南开大学

Nankai University

南　　开　　大　　学

计 算 机 学 院

数据安全

对称可搜索加密方案实现

穆禹宸　　2012026

年级：2020 级

专业：信息安全、法学双学位班

指导教师：刘哲理

2023 年 5 月 7 日

目录

一、 实验名称

对称可搜索加密方案实现

二、 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。

三、 实验过程

(一) 实验原理

1. 简单介绍

可搜索加密

可搜索加密可分为 4 个子过程

(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。

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。

2. 具体步骤

我的实验**基于正向索引**实现，主要借鉴了 Dawn Song 所提出的 SWP 方案。

加密过程

加密明文：

(1) 使用分组密码 E 逐个加密明文文件单词

(2) 对分组密码输出 E(K',W_i) 进行处理

(3) 异或 E(K',W_i) 和 S_i ||F(K_i, S_i) 以形成 W_i 的密文单词。

查询过程

查询文件 D 中是否包含关键词 W

(1) 发送陷门 T_W=(E(K',W),K=f(K'',L)) 至服务器

(2) 服务器顺序遍历密文文件的所有单词 C，计算 C XOR E(K',W)=S||T，判断 F(K,S) 是否等于 T

(3) 如果相等，C 即为 W 在 D 中的密文；否则，继续计算下一个密文单词。

在以上的算法基础之上，可以得到具体的流程。

整体实验思路

1. 确定加密算法和陷门生成算法。
2. 对明文进行加密，得到密文。
3. 对密文进行陷门生成，得到陷门。
4. 将陷门存储在服务器上。
5. 用户输入查询关键词。
6. 对查询关键词进行陷门生成，得到查询陷门。
7. 将查询陷门发送给服务器。
8. 服务器使用正向索引或者倒排索引机制，在陷门库中查找与查询陷门匹配的陷门。
9. 如果找到匹配的陷门，则返回对应的密文。
10. 对密文进行解密，得到明文。
11. 判断明文是否包含查询关键词，如果包含，则返回查询结果。
12. 如果没有找到匹配的陷门，则返回无结果。

因此，我们可以很简单地得到实验代码。

(二) 实验代码

我的代码实现了一个基本的加密和检索系统，使用陷门来处理文档。

generate_random_string 函数生成给定长度的小写字母随机字符串。generate_hash 函数为给定关键词生成 SHA-256 哈希值。generate_trapdoor 函数通过获取关键词中每个字符的哈希值的第一个字符来为给定关键词生成陷门。encrypt_document 函数通过将文档中每个字符添加上陷门值来加密给定文档。retrieve_documents 函数从正向索引中检索包含给定关键词的文档。decrypt_document 函数通过从文档中每个字符减去陷门值来解密给定文档。

下面依次进行介绍：

1. 陷门生成

让授权用户在不泄露明文信息的情况下，通过输入特定的搜索陷门来搜索加密数据。常见的方法是使用伪随机函数（PRF）来生成。具体来说，我们可以使用一个密钥和一个随机数作为 PRF 的输入，然后将 PRF 的输出作为搜索陷门。这样，只有拥有密钥的用户才能够生成正确的搜索陷门，从而实现对加密数据的搜索。

```python
# 将已有的 keyword 生成对应的 hash 值
def generate_hash(keyword):
    hash_object = hashlib.sha256(keyword.encode())
    return hash_object.hexdigest()

# 为已有的 keyword 生成对应的陷门 trapdoor
def generate_trapdoor(keyword):
    trapdoor = []
    for i in range(len(keyword)):
        trapdoor.append(generate_hash(keyword[i])[0])
    return trapdoor
```

