所有项目均个人（闫佳苗 202100460139）完成，以下是项目清单

\*Project1: implement the naïve birthday attack of reduced SM3

\*Project2: implement the Rho method of reduced SM3

\*Project3: implement length extension attack for SM3, SHA256, etc.

\*Project4: do your best to optimize SM3 implementation (software)

\*Project5: Impl Merkle Tree following RFC6962

\*Project10: report on the application of this deduce technique in Ethereum with ECDSA

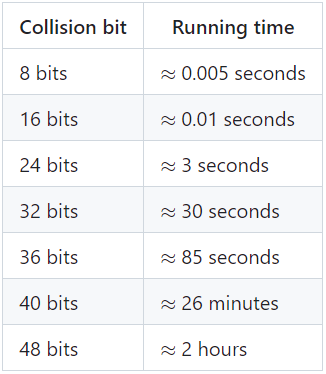
\*Project14: Implement a PGP scheme with SM2

**projec1：the naïve birthday attack of reduced SM3**

利用朴素算法对SM3进行碰撞攻击，直接寻找两个哈希相同的字符串。

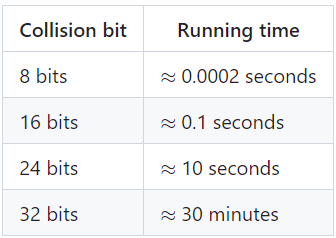
Environment: Pycharm 4.5.3 Python 3.9

Testing Result：



**Project2: implement the Rho method of reduced SM3**

**Environment:** Ubuntu 20.04  
**Testing Result：**



## 原理

## **第二原像攻击**，即给定消息M1时，攻击者能够找到另一条消息M2,其哈希值与M1的哈希值相同。本实验利用Pollard Rho算法实现了**第二原象攻击**，即对于指定的字符串，找到与之哈希相同的字符串。最终,在可接受的时间里实现了32比特的第二原象攻击。 Rho攻击（来自Pollard Rho算法），流程如下

## 1.给定具有n比特哈希值的哈希函数，选择一些随机哈希值H1,设H1'=H1

2.计算H2=Hash(H1),H2'=Hash(Hash(H1'))

3.迭代该过程并计算Hi+1=Hash(Hi)，Hi+1'=Hash(Hash(Hi'))，直到有一个i可以满足Hi+1=Hi+1'

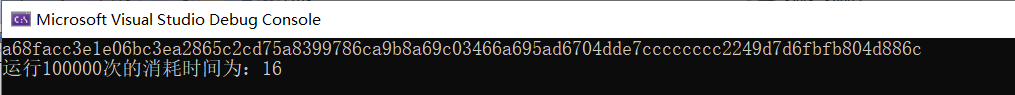
**Project3: implement length extension attack for SM3, SHA256, etc**

本任务为根据SHA256的特性，对SHA256进行长度扩展攻击。由于SHA256采用的也是md结构，进行长度扩展攻击的主要方式为首先得到一个之前的哈希值，之后再将新的消息内容添加在其后，计算出一个新的哈希值。对于正常的SHA256算法而言，其初始的h值是固定的，会随着对每一个消息分块进行迭代后随之变化，这里需要做的就是对于SHA256的实现进行一定的修改，使得其可以进入到任一轮次的迭代中。先计算出前缀消息的哈希值，以h的形式（存有8个32位长数值的数组）输出，之后设置进行长度扩展攻击的后缀消息，将h和消息作为输入进行计算，可以得到长度扩展攻击的最终结果。

**Project4: do your best to optimize SM3 implementation (software)**

运行指导：以main.cpp作为主文件，其他的作为函数库文件引入，默认运行的是优化后版本。对于SM3的优化主要是采用了SIMD方式，即单指令多数据集，应用的位置主要是在消息扩展的过程中，因为消息扩展过程里存在指令独立的情况，可以同时运行，因而引入SIMD的相关库函数，可以进行并行的操作。

