实验六：对称可搜索加密方案的实现

2112852　密码科学与技术　胡佳佳

May 2024

1　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，能分别完成加密、陷门生成、检索和解密四个过程。本次实验是基于正向索引机制实现的。

2　实验原理

2.1　可搜索加密 [1]

关键词检索是一种常见的操作，比如数据库全文检索、邮件按关键词检索、在 Windows 系统里查找一个文件等。**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。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。

2.2　基于正向索引的构造

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”的方式组织文档与关键词的关系，我们称这种方式为正向索引。一个具体的正向索引机制的实现是 Dawn Song 在 2000 年提出的 SWP 方案 [2]（图 1）。

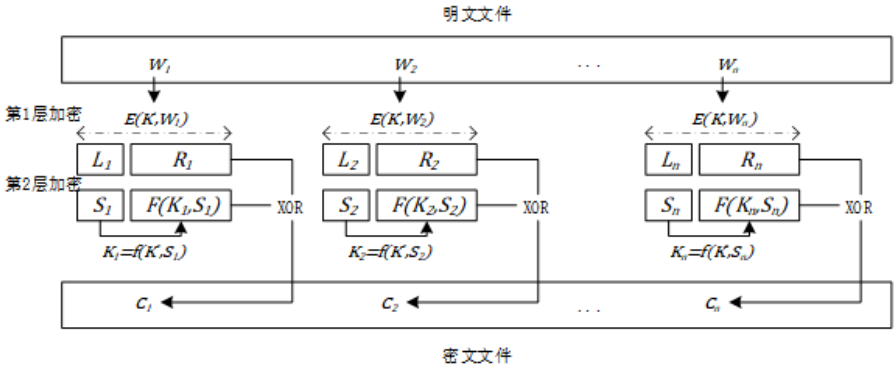

Figure 1: SWP 方案

SWP 方案在预处理过程中根据文件长度产生伪随机流 $S_1, S_2, ..., S_n$（n 为待加密文件中“单词”个数），然后采用两个层次加密：
在第 1 层，使用分组密码逐个加密明文文件单词；
在第 2 层，对分组密码输出 $E(K', W_i)$ 进行处理：

1. 将密文等分为 L_i 和 R_i 两部分；
2. 基于 L_i 生成二进制字符串 $S_i||F(K_i, S_i)$，这里，$K_i = f(K", L_i)$，$||$ 为符号串连接，F 和 f 为伪随机函数；
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。

查询文件 D 中是否包含关键词 W，只需发送陷门 $T_w = (E(K', W), K = f(K", L))$ 至服务器（L 为 $E(K', W_i)$ 的左部），服务器顺序遍历密文文件的所有单词 C，计算 $CXORE(K', W) = S||T$，判断 $F(K, S)$ 是否等于 T：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。
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 $W||\alpha$ 形式表示，α 为 W 在文件中出现的位置，仍按图 1所示加密，但查询时可增加对关键词出现位置的约束。

3 实验环境

IDE：VScode
编程语言：python 3.11.4

4 实验步骤

为了简化实验，我们把一个连续的字符串看作是文件（file），其中的每个字符看作是关键词 (keyword)，并随机生成了一组文档（doc, 由多个文件组成）进行简单模拟。

```
#1 生成随机文档
doc = []
for i in range(30):
    doc.append(generate_random_str(10))
```

4.1 加密过程

在加密部分，第一层仅使用了哈希来模拟；第二层仅用了偏移加密来进行模拟。

```
# 加密文档对应关键词
def encrypt_doc(doc, trapdoors):
    encrypted_doc = []
    for i in range(len(doc)):
        encrypted_word = []
        for j in range(len(doc[i])):
            encrypted_char = chr(ord(doc[i][j]) + ord(trapdoors[i][j]))
            encrypted_word.append(encrypted_char)
        encrypted_doc.append(''.join(encrypted_word))
    return encrypted_doc
```

4.2 陷门生成过程

我们通过对应关键词的哈希值产生相应的陷门。

```
# 为 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
```

4.3 检索过程

查找包含所有关键词的所有文件

```
# 查询正向索引，返回包含该 keyword 的文件
def retrieve_docs(keyword, index):
    docs = []
    for char in keyword:
        if char in index:
            docs.append(set(index[char]))
```

```
        if len(docs) == 0:
            return []
        else:
            return list(set.intersection(*docs))
```

