南開大學

《数据安全》课程实验报告

实验六：对称可搜索加密方案的实现

学　　院　网络空间安全学院
专　　业　信息安全
学　　号　2112060
姓　　名　孙蕗

目录

一、 实验目的

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、 实验原理

1. 可搜索加密

可搜索加密（Searchable Encryption，简称 SE）是一种密码原语，允许数据加密后仍能对密文数据进行关键字检索，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。

（1） 可搜索加密的子过程

可搜索加密可以分为 4 个子过程：

- 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
- 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，要求陷门不能泄露关键词的任何信息
- 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
- 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

（2） 可搜索加密的分类

可搜索加密可以分为对称可搜索加密和非对称可搜索加密

- 对称可搜索加密(Symmetric searchable encryption,SSE):加解密过程采用相同的密钥，陷门生成也需要密钥的参与，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。
- 非对称可搜索加密(Asymmetric searchable encryption,ASE):公钥用于对明文信息加密和目标密文的检索，私钥用于解密密文信息和生成关键词陷门。非对称可搜索加密算法通常较为复杂，加解密速度较慢，其公私钥相互分离的特点，非常适用于私钥生成待检索关键词陷门，可用于多用户同时进行关键词检索的场景等。

（3） 对称可搜索加密的定义

以下为对称可搜索加密的定义：

1. 形式化定义

定义在字典Δ= $\{W_1, W_2, \cdots, W_d\}$上的对称可搜索加密算法可描述为五元组：

$SSE = (KeyGen, Encrypt, Trapdoor, Search, Decrypt)$，其中，

- $\boldsymbol{K = KeyGen(\lambda)}$：输入安全参数$\lambda$，输出随机产生的密钥$K$；
- $\boldsymbol{(I, C) = Encrypt(K, D)}$：输入对称密钥$K$和明文文件集$D = (D_1, D_2, \cdots, D_n)$，输出索引$I$和密文文件集$C = (C_1, C_2, \cdots, C_n)$。对于无需构造索引的SSE方案，$I = \emptyset$；
- $\boldsymbol{T_W = Trapdoor(K, W)}$：输入对称密钥$K$和关键词$W$，输出关键词陷门$T_W$；
- $\boldsymbol{D(W) = Search(I, T_W)}$：输入索引$I$和陷门$T_W$，输出包含$W$文件的标识符集合$D(W)$；
- $\boldsymbol{D_i = Decrypt(K, C_i)}$：输入对称密钥$K$和密文文件$C_i$，输出相应明文文件$D_i$。

1. 基本定义

- **正确性**

如果对称可搜索加密方案SSE是正确的，

那么对于$KeyGen(\lambda)$和$Encrypt(K, D)$输出的K和(I, C)，都有

$$\boldsymbol{Search\big(I, Trapdoor(K, W)\big) = D(W)}$$

和$\boldsymbol{Decrypt(K, C_i) = D_i}$成立，

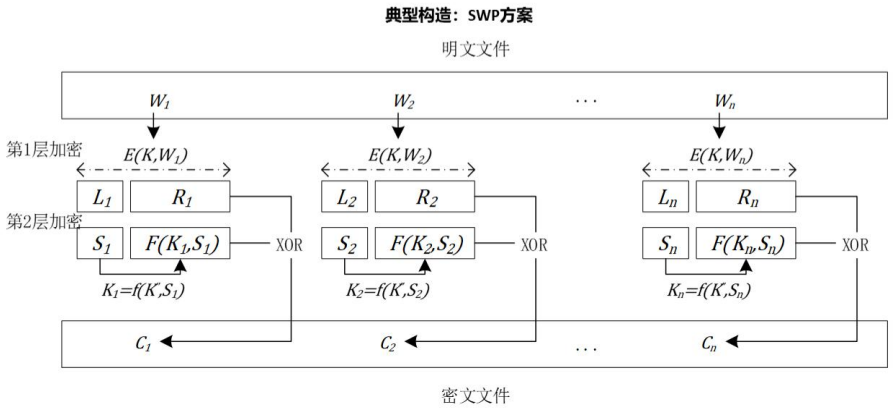
这里$C_i \in C, i = 1,2, \cdots, n.$

（4） 基于正向索引构造对称可搜索加密

基本构造思路是：

- 将文件进行分词；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，可以称这种方式为正向索引(和后面倒排索引进行区分)。2000 年，Dawn Song 所提出的 SWP 方案，就采用了这种方式。



优点：SWP 方案通过植入“单词”位置信息，能够支持受控检索(检索关键词的同时，识别其在文件中出现的位置)。例如，将所有“单词”以 W||a 形式表示，a 为 W 在文件中出现的位置，但查询时可增加对关键词出现位置的约束。

SWP 方案存在一些缺陷：

- 效率较低，单个单词的查询需要扫描整个文件，占用大量服务器计算资源；
- 在安全性方面存在统计攻击的威胁。例如，攻击者可通过统计关键词在文件中出现的次数来猜测该关键词是否为某些常用词汇。

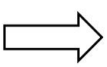
（5） 基于倒排索引构造对称可搜索加密

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(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 w_1 索引了一个列表，KeyValue 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。

Key	Value
w_1	$D_1, D_4, \ldots, D_{15}$
w_2	$D_1, D_3, \ldots, D_{22}$
⋮	⋮
w_m	$D_2, D_3, \ldots, D_n$



Key	Value
$E(w_1)$	$D_1, D_4, \ldots, D_{15}$
$E(w_2)$	$D_1, D_3, \ldots, D_{22}$
⋮	⋮
$E(w_m)$	$D_2, D_3, \ldots, D_n$