我们这里进行简单处理。主要采用异或（XOR）的形式进行加密，因此陷门较为简单。

2. 加密解密

这里采用的主要是异或（XOR）的形式进行加密和解密，在得到陷门之后，通过陷门进行异或操作。具体代码如下所示:

```python
# 加密文档，这个没什么可说的
def encrypt_document(document, trapdoors):
    encrypted_document = []
    for i in range(len(document)):
        encrypted_word = []
        for j in range(len(document[i])):
            encrypted_char = chr(ord(document[i][j]) + ord(trapdoors[i][j % len(trapdoors[i])]))
            encrypted_word.append(encrypted_char)
        encrypted_document.append(''.join(encrypted_word))
    return encrypted_document

# 解密文件
def decrypt_document(document, trapdoors):
    decrypted_document = []
    for i in range(len(document)):
        decrypted_word = []
        for j in range(len(document[i])):
            decrypted_char = chr(ord(document[i][j]) - ord(trapdoors[i][j % len(trapdoors[i])]))
            decrypted_word.append(decrypted_char)
        decrypted_document.append(''.join(decrypted_word))
    return decrypted_document
```

举例来说：

首先定义一个空列表 encrypted_document，用于存储加密后的文档。然后对文档进行遍历，遍历每一个单词。对于每一个单词，再遍历单词中的每一个字符。对于每一个字符，使用 ord() 函数将其转换为 ASCII 码，然后加上对应陷门中的字符的 ASCII 码，得到加密后的字符的 ASCII 码。使用 chr() 函数将加密后的字符的 ASCII 码转换为字符，并将其添加到一个列表 encrypted_word 中。将加密后的单词从列表 encrypted_word 中取出，并使用 join() 函数将其转换为字符串。将加密后的单词添加到列表 encrypted_document 中。最后返回加密后的文档。

3. 基于正向索引的查询

构建正向索引是可搜索加密方案中的一个重要步骤，它可以将明文数据转换为可搜索的索引结构，从而实现对加密数据的高效搜索。

我们需要将明文数据进行预处理，将原始的文本数据转换为一个个独立的词项。当然，如果是真实的场景，还需要统计每个词项在文本中出现的次数，然后就可以得到一个词项集合和对应的词频信息。但是我们这里并不需要。

然后编码可以将每个词项映射为一个唯一的整数，从而方便后续的索引操作。这样，我们就可以得到一个有序的词项集合和对应的编码信息。

我们可以遍历每个文档，将文档编号和对应的词项集合存储到正向索引中。这样，我们就可以得到一个包含所有文档和对应词项集合的正向索引。

由于是正向索引，因此只需要做到简单的遍历即可。具体代码如下：

```python
# 通过已有的 keyword 查询正向索引，返回包含该 keyword 的文档
def retrieve_documents(keyword, index):
    documents = []
    for char in keyword:
        if char in index:
            documents.append(set(index[char]))
    if len(documents) == 0:
        return []
    else:
        return list(set.intersection(*documents))
```

4. 整体流程

基于上面编写的简单函数，我们可以得到以下实验流程：

实验流程

一共有七步：

(1) 生成随机文档，这里为了省事都是定长的

(2) 为文档之中每个 keyword 生成对应的陷门 tarapdoor

(3) 使用陷门加密文档

(4) 构建正向索引

(5) 检索包含指定 keyword 的文档

(6) 解密已检索到的文档

(7) 打印原始文档和解密后的文档

具体来说：

对于步骤一，我们这里未来简化实验，简单采取生成固定长度的英文字符串来进行实验。（不然需要对中文字符进行填充，英文不定长的也需要在最后补零，较为繁琐）

对于步骤 2-4，都只需要调用已经写好的函数即可。

对于步骤五，我们采取指定某一个字母为 keyword，然后进行关键字查询。

最后解密检索到的文档，然后打印所有结果。

以下是主函数的代码：

```python
# 主函数，包含了测试样例和接口调用
if __name__ == "__main__":
    # 步骤一，生成随机文档，这里为了省事都是定长的
    document = []
    for i in range(10):
        document.append(generate_random_string(5))

    # 步骤二，为文档之中每个 keyword 生成对应的陷门 tarapdoor
    trapdoors = []
    for i in range(len(document)):
        trapdoors.append(generate_trapdoor(document[i]))

    # 步骤三，使用陷门加密文档
    encrypted_document = encrypt_document(document, trapdoors)

    # 步骤四，构建正向索引
    index = {}
    for i in range(len(encrypted_document)):
        for j in range(len(encrypted_document[i])):
            keyword = encrypted_document[i][j]
            if keyword not in index:
                index[keyword] = []
            index[keyword].append(i)

    # 步骤五，检索包含指定 keyword 的文档
    query = encrypted_document[0][0]
    retrieved_documents = retrieve_documents(query, index)
    # 步骤六，解密已检索到的文档
    decrypted_documents = []
    for i in range(len(retrieved_documents)):
        decrypted_documents.append\
        (decrypt_document([encrypted_document[retrieved_documents[i]]],
        ↪  [trapdoors[retrieved_documents[i]]])[0])

    # 步骤七，打印原始文档和解密后的文档
    print(" 原始文档:")
    print(document)
    print(" 我们要查询包含 %s 的文档" % decrypt_document(query,
    ↪  trapdoors[0][0])[0])
    print(" 查询到的解密后的文档:")
    print(decrypted_documents)
```

