《数据安全》实验报告

姓名：管昀玫 学号：2013750 班级：计科一班

**实验名称：**

对称可搜索加密方案实现

**实验要求：**

根据正向索引或者倒排索引机制，提供一种可搜索加密方案的模拟实现，应能分别完成加密、陷门生成、检索和解密4个过程。

**实验过程：**

1. 可搜索加密概述

可搜索加密（Searchable Encryption，简称SE）则是一种密码原语，它允许数据加密后仍能对密文数据进行关键词检索，允许不可信服务器无需解密就可以完成是否包含某关键词的判断。

在使用了可搜索加密技术的情况下，用户在向服务器提交搜索申请时如果直接提交关键字的明文，就违背了这项技术的初衷：不泄露任何关于明文的消息。因此，SE中并不是直接把关键字提交给服务器，而是对关键字进行“加密”后再提交上去。加密后的关键字是称作陷门，这种在查询关键字时提交的陷门也叫做检索陷门。

在动态可搜索加密加密中，还有另一种陷门：更新陷门，是在进行更新操作时提交的陷门。包含了更新操作(add / del，分别指插入和删除)，关键字w和文档ind。

可搜索加密可分为4个子过程（如图6.1.1所示）：

1. 加密过程：用户使用密钥在本地对明文文件进行加密并将其上传至服务器；
2. 陷门生成过程：具备检索能力的用户使用密钥生成待查询关键词的陷门（也可以称为令牌），要求陷门不能泄露关键词的任何信息；
3. 检索过程：服务器以关键词陷门为输入，执行检索算法，返回所有包含该陷门对应关键词的密文文件，要求服务器除了能知道密文文件是否包含某个特定关键词外，无法获得更多信息；
4. 解密过程：用户使用密钥解密服务器返回的密文文件，获得查询结果。



而对称可搜索加密(Symmetric searchable encryption, SSE)，旨在加解密过程中采用相同的密钥之外，陷门生成也需要密钥的参与，通常适用于单用户模型，具有计算开销小、算法简单、速度快的特点。

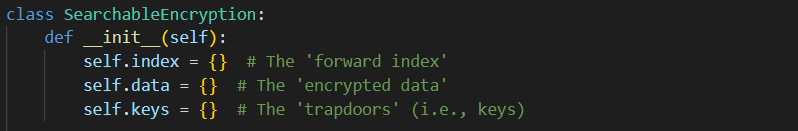
对称可搜索加密流程如下：加密过程中，用户执行算法生成对称密钥，使用加密明文文件集，并将加密结果上传至服务器。检索过程中，用户执行 算法，生成待查询关键词的陷门；服务器使用检索到文件标识符集合，并根据中文件标识符提取密文文件以返回用户；用户最终使用解密所有返回文件,得到目标文件。

1. 基于正向索引构造

基本构造思路是：将文件进行分词，提取所存储的关键词后，对每个关键词进行加密处理；在搜索的时候，提交密文关键词或者可以匹配密文关键词的中间项作为陷门，进而得到一个包含待查找的关键词的密文文件。

因为是按照“文档标识ID：关键词1, 关键词2, 关键词3, …, 关键词n”的方式组织文档与关键词的关系，我们可以称这种方式为正向索引。

首先我们需要创建一个可搜索加密类，初始化index/data和keys为字典。在此例子中，我们使用一个简单的正向索引数据结构，其中每个索引是一个单词，每个值是包含该单词的文档ID的列表。

