对称可搜索加密方案实现实验报告

- 姓名：陈睿颖
- 学号：2013544
- 专业：计算机科学与技术

1. 实验内容

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。

2. 实验原理

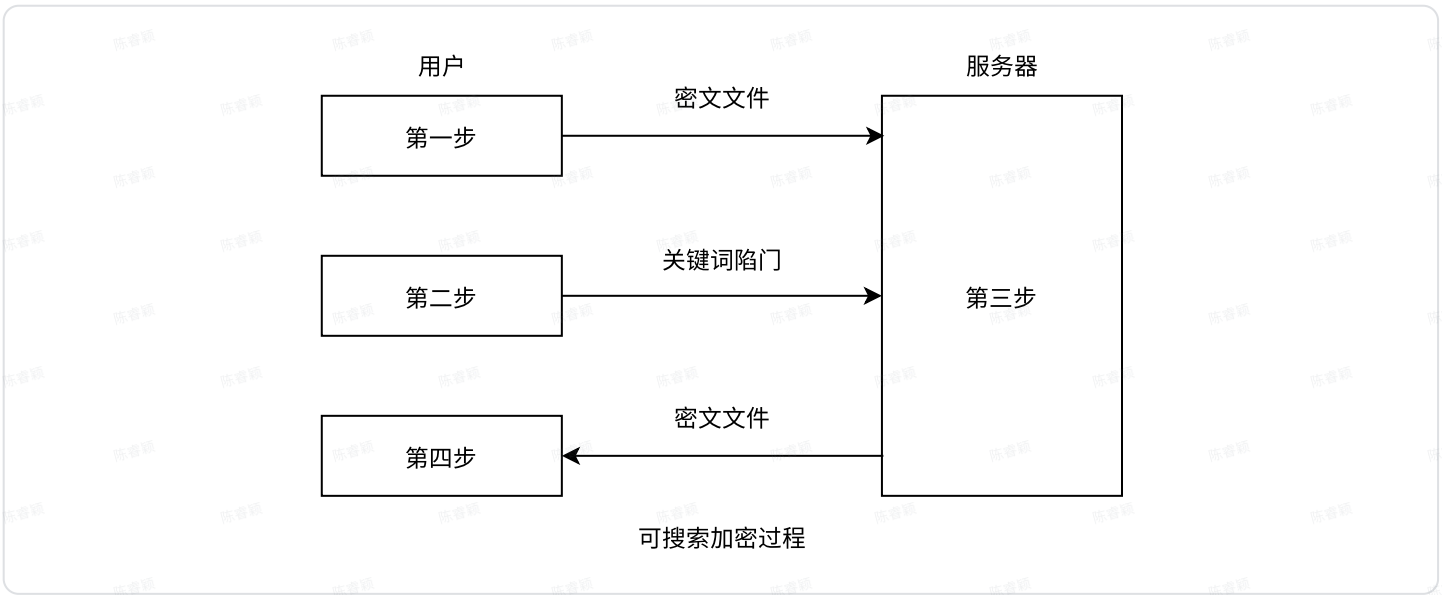
2.1 对称可搜索加密

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。

可分为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（也可以称为令牌），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。



2.2 GCM加密模式

GCM（Galois/Counter Mode）是一种加密模式，通常用于加密传输层安全协议（TLS）和互联网协议安全协议（IPsec）中。GCM加密模式具有高效、强安全性和高度可扩展性等特点，在保证传输数据安全的同时，也能够提供数据完整性和认证。

GCM加密模式将CTR模式和GMAC算法结合起来，其中CTR模式用于加密，GMAC算法用于计算消息认证码（MAC）。GCM加密模式的加密过程如下：

1. 选择一个加密密钥K，用于对明文进行加密。
2. 选择一个随机的初始化向量（IV），与明文一起输入GCM算法。
3. 使用CTR模式将明文加密得到密文。CTR模式中的计数器值每次递增，以确保每个加密块都有不同的密钥流。
4. 使用GMAC算法计算MAC。GMAC算法使用Galois域运算来计算MAC，其中密钥流被用作Galois域的元素。GMAC算法不需要额外的传输步骤来传输MAC，因为MAC已经被附加到密文中。
5. 返回密文和MAC。

解密过程与加密过程类似，只需要使用相同的加密密钥K和IV，并使用CTR模式解密密文。在解密过程中，可以使用计算得到的MAC来验证密文的完整性和认证消息的发送者。

3. 代码设计

3.1 加密

该部分代码实现如下：

```
    def _encrypt_document(self, document):
        """
        使用AES GCM模式加密文档并返回密文、初始化向量和认证标签
        """
        # 生成随机初始化向量
        iv = os.urandom(AES.block_size)

        # 使用GCM模式加密文档
        cipher = AES.new(self.key, AES.MODE_GCM, nonce=iv)
        cipher_text, tag = cipher.encrypt_and_digest(pad(document.encode('utf-8'), AES.block_size))

        return cipher_text, iv, tag
```