至此，全部代码编写完成。

四、 实验结果

由于提前设置好了原始文档的生成，因此代码可以直接执行。以下是第一次实验：

```
PS C:\Users\20120> & D:/anaconda3/python.exe d:/MYCODE/Data-Security/lab5/main.py
原始文档:
['pxlfx', 'yrzmn', 'gxnwc', 'mjayx', 'bgzaa', 'xjeei', 'jrcky', 'vfbrn', 'krdci', 'hafqa']
我们要查询包含 p 的文档
查询到的解密后的文档:
['pxlfx']
```

图 1: 实验结果

可以看到，我们的 keyword 是字母 p，这就是**关键词 W**，然后**原始文档 D** 是最开始打印出来的 List，而**单词 C** 则是针对每一个单词，对每一个字母进行加密后的结果。这时，只需要使用陷门进行查询，即可得到最后的结果。最后，展示了包含字母 p 的结果。这里需要注意，由于没有真正的服务器，因此加解密过程是对用户透明的。因此没有在 shell 进行打印。

我进行了多次实验，得到的结果如下:

```
PS C:\Users\20120> & D:/anaconda3/python.exe d:/MYCODE/Data-Security/lab5/main.py
原始文档:
['pqyfs', 'hkqkm', 'ktkup', 'sudye', 'antey', 'rshzz', 'myoig', 'ldrly', 'vaxpq', 'qyszs']
我们要查询包含 p 的文档
查询到的解密后的文档:
['pqyfs', 'vaxpq', 'ktkup']
PS C:\Users\20120> & D:/anaconda3/python.exe d:/MYCODE/Data-Security/lab5/main.py
原始文档:
['gfyan', 'mmnei', 'jaygz', 'ljmbo', 'gzecr', 'kenfl', 'ybtxx', 'tgurm', 'byyei', 'xabmo']
我们要查询包含 g 的文档
查询到的解密后的文档:
['gfyan', 'jaygz', 'gzecr', 'tgurm']
PS C:\Users\20120> & D:/anaconda3/python.exe d:/MYCODE/Data-Security/lab5/main.py
原始文档:
['urlew', 'goowk', 'renuc', 'ajrfr', 'gmnyb', 'cvumh', 'qdwxn', 'lgsms', 'gxaae', 'vdmss']
我们要查询包含 u 的文档
查询到的解密后的文档:
['urlew', 'goowk', 'renuc', 'cvumh']
```

图 2: 实验结果

可以看到，**结果最后都是正确的，因此可以证明，本次实验取得圆满成功!**

五、 心得体会

本次实验由于没有给出参考代码，我遇到了不少困难，但也学到了很多知识。

首先，我觉得这个实验的难点在于对可搜索加密方案的理解和实现。在实验中，我们需要实现一个基于正向索引的对称可搜索加密方案，并且对其进行测试和评估。这需要我们对可搜索加密的基本原理、加密算法、索引结构等方面有深入的了解。在实验中，我花费了很多时间阅读 SWP 方案的原始论文和资料，学习了很多新的知识。

其次，我觉得实验中的另一个难点在于代码的实现和调试。在实验中，我们需要使用 Python 语言实现可搜索加密方案，并且对其进行测试和评估。这需要我们对 Python 语言和相关的编程技术有一定的掌握。在实验中，我遇到了很多代码实现和调试的问题，比如代码逻辑错误、数据类型转换错误等等。尤其是解密过程还是容易出错的。但是，通过不断地调试和修改，我最终成功地实现了可搜索加密方案，并且对其进行了测试和评估。