index 存储了每个文档的加密单词列表，data 存储了每个文档的加密数据，keys 存储了每个单词的陷门（trapdoor）。

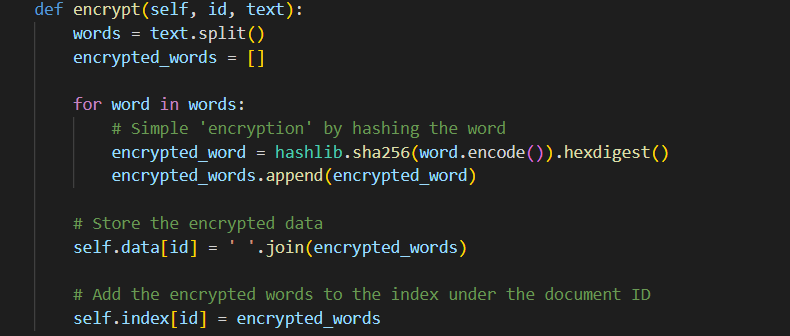
①加密过程

首先将文档按空格拆分成单词，然后对每个单词进行简单的哈希加密，并将加密后的单词列表存储到实例变量 data 中。同时，将每个加密单词列表存储到实例变量 index 中，以文档ID为键，加密单词列表为值。加密后的结果以字符串形式保存。

其中，我们使用的加密方法为SHA-256哈希函数。哈希函数是一种将任意长度的输入（在这里是单词）转换为固定长度输出的算法。哈希函数的输出通常被称为哈希值或消息摘要。SHA-256是一种常用的哈希函数，它将输入转换为一个长度为256位的十六进制数字。

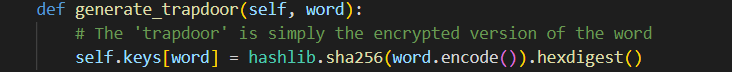
在这里，输入单词首先被编码为字节串（byte string），即由ASCII或Unicode字符组成的二进制序列，然后被传递给SHA-256哈希函数进行处理。哈希函数将这个字节串转换为一个256位的哈希值，然后使用hexdigest()方法将哈希值转换为十六进制字符串，以便存储和传输。

最终，这个加密过的单词被存储在self.keys字典中，其中它是一个“trapdoor”，可以用于后续的搜索操作。这样做的原因是因为SHA-256哈希函数具有单向性质，即不能从哈希值反推出原始输入。因此，只有知道原始输入的人才能计算出对应的哈希值。这种“trapdoor”机制可以实现可搜索加密的核心功能：在保持数据加密的同时，允许使用某些密钥搜索数据。



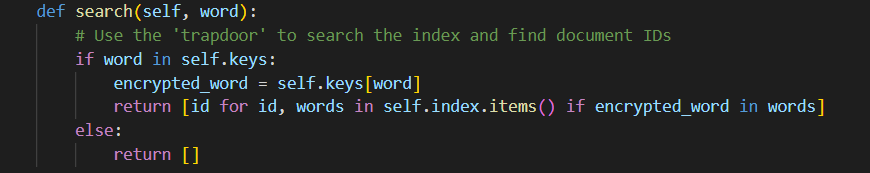
②陷门生成过程

在这里定义了 generate\_trapdoor 方法，用于生成给定单词的陷门。方法将给定单词进行简单的哈希加密，并将加密后的值存储到实例变量 keys 中，以单词为键，陷门为值。



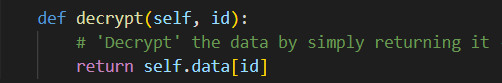
③检索过程

search函数用于在加密数据中搜索给定单词。首先通过实例变量 keys 获取给定单词的陷门。如果存在，则遍历实例变量 index 中的每个加密单词列表，找到包含给定单词的列表，并将对应的文档ID添加到结果列表中。最后返回结果列表。



④解密过程

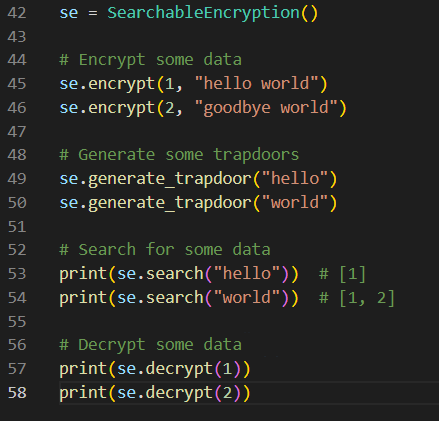
在解密过程中，直接返回以id为索引的data即可。实际上只是返回保存在data中的加密后的字符串，如果要得到原文需要使用解密密钥。