主要实现步骤：

1. 随机生成初始化向量（iv）：在加密过程中，初始化向量用于提高加密的安全性和随机性。
2. 使用AES GCM模式加密文档：使用AES算法和GCM模式加密文档。这里使用了AES.block_size指定块大小，以确保密文长度与原始文本长度匹配。同时，使用了cipher.encrypt_and_digest()方法对文档进行加密和摘要处理，生成密文（cipher_text）和认证标签（tag）。
3. 返回加密结果：返回加密后的密文、初始化向量和认证标签。这些信息将用于解密和验证文档的完整性

3.2 陷门生成

该部分实现生成查询陷门并加密的过程。该方案基于加密索引，其中查询陷门可以被用于在加密索引中查找包含关键词的文档ID集合，而同时保护了用户的查询隐私。

在具体实现中，`_generate_trapdoor` 函数接受一个查询字符串并生成对应的查询陷门。该函数遍历查询中的每个关键词，并查找包含这些关键词的文档ID集合，最终将这些集合组成一个字典，用 SHA256 哈希函数将其压缩为一个 128 位的陷门。

`generate_and_encrypt_trapdoor` 函数调用 `_generate_trapdoor` 函数生成查询陷门，然后使用 `pickle` 将其序列化为一个字节串。接下来，函数使用与之前加密文档相同的 AES GCM 模式和密钥加密查询陷门，并返回密文、初始化向量和认证标签。

```
    def _generate_trapdoor(self, query):
        """
        根据查询语句生成查询陷门
        """
        # 查询每个关键词对应的加密文档ID
        trapdoor = {}
        for keyword in query.split():
            if keyword in self.inverted_index:
                trapdoor[keyword] = self.inverted_index[keyword]
            else:
                trapdoor[keyword] = set()

        # 使用SHA256哈希函数计算加密文档ID的集合为一个128位的陷门
        hash_object = SHA256.new(data=str(trapdoor).encode('utf-8'))
        return hash_object.digest()


    def generate_and_encrypt_trapdoor(self, query):
        """
        生成查询陷门并加密
        """
        trapdoor = self._generate_trapdoor(query)

        # 将查询陷门序列化为字节串
        trapdoor_bytes = pickle.dump(trapdoor)

        # 生成随机初始化向量
        iv = os.urandom(AES.block_size)

        # 使用GCM模式加密查询陷门
        cipher = AES.new(self.key, AES.MODE_GCM, nonce=iv)
        cipher_text, tag = cipher.encrypt_and_digest(pad(trapdoor_bytes,
AES.block_size))

        return cipher_text, iv, tag
```

3.3 检索

在进行检索前，要先建立倒排索引：

```
    def build_index(self, documents):
        """
        建立加密的倒排索引并返回加密后的文档和倒排索引
```

```
        """
        # 生成密钥
        self.key = os.urandom(16)

        # 初始化倒排索引
        self.inverted_index = {}

        # 加密文档
        self.encrypted_documents = []
        for document in documents:
            # 加密文档并存储加密后的文档、初始化向量和认证标签
            self.encrypted_documents.append(self._encrypt_document(document))

            # 更新倒排索引
            for keyword in document.split():
                if keyword not in self.inverted_index:
                    self.inverted_index[keyword] = set()
                self.inverted_index[keyword].add(len(self.encrypted_documents)
- 1)

        # 返回加密后的文档和倒排索引
        return self.encrypted_documents, self.inverted_index, self.key

```

下面进行检索函数的代码编写，用于在加密的文档集合中搜索与查询字符串匹配的文档。它可以接受一个查询字符串和一个可选的查询陷门作为参数。

当没有提供查询陷门时，它会使用查询字符串生成一个查询陷门，然后返回这个查询陷门，该查询陷门是一个字典，其键是查询字符串中的每个关键词，值是与每个关键词相关联的加密文档ID集合。

当提供了查询陷门时，它将使用查询陷门从加密的文档集合中检索相关文档ID，然后对这些文档进行解密，并将它们添加到结果列表中返回。具体来说，对于每个文档ID，它会使用相应的初始化向量和认证标签，解密加密文本，并使用AES解密算法和GCM模式进行解密。最后，将解密的文本添加到结果列表中，返回解密后的文本列表。

```
    def search_index(self, query, trapdoor=None):
        """
        用GCM加密模式解密相关文档并返回
        """
        # 检索相关文档ID
        related_document_ids = set()
        if trapdoor is None:
            # 生成查询陷门
            trapdoor = {}
            for keyword in query.split():
```

```
                trapdoor[keyword] = set()
                if keyword in self.inverted_index:
                    trapdoor[keyword] = self.inverted_index[keyword]
            return trapdoor
        else:
            for keyword in trapdoor:
                if keyword in self.inverted_index:
                    related_document_ids |= trapdoor[keyword]

        # 解密相关文档并返回
        decrypted_documents = []
        for document_id in related_document_ids:
            # 解密文档
            cipher_text, iv, tag = self.encrypted_documents[document_id]
            cipher = AES.new(self.key, AES.MODE_GCM, nonce=iv)
            output = bytearray(len(cipher_text))
            plaintext = unpad(cipher.decrypt_and_verify(cipher_text, tag,
output=output), AES.block_size).decode('utf-8')

            # 添加到结果列表
            decrypted_documents.append(plaintext)

        # 返回解密后的文档列表
        return decrypted_documents
```

