数据安全 -- 对称可搜索加密方案实现

学号：2212452
姓名：孟启轩
专业：计算机科学与技术

一、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、检索和解密四个过程

二、实验思路

在本次实验中我选择倒排索引机制，来进行可搜索加密方案的模拟实现。

```
明文 → 加密 + 哈希 → 密文 + 关键词哈希值 → 倒排索引
陷门（基于条件）→ 哈希 → 匹配密文 → 解密 → 验证明文
```

1. 加密

目标：对文档及其关键词进行加密处理，并构建倒排索引
步骤：

1. **密钥生成**：随机生成加密密钥 k（可为字符串或随机数）。
2. **文档加密**：
   - 使用对称加密算法（如 AES）或非对称加密算法（如 RSA）对明文 m 进行加密，生成密文 c。
3. **关键词处理**：
   - 对文档中的每个关键词计算其哈希值（如 SHA-256），作为唯一标识符。
4. **索引构建**：
   - 将关键词的哈希值与对应文档的哈希值关联，存储到倒排索引中。
5. **结果返回**：
   - 向用户返回密文 c 和密钥 k。

2. 陷门生成

目标：根据搜索条件生成用于加密检索的陷门
步骤：

1. **条件定义**：明确搜索条件（如“仅检索长度为10的明文”）。
2. **哈希函数设计**：
   - 针对条件设计哈希函数 H，将满足条件的明文映射为固定长度的哈希值。
3. **陷门生成**：
   - 将满足条件的明文通过 H 计算哈希值，直接作为陷门 d 返回。

3. 检索

目标：通过陷门从加密索引中匹配文档
流程：

1. **客户端操作**：
   - 输入陷门 d，计算其哈希值 $h = H(d)$。
2. **服务器操作**：
   - 在倒排索引中查找哈希值为 h 的记录，返回对应的密文 c 和密钥 k。
3. **客户端验证**：
   - 使用密钥 k 解密密文 c，得到明文 m。
   - 检查 m 是否满足原始搜索条件，若满足则返回 m，否则丢弃。

4. 解密

目标：验证并还原加密文档
步骤：

1. **输入信息**：用户提供密文 c 和密钥 k。
2. **解密过程**：
   - 使用密钥 k 对密文 c 解密，得到明文 m。
3. **哈希验证**：
   - 客户端计算 m 的哈希值，并与服务器返回的文档哈希值比对。
4. **结果输出**：
   - 若哈希匹配，则确认文档合法性并显示明文 m；否则拒绝解密。

三、实验过程

1、编写程序

根据实验思路编写测试程序，程序源代码在报告最后，也可以在GitHub - MengQxuan/datasecurity上查看。

2、测试

运行程序，查询`Canon`，得到的查询结果符合预期。

```
mqx@LAPTOP-2BCGD4JI:/mnt/f/study/datasecurity/lab6$ python3 test.py
Indexing document: Canon, established in 1937 and headquartered in Tokyo, Japan, is a global leader in imaging and information technology. Renowned for its innovative optical technol
ogies, the company specializes in cameras, printers, medical equipment, and industrial solutions. Guided by the philosophy of symbiosis, Canon aims to create value through technologi
cal advancements while contributing to a sustainable society.

Processing query: Canon
Generated trapdoor: ba85ac9e... (first 8 chars shown)
Found matching encrypted words: ['77801722...']

Search Results:
匹配查询 'Canon' 的结果: Canon
mqx@LAPTOP-2BCGD4JI:/mnt/f/study/datasecurity/lab6$
```

四、心得体会

通过本次实验，我深入理解了可搜索加密技术的核心原理及其在隐私保护场景中的应用。实验过程中，我通过实现加密、陷门生成、倒排索引构建和检索流程，掌握了如何在不暴露明文的前提下支持关键词搜索的技术细节。通过模拟检索过程，我进一步加深了对加密数据处理流程的理解，并意识到在实际应用中需综合考虑算法性能、数据规模和安全性需求。

附录————完整代码

```
import hashlib

class SearchableEncryptionScheme:
    """
    基于倒排索引的可搜索加密方案实现

    包含以下核心功能：
    1. 加密（encrypt）
    2. 陷门生成（generate_trapdoor）
    3. 文档索引构建（index_document）
    4. 加密检索（search）
    """

    def __init__(self):
        """初始化加密索引结构"""
        self.forward_index = {}  # 加密词 -> 原始词集合
        self.inverted_index = {}  # 陷门 -> 加密词集合

    def encrypt(self, word):
        """
        使用SHA-256哈希算法对明文进行加密

        Args:
            word (str): 需要加密的明文单词

        Returns:
            str: SHA-256加密后的十六进制字符串
        """
        return hashlib.sha256(word.encode()).hexdigest()

    def generate_trapdoor(self, encrypted_word):
        """
        根据加密词生成搜索陷门

        Args:
            encrypted_word (str): 已加密的单词

        Returns:
            str: 用于检索的陷门值
        """
        return hashlib.sha256(encrypted_word.encode()).hexdigest()

    def index_document(self, document):
        """
        构建加密文档索引

        Args:
            document (str): 需要加密索引的文档内容
        """
        print(f"Indexing document: {document}")

        for word in document.split():
            encrypted_word = self.encrypt(word)
            trapdoor = self.generate_trapdoor(encrypted_word)

            # 构建正向索引（加密词 -> 原始词）
            if encrypted_word not in self.forward_index:
                self.forward_index[encrypted_word] = set()
            self.forward_index[encrypted_word].add(word)

            # 构建倒排索引（陷门 -> 加密词）
            if trapdoor not in self.inverted_index:
                self.inverted_index[trapdoor] = set()
            self.inverted_index[trapdoor].add(encrypted_word)

    def search(self, query):
```

```
        """
        执行加密检索操作

        Args:
            query (str): 需要搜索的明文查询词

        Returns:
            set: 匹配的原始明文单词集合
        """
        print(f"\nProcessing query: {query}")

        encrypted_query = self.encrypt(query)
        trapdoor = self.generate_trapdoor(encrypted_query)

        print(f"Generated trapdoor: {trapdoor[:8]}... (first 8 chars shown)")

        if trapdoor in self.inverted_index:
            encrypted_results = self.inverted_index[trapdoor]
            results = set()

            for encrypted_word in encrypted_results:
                results.update(self.forward_index[encrypted_word])

            print(f"Found matching encrypted words: {[e[:8]+'...' for e in encrypted_results]}")
            return results
        else:
            print("No matching trapdoor found in index")
            return set()

if __name__ == "__main__":
    # 初始化加密方案实例
    ses = SearchableEncryptionScheme()

    # 待索引的示例文档
    document = "Canon, established in 1937 and headquartered in Tokyo, Japan, is a global leader in imaging and information technology. Renow

    # 构建加密索引
    ses.index_document(document)

    # 执行搜索查询
    search_query = "Canon"
    search_results = ses.search(search_query)

    # 显示最终结果
    print("\nSearch Results:")
    if search_results:
        print(f"匹配查询 '{search_query}' 的结果: {', '.join(search_results)}")
    else:
        print(f"未找到与 '{search_query}' 匹配的结果")
```