测试：

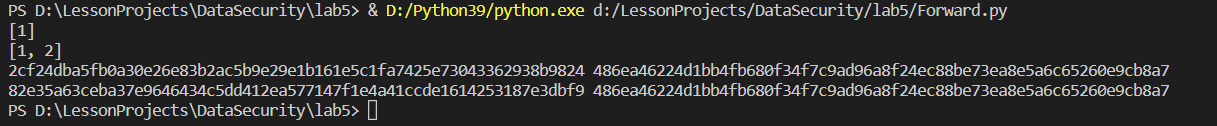
先将两个文档加密并保存到正向索引和加密数据中，然后生成两个陷门（即单词 "hello" 和 "world" 的哈希值），接着可以使用陷门在正向索引中查找匹配的文档ID。最后可以使用 decrypt 方法对加密数据进行解密。

我们尝试搜索hello和world。对于第一个搜索，理想的返回结果为第一个文档；对于第二个搜索，理想的返回结果为第一个文档和第二个文档。



结果：

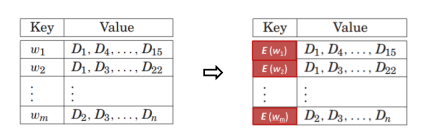
如下图所示，第一个搜索返回的是第一个文档，第二个搜索返回的是第1、2个文档。实验结果正确。



1. 基于倒排索引构造

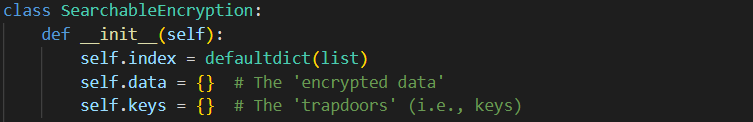
对称可搜索加密多数基于索引结构来提升检索的效率，比如倒排索引。

倒排索引（英语：Inverted index），也常被称为反向索引、置入档案或反向档案，是一种索引方法，被用来存储在全文搜索下某个单词在一个文档或者一组文档中的存储位置的映射。它是文档检索系统中最常用的数据结构。通过倒排索引，可以根据单词快速获取包含这个单词的文档列表。如图所示，关键词w1索引了一个列表，列表中存储了所有关联的文档的标识ID。



一种基于倒排索引的简单构造方法为：将关键词加密，提交密文关键词或者可以匹配密文关键词的中间项作为陷门，进而快速检索到匹配的密文文件列表，获得相应的文件标识ID。

和正向索引一样，我们首先构造一个可搜索加密类，它有三个属性：index、data 和 keys。这些属性将分别存储索引、加密数据和陷门（trapdoors），如下图所示：

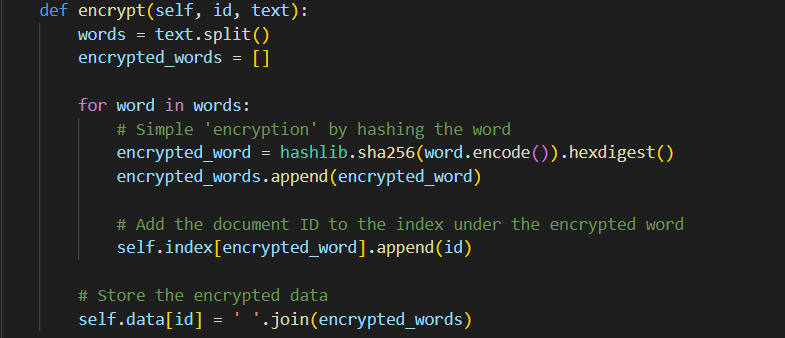


①加密过程

encrypt() 方法接受文档 ID 和文本作为输入，并将其转换为一系列单词。对于文档中的每个单词，将其哈希为加密单词，并将其添加到加密单词列表中。

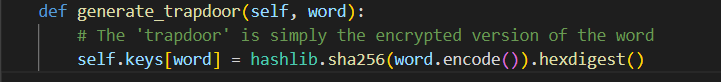
将文档 ID 添加到倒排索引中，以便在搜索时能够快速查找。将加密单词列表用空格分隔，并将其存储在加密数据字典中，使用文档 ID 作为键。