3.4 解密

使用方法 `decrypt_trapdoor`，用于解密查询陷门并反序列化为 Python 对象。

具体实现步骤如下：

1. 使用给定的密钥和随机初始化向量以 GCM 模式初始化一个 AES 对象 `cipher`。
2. 使用 `cipher` 对象解密给定的密文 `cipher_text`，并校验认证标签 `tag`，得到解密后的字节串。
3. 对解密后的字节串进行反序列化，得到 Python 对象 `trapdoor`。
4. 返回 Python 对象 `trapdoor`。

```
    def decrypt_trapdoor(self, cipher_text, iv, tag):
        """
        解密查询陷门并反序列化为Python对象
        """
        cipher = AES.new(self.key, AES.MODE_GCM, nonce=iv)
```

```
        output = bytearray(len(cipher_text))
        trapdoor_bytes = unpad(cipher.decrypt_and_verify(cipher_text, tag, output=output), AES.block_size)
        trapdoor = pickle.load(trapdoor_bytes)
        return trapdoor
```

4. 实验结果

使用如下代码进行测试：

```
#测试
documents = ['Hello world', 'I am chenruiying', 'I love nku',
             'apple', 'banana', 'orange', 'hahahaha', 'kiwi', 'pepper',
             'nonono I hate the apple',
             'apple', 'banana', 'orange', 'haha', 'ki',
             'cannot think more about this','everything is fine!']
encrypted_index = EncryptedIndex()

# 建立加密索引
encrypted_documents, inverted_index, key = encrypted_index.build_index(documents

# 为查询构建陷门
query = 'I'
cipher = AES.new(key, AES.MODE_ECB)
trapdoor = cipher.encrypt(pad(query.encode('utf-8'), AES.block_size))

# 使用陷门进行检索
decrypted_documents = []
for document_id, (cipher_text, iv, tag) in enumerate(encrypted_documents):
    # 解密陷门
    cipher = AES.new(key, AES.MODE_ECB)
    decrypted_trapdoor = cipher.decrypt(trapdoor)

   #计算密文和陷门之间是否匹配
    cipher = AES.new(key, AES.MODE_GCM, nonce=iv)
    output = bytearray(len(cipher_text))
    score = output == decrypted_trapdoor

    # 匹配成功，将密文加入到解密文档中
    if score:
        decrypted_documents.append(documents[document_id])

# 打印解密文档
```

```
print(decrypted_documents)
```

应返回所有含“I”的字符串。运行结果如下：

```
PS D:\MyFiles\Data Security\对称可搜索加密实现> python3 .\SSE.py
['I am chenruiying', 'I love nku', 'nonono I hate the apple']
PS D:\MyFiles\Data Security\对称可搜索加密实现>
```

实验成功！

5. 反思体会

5.1 优缺点分析

优点：

- 数据和查询都被加密，保护了隐私和安全。
- 使用AES GCM加密模式进行加密和认证，提供了机密性和完整性保护。
- 生成查询陷门使用了SHA256哈希函数，生成的陷门可以代表查询语句，但不会泄露原始查询语句的信息。
- 使用了倒排索引的方式来存储关键词，可以提高检索速度。

缺点：

- 使用pickle库来序列化查询陷门，pickle可以带来安全问题，因为它可以序列化包含可执行代码的对象。
- 使用固定长度的密钥（16字节），可能会受到暴力攻击或密码分析攻击。
- 存储索引和文档的空间开销大。
- 由于AES GCM模式需要访问整个加密文档，因此检索和解密速度较慢。如果文档很大，可能会导致性能问题。
- 查询陷门不支持模糊匹配或通配符查询，只能匹配完全匹配的关键字。

5.2 心得收获

- 学到了如何使用Python中的 `pickle` 模块来对Python对象进行序列化和反序列化，以及如何使用Python中的 `Crypto` 模块来实现对文档和查询结果的加密和解密，以保护敏感信息的安全性。
- 学习了如何使用Python中的 `os` 模块来生成随机数，以及如何使用Python中的哈希函数来对陷门进行计算。
- 学习了如何使用AES加密算法和GCM模式来实现加密和解密，并使用填充来确保数据长度的正确性，以及使用认证标签来确保数据的完整性和真实性。