三、 实验过程

实验思路如下：

1. 对明文进行加密得到密文
2. 对密文进行陷门生成过程得到陷门
3. 将陷门存储在服务器上
4. 确定查询关键词，并对关键词进行陷门生成得到查询陷门
5. 将查询陷门发送到服务器
6. 服务器使用正向索引，在陷门库中查找与查询陷门匹配的陷门
7. 如果找到匹配的陷门，则返回响应的密文
8. 对密文进行解密得到明文
9. 如果明文包含查询关键词，则返回查询结果
10. 如果没有找到匹配的陷门，则返回无结果

1. 生成随机字符串

```python
# 生成随机定长字符串的函数
def generate_random_string(min_length, max_length):
    length = random.randint(min_length, max_length)
    letters = string.ascii_lowercase
    return ''.join(random.choice(letters) for i in range(length))
```

生成随机字符串，字符串由小写字母组成。字符串的长度在 min_length 到 max_length 不定。

2. 生成哈希值

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

为关键词 keyword 生成哈希值，以十六进制字符串的形式返回

3. 生成陷门

```python
# 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
```

获取关键词中每个字符的哈希值的第一个字符为关键词生成陷门

4. 使用陷门加密文档

```
# 加密文档
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
```

遍历文档中的每个单词，采用异或加密。将字符的 ASCII 值与 trapdoor 中对应位置的字符的 ASCII 值相加，并将结果转换为字符。这样，每个单词都会被加密为一个新的字符串，这些字符串被组合成一个加密后的文档。

5. 解密文档

```
# 解密文档
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
```

遍历文档中的每个单词，对单词中的每个字符进行解密，解密方法是将字符的 ASCII 值与 trapdoor 中对应位置的字符的 ASCII 值相减，并将结果转换为字符。这样，每个单词都会被解密为一个新的字符串，这些字符串被组合成一个解密后的文档。

6. 正向索引检索包含给定关键词的文档

```
# 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))
```

遍历检查关键字的每个字符是否在索引中，如果在索引中，获取包含该字符的文档的集合。然后，它返回所有包含所有关键字字符的文档的集合。

7. 实验实现

首先生成 10 个长度为 5-10 的字符串组成随机文档，并输出明文的原始文档

```
# 生成随机文档，长度不定
document = []
for i in range(10):
    document.append(generate_random_string(5, 10))  # 随机生成长度在5到10之间的字符串
```

```
print("原始文档:")
print(document)
```

```
原始文档:
['tvfuxpg', 'ruvbfxic', 'xhsjdblhoz', 'ehgctmd', 'pvjftaiclj', 'dzvata', 'gdruvo', 'kvmbizufs', 'szkjuifbo', 'zonvypg']
```

为文档中每个关键词生成陷门，使用陷门加密文档

```
# 为文档中每个keyword生成对应的陷门trapdoor
trapdoors = []
for i in range(len(document)):
    trapdoors.append(generate_trapdoor(document[i]))
```

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

构建正向索引，索引的键是加密后的字符，值是包含该字符的文档列表

```
# 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)
```

查询加密后的文档中的第一个字符，并使用索引找到包含该字符的文档

```
# 检索包含指定keyword的文档
query = encrypted_document[0][0]
retrieved_documents = retrieve_documents(query, index)
```

```
print("要查询包含 %s 的文档" % decrypt_document(query, trapdoors[0][0])[0])
```

```
要查询包含 t 的文档
```

解密检索到的文档

```
# 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])
```

```
# 打印原始文档和解密后的文档
print("原始文档:")
print(document)
print("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)
```

```
原始文档:
['tvfuxpg', 'ruvbfxic', 'xhsjdblhoz', 'ehgctmd', 'pvjftaiclj', 'dzvata', 'gdruvo', 'kvmbizufs', 'szkjuifbo', 'zonvypg']
查询包含 t 的文档
查询到的解密后的文档:
['tvfuxpg', 'ehgctmd', 'pvjftaiclj', 'dzvata']
```

四、 实验结论及心得体会

本次实验通过对称可搜索加密方案的模拟实现，我们成功地完成了加密、陷门生成、检索和解密四个过程。具体实现步骤如下：

我们首先生成了 10 个长度为 5-10 的随机字符串，组成了原始文档。

针对原始文档中的每个关键词，我们生成了对应的陷门，并使用陷门对文档进行加密。

构建了一个正向索引，将加密后的字符作为键，对应的文档列表作为值。

对于给定的查询关键词，我们通过索引检索到了包含该关键词的文档。

最后，我们对检索到的文档进行解密，得到了查询结果。

在完成这个实验的过程中，我学到了很多关于对称可搜索加密方案的知识，也深入理解了加密、索引和检索在数据安全中的作用和实现方式。

首先，我意识到了加密在保护数据隐私方面的重要性。通过对文档进行加密，即使数据存储在不可信的服务器上，也能够保证数据的机密性，不会被未经授权的用户访问或窃取。其次，索引的构建对于实现可搜索加密方案至关重要。在实验中，我了解到了正向索引和倒排索引两种构建方式，并了解到它们各自的优缺点。正向索引适用于少量关键词的检索，而倒排索引则适用于大规模文档的检索，并且能够提高检索效率。另外，我也学会了如何生成陷门并在检索时利用陷门进行查询。这个过程中需要注意的是，陷门的生成应当保证不泄露关键词的任何信息，以确保用户的隐私安全。通过这个实验，我对可搜索加密方案有了更深入的理解，也掌握了实现这种方案的基本技术和方法。这将对我今后在数据安全领域的学习和工作有所帮助。