这里使用的加密方案为sha256，详见正向索引部分。



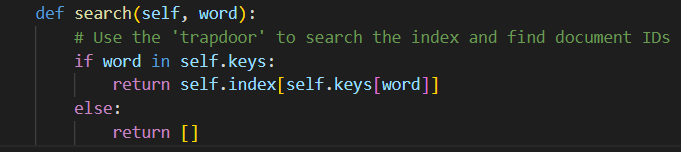
②陷门生成过程

generate\_trapdoor() 方法接受单词作为输入，并将其哈希为陷门（trapdoor），以便在搜索时使用。将陷门添加到陷门字典中，使用单词作为键。



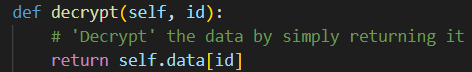
③搜索过程

搜索过程为输入单词，并使用陷门字典查找该单词的陷门。如果单词在陷门字典中，则使用倒排索引查找相应的文档 ID，将其作为列表返回。如果单词不在陷门字典中，则返回空列表。



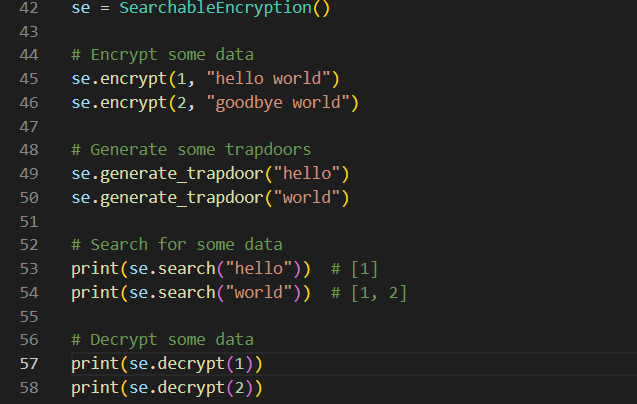
④解密过程

和正向索引类似。只是返回保存在data中的加密后的字符串，如果要得到原文需要使用解密密钥。



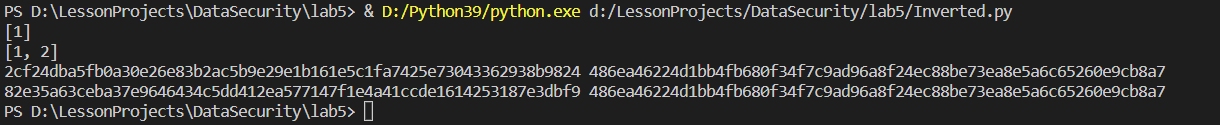
测试：

这里使用和正向索引一样的测试数据，即两个文档分别为“hello world”和“goodbye world”，分别搜索“hello”和“world”，预期结果为返回文档[1]和返回文档[1, 2]。



实验结果：

可以看到，分别返回了文档[1]和[1, 2]，说明实验正确。



**心得体会：**

通过这次实验，我更深入地了解了对称可搜索加密。通过手动实现了基于正向索引和倒排索引的可搜索加密，我的编程能力进一步提升。

在实现的过程中，我还了解了另外一种基于可搜索加密SWP方案的实现（https://www.jianshu.com/p/d620bcf235e5，GitHub代码为：https://github.com/Suxyuuu/SWP-of-searchable-encryption）

其总体思路为：

将数据划分成一个个的单词，对其进行对称加密并设置陷门后上传。查询时也会对关键词进行加密处理，防止关键词内容的泄露。整个实现过程由几个主要步骤组成（不分先后顺序）：DES加密与解密，伪随机序列的生成，带密钥的Hash函数。

DES加密解密：在openssl库函数的基础上实现。

伪随机序列：使用C++标准库中伪随机函数来近似生成。

带密钥的Hash函数：在openssl库函数的基础上实现。

通过阅读该代码，加深了我对SWP方案的理解。