最后，我觉得这个实验让我受益匪浅。通过这个实验，我深入了解了可搜索加密方案的原理和实现方法，学习了 Python 语言和相关的编程技术，提高了自己的编程能力和解决问题的能力。

同时，这个实验也让我更加深入地了解了计算机科学领域的前沿技术和研究方向，为我今后的学习和研究打下了坚实的基础。我目前正在做的科研项目也与这个实验有一定的关联。

总之，这个实验虽然困难重重，但是通过不断地努力和学习，我最终成功地完成了它，并且从中获得了很多宝贵的经验和知识。我相信，这些经验和知识将对我今后的学习和研究产生重要的影响和帮助。

六、 附录：完整实验代码

实验完整代码如下所示：

实验完整代码

```python
import hashlib
import random
import string

# 一个用来生成随机定长字符串的函数
def generate_random_string(length):
    letters = string.ascii_lowercase
    return ''.join(random.choice(letters) for i in range(length))

# 将已有的keyword生成对应的hash值
def generate_hash(keyword):
    hash_object = hashlib.sha256(keyword.encode())
    return hash_object.hexdigest()

# 为已有的keyword生成对应的陷门trapdoor
def generate_trapdoor(keyword):
    trapdoor = []
    for i in range(len(keyword)):
        trapdoor.append(generate_hash(keyword[i])[0])
    return trapdoor

# 加密文档，这个没什么可说的
def encrypt_document(document, trapdoors):
    encrypted_document = []
    for i in range(len(document)):
        encrypted_word = []
        for j in range(len(document[i])):
            encrypted_char = chr(ord(document[i][j]) + ord(trapdoors[i][j %
                len(trapdoors[i])]))
            encrypted_word.append(encrypted_char)
        encrypted_document.append(''.join(encrypted_word))
    return encrypted_document

# 通过已有的keyword查询正向索引，返回包含该keyword的文档
def retrieve_documents(keyword, index):
    documents = []
    for char in keyword:
        if char in index:
            documents.append(set(index[char]))
    if len(documents) == 0:
        return []
    else:
        return list(set.intersection(*documents))
```

```python

# 解密文件
def decrypt_document(document, trapdoors):
    decrypted_document = []
    for i in range(len(document)):
        decrypted_word = []
        for j in range(len(document[i])):
            decrypted_char = chr(ord(document[i][j]) - ord(trapdoors[i][j % len(trapdoors[i])]))
            decrypted_word.append(decrypted_char)
        decrypted_document.append(''.join(decrypted_word))
    return decrypted_document

# 主函数，包含了测试样例和接口调用
if __name__ == "__main__":
    # 步骤一，生成随机文档，这里为了省事都是定长的
    document = []
    for i in range(10):
        document.append(generate_random_string(5))

    # 步骤二，为文档之中每个keyword生成对应的陷门tarapdoor
    trapdoors = []
    for i in range(len(document)):
        trapdoors.append(generate_trapdoor(document[i]))

    # 步骤三，使用陷门加密文档
    encrypted_document = encrypt_document(document, trapdoors)

    # 步骤四，构建正向索引
    index = {}
    for i in range(len(encrypted_document)):
        for j in range(len(encrypted_document[i])):
            keyword = encrypted_document[i][j]
            if keyword not in index:
                index[keyword] = []
            index[keyword].append(i)

    # 步骤五，检索包含指定keyword的文档
    query = encrypted_document[0][0]
    retrieved_documents = retrieve_documents(query, index)
    # 步骤六，解密已检索到的文档
    decrypted_documents = []
    for i in range(len(retrieved_documents)):
        decrypted_documents.append\
        (decrypt_document([encrypted_document[retrieved_documents[i]]], [trapdoors[retrieved_documents[i]]])[0])

    # 步骤七，打印原始文档和解密后的文档
```

```
89      print("原始文档:")
90      print(document)
91      print("我们要查询包含 %s 的文档" % decrypt_document(query, trapdoors
            [0][0])[0])
92      print("查询到的解密后的文档:")
93      print(decrypted_documents)
```