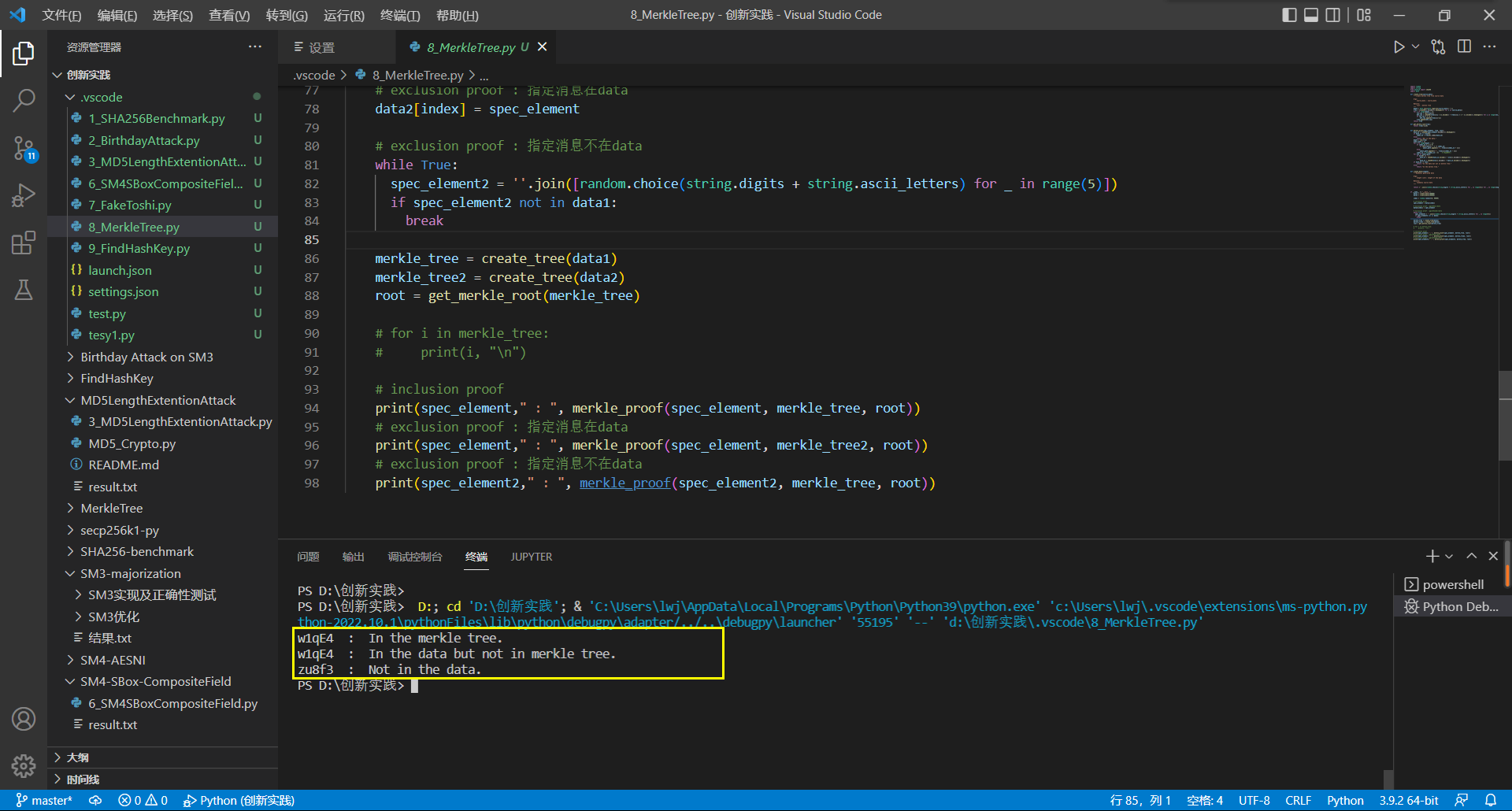
构造Merkle tree时：  
（1）对data blocks分别计算哈希值（sha256等算法）；  
（2）每层两两计算获得hash值；（如图）  
（3）直至计算至最上一层，得到proof。



2、代码运行指导

直接运行即可。

3、代码运行过程截图



**八、Find a key with hash value “sdu\_cst\_20220610” under a message composed of your name followed by your student ID. For example, “San Zhan 202000460001”.**