优化前：



优化后：



**Project5: Impl Merkle Tree following RFC6962**

运行指导：以main.cpp作为主文件，余下两个作为函数库文件引入来运行。

本实验为实现一个merkle tree，同时对于在其中和不在其中的点进行查询。

merkle tree的主要原理为设置叶节点作为需要验证的消息，每个父节点为左右两个子节点哈希值并起来后计算的新的哈希值，直到得到根节点，才成功构造出一个merkle tree。

具体的代码已经给出，以下需要验证三个问题：

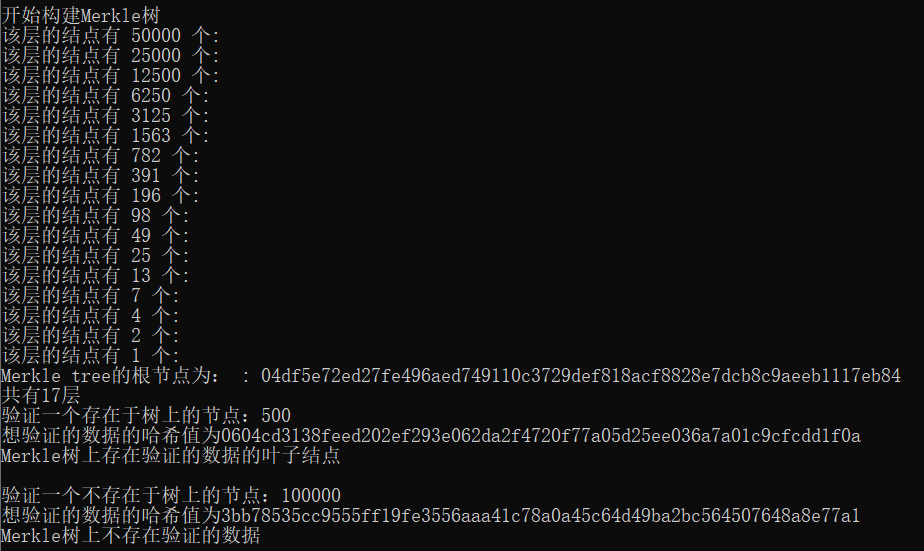
1.构造一棵有10k个叶节点的merkle tree 2.验证一个在其中的节点。

3.验证一个不在其中的节点。

以下为调用代码部分：



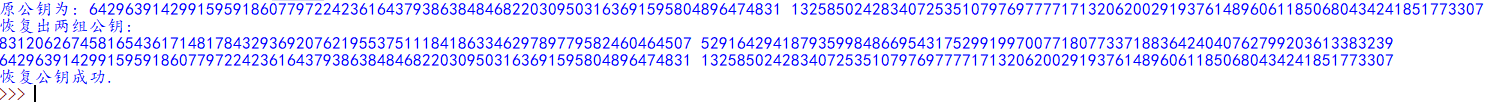
以下为运行结果，可以看到这个树为17层（不算最低一层叶节点的话），同时也通过了对于消息内容是否在其中的验证：



**Project10: report on the application of this deduce technique in Ethereum with ECDSA**

该项目首先调用ecdsa库对消息进行了签名，然后用其库中的recover\_public\_keys函数恢复出对应消息和签名所使用的公钥。这里使用的椭圆曲线是NIST256p。根据椭圆曲线的公式来看，一个x可以对应两个不同的曲线上的点，也就可以得到两个不同的公钥，且这两个公钥都可以用于验证签名。所以只要最后恢复的公钥中包含原本的公钥就算成功。

运行结果：



**Project14: Implement a PGP scheme with SM2**

该项目的内容为对sm2算法实现PGP协议，PGP协议包括非对称加密和对称加密两部分，这里sm2应用于非对称加密，而使用AES作为对称加密的算法。主要流程为先对于AES的密钥也就是会话密钥进行sm2公钥加密，再将明文部分利用AES进行加密，之后将两部分内容拼接在一起，发送给接收方。