4.4 解密过程

对应的解密过程：

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

4.5 综合

SWP 算法流程如下（2）：

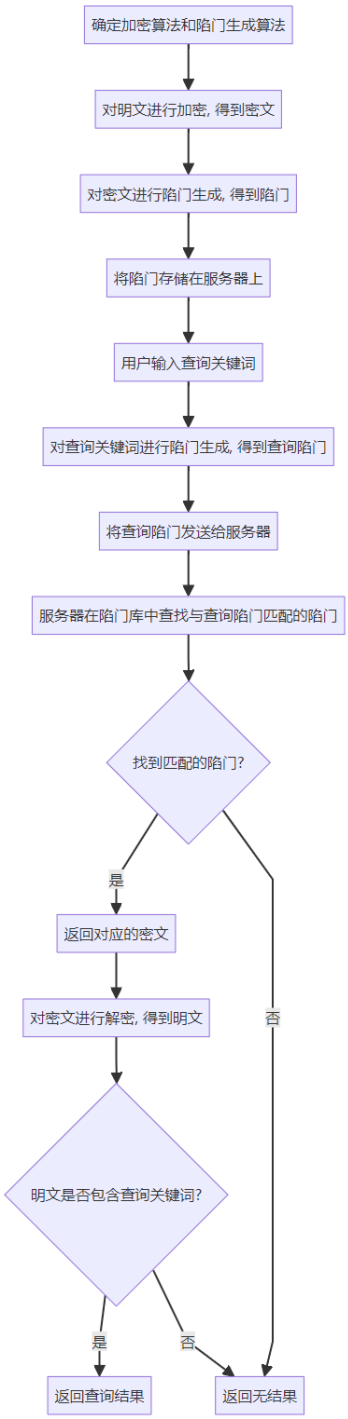
确定加密算法和陷门生成算法
对明文进行加密, 得到密文
对密文进行陷门生成, 得到陷门
将陷门存储在服务器上
用户输入查询关键词
对查询关键词进行陷门生成, 得到查询陷门
将查询陷门发送给服务器
服务器在陷门库中查找与查询陷门匹配的陷门
找到匹配的陷门?
是
否
返回对应的密文
对密文进行解密, 得到明文
明文是否包含查询关键词?
是
否
返回查询结果
返回无结果

Figure 2: 流程图

按照以上算法，我们有如下主函数：

```
if __name__ == "__main__":
'''
为了简化实验，我们把
一个连续的字符串看作是文件（file），
其中的每个字符看作是关键词（keyword），
并随机生成了一组文档（doc，由多个文件组成）进行简单模拟.

在加密部分，第一层仅使用了哈希来模拟；第二层仅用了偏移加密
'''
#1 生成随机文档
doc = []
for i in range(30):
    doc.append(generate_random_str(10))

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

#3 使用陷门加密文档
encrypted_doc = encrypt_doc(doc, trapdoors)

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

#5 检索包含指定 keyword 的文档
query = encrypted_doc[0][0]
retrieved_docs = retrieve_docs(query, index)

#6 解密已检索到的文档
decrypted_docs = []
for i in range(len(retrieved_docs)):
    decrypted_docs.append(
        decrypt_doc(
        [encrypted_doc[retrieved_docs[i]]],
        [trapdoors[retrieved_docs[i]]])[0]
    )

#7 结果输出和测试
```

```
    print(" 客户端明文文件（每个连续字符串代表一篇文件）:")
    print(doc)
    print(" 查询包含 %s 的文件" % decrypt_doc([query], [trapdoors[0]])[0])
    print(" 查询到文件解密后:")
    print(decrypted_docs)
```

4.6 运行结果

```
PS C:\Users\Nutcracker> & D:/packages/anacondaPkg/python.exe h:/Desktop/debug.py
客户端明文文档（每个连续字符串代表一篇文档）:
['bzlzkhldps', 'uftrjixeug', 'sucyazigen', 'xaklkvdvwx', 'wvpiskshbe', 'dvjjnifvlk', 'rfseueyhtl',
 'fktnsuctsw', 'noebmtrzxq', 'exfwnwsmck', 'jlkbhqgidl', 'lytctfvkpv', 'tojrmeqiae', 'fjpgozzmdu',
 'usskxkwkjj', 'wpcbpxvcvg', 'mlftnlyckc', 'vfkhamsxcr', 'fnrwtazumw', 'tczziihsmn', 'crfsxxhecl',
 'bpqwzlyqqf', 'jwsyapqowl', 'zbxolmlqqd', 'nzrhzncyio', 'rtylhppozs', 'bdgoxofwlk', 'clbfyfyyow',
 'pggnkvhanq', 'sicgwkxtyf']
查询包含 b 的文档
查询到文档解密后:
['bzlzkhldps', 'sucyazigen', 'xaklkvdvwx', 'wvpiskshbe', 'dvjjnifvlk', 'fktnsuctsw', 'noebmtrzxq',
 'exfwnwsmck', 'jlkbhqgidl', 'lytctfvkpv', 'fjpgozzmdu', 'wpcbpxvcvg', 'mlftnlyckc', 'vfkhamsxcr',
 'tczziihsmn', 'crfsxxhecl', 'bpqwzlyqqf', 'zbxolmlqqd', 'nzrhzncyio', 'bdgoxofwlk', 'clbfyfyyow',
 'sicgwkxtyf']
```

Figure 3: 运行结果截图

可以发现，在服务器不知道关键词的前提下，利用客户端提供的陷门集，该程序成功筛出所有包含关键词的文件，并返回客户端成功解密。

5 实验总结

本实验成功地根据正向索引机制，提供一种可搜索加密方案的基础模拟实现，能分别完成加密、陷门生成、检索和解密四个过程，验证了 SWP 可搜索加密方案的正确性。而进一步，我们也可以考虑到 SWP 方案存在一些缺陷：

1. 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；
2. 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。

为此，对于第一点而言，可以采用倒排索引进一步加速检索过程；至于可能存在的统计猜测攻击，我们可以考虑在特定范围内一些分布较均匀的概率分布对关键词进行进一步地混淆来抵御攻击。

References

[1] 刘哲理，“数据安全课程实验教材”，2024，第七章

[2] Dawn Xiaoding Song, D. Wagner and A. Perrig, "Practical techniques for searches on encrypted data," Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000, Berkeley, CA, USA, 2000, pp. 44-55, doi: 10.1109/SECPRI.2000.848445.