Git仓库地址：https://github.com/QingYun65/Find-Hash-Key.git

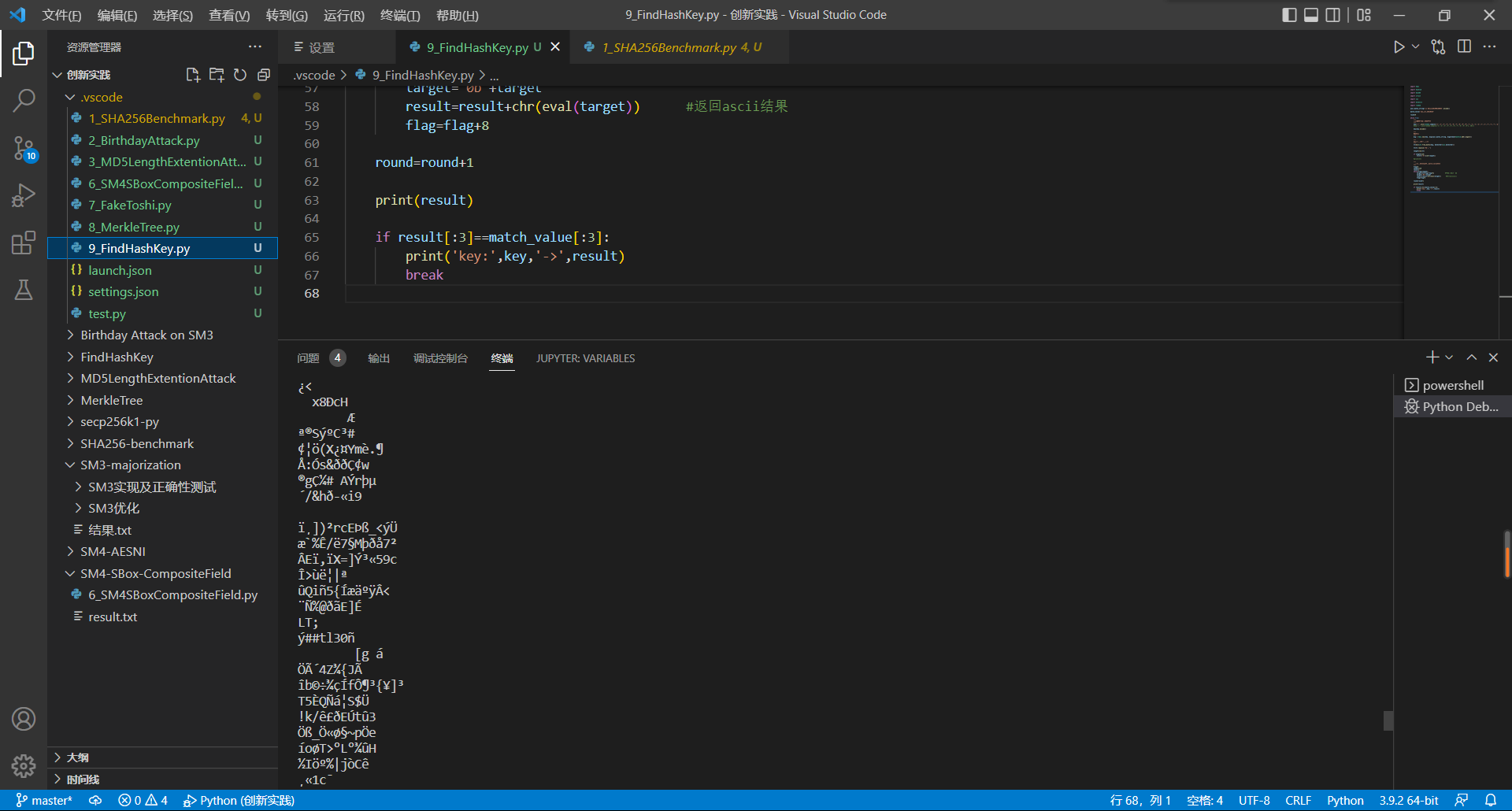
1、项目简介

选择消息为“姓名+学号“（”Li Wenjie201900190054“），哈希函数选择MD5，通过从['1','2','3','4','5','6','7','8','9','0','z','y','x','w','v','u','t','s','r','q','p','o','n','m','l','k','j','i','h','g','f','e','d','c','b','a']随机生成长度为10的key，将Hash结果与“sdu\_cst\_20220610”进行比较，若匹配则找到合适的满足要求的key。

2、运行指导

直接运行即可，但是由于Hash值位数确实比较多，密钥空间也不确定，没有成功搜索出key。

3、代码运行过程截图



**九、Find a 64-byte message under some k fulfilling that their hash value is symmetrical.**

Git仓库地址：https://github.com/QingYun65/Symmetrical-Hash-Value.git

1、项目简介

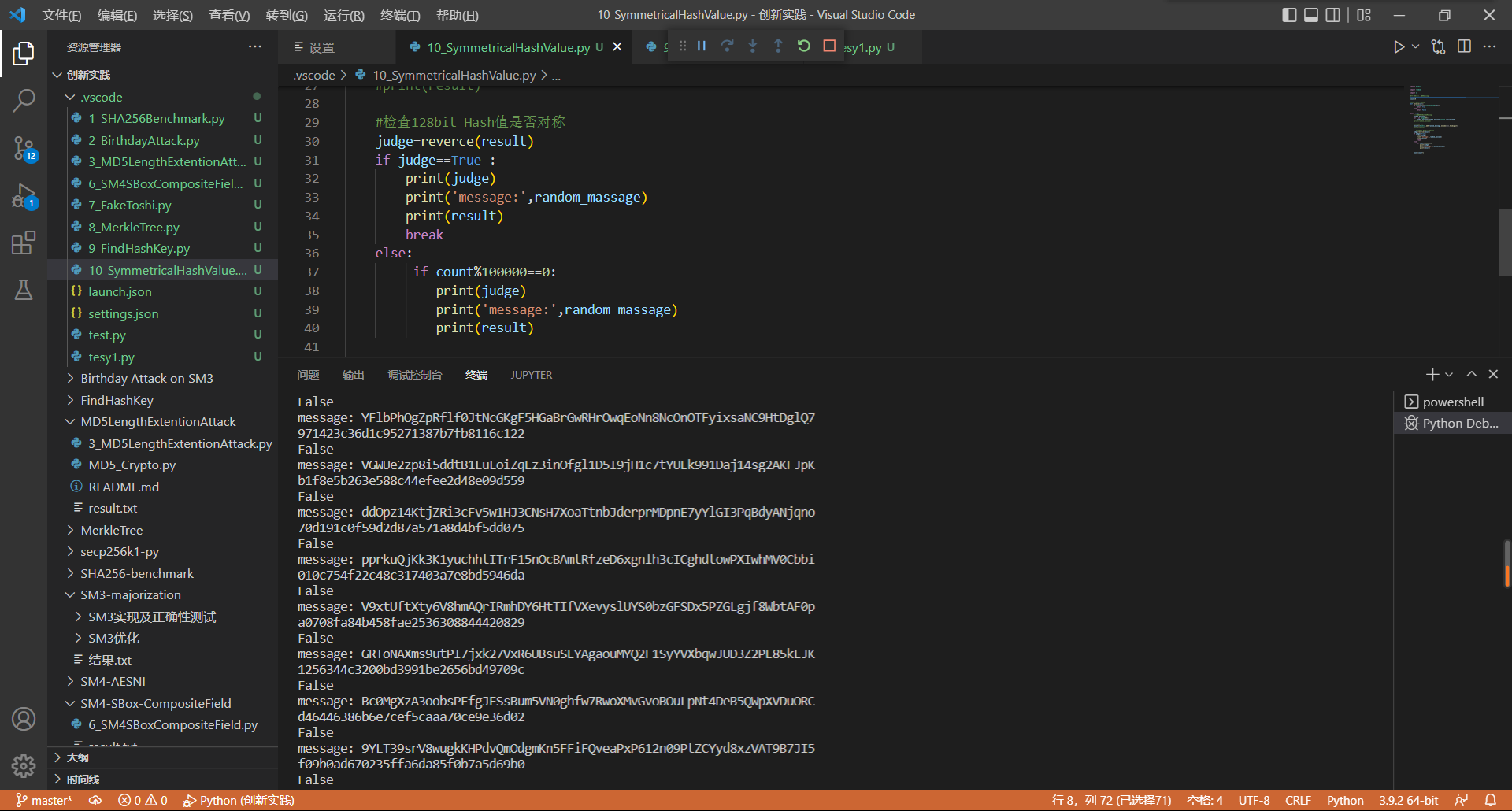
首先不添加密钥key，尝试找到一个64字节的message，经过MD5加密后其hash值是对称。基本思路：从字母表seed = "1234567890abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ"中随机生成长度为64的字符串，经过MD5加密后得到其Hash值，由reverce()函数判断Hash值是否对称，若对称则找到了符合要求的message。同时，除了找到符合要求的message以外，每100000次MD5加密以后也会打印结果。

2、运行指导

直接运行即可。

3、代码运行过程截图





**十、Forge a signature to pretend that you are Satoshi**

Git仓库地址：https://github.com/QingYun65/FakeToshiSignature.git

1、项目简介

基于签名的延展性，我们可以忽略私钥，利用已有合法签名构造另外的有效签名。由这个思想我们尝试伪造Satoshi的一个有效签名，原理如下。

Secp256k1签名过程简记如下：

生成签名：s = H(m)/k + r/k \* x*s*=*H*(*m*)/*k*+*r*/*k*∗*x*  
验证签名：sR = H（m）G + rP*sR*=*H*（*m*）*G*+*rP*  
求得R点值：R = H(m)/s\*G + r/s \* P*R*=*H*(*m*)/*s*∗*G*+*r*/*s*∗*P*  
比较r值：r == R.x*r*==*R*.*x*，验证r是否一致。

Secp256k1伪签名构造：

由验证过程可得，R = H(m)/s\*G + r/s \* P*R*=*H*(*m*)/*s*∗*G*+*r*/*s*∗*P*  
若令u = H(m)/s, v = r/s，

那么R = uG + vP，

通过随机生成（或者选取）u、v的值u’、v’，使得u'G + v'P = R'

计算出 r' = R'.x， 从而得到 s' = r'/v， H(m') = u \* s'，

那么对于消息m‘，其签名结果为(r’，s’)，该签名可以通过验证。

2、运行指导

7\_FakeToshi.py、helper.py以及ecc.py放在同一个VScode项目中，运行7\_FakeToshi.py即可得到一个可以通过验证的有效签名。

3、代码运行过程截图

