南开大学
Nankai University

南　　开　　大　　学

网络空间安全学院

数据安全实验报告

实验 6：对称可搜索加密方案实现

姓名：付政烨

学号：2113203

年级：2021 级

专业：信息安全、法学

2024 年 5 月 24 日

目录

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、 实验内容

(一) 程序执行流

本次实验实现了了一个名为 DocumentSecurity 的 Python 类（详见附件“实验源代码”），其旨在提供文档安全的功能。首先，该类导入了操作系统相关模块（os），密码学相关模块（cryptography.hazmat），以及生成随机字符串所需的模块（random 和 string）。在初始化方法（init）中，使用默认的后端和生成的随机密钥与随机初始化向量初始化了类的属性。此外，类还包括了生成随机字符串、加密和解密文本、对文档中关键词进行加密和解密、创建正向索引以及检索文档等方法。在主程序中，创建了 DocumentSecurity 类的实例，并生成了一个包含随机字符串的文档，并对其进行加密和创建索引。最后，以第一个加密关键词作为查询条件，检索包含该关键词的文档，并对其进行解密。整个过程通过自定义的输出函数了不同部分的输出结果，包括原始文档、查询关键词、解密后的关键词以及解密后的文档。

(二) 加密 (Encrypt)

加密过程是将明文转换为无法直接阅读的密文，以保护数据的机密性。在此代码中，使用 AES 加密算法以及 CBC 模式实现了文档的加密。AES 是一种广泛使用的对称加密算法，需要一个密钥和一个初始化向量（IV）来进行加密。

- **密钥 (key)**: 用于加密和解密的密钥，此处生成了一个 256 位的随机密钥。
- **初始化向量 (iv)**: CBC 模式下需要一个随机的初始化向量，用于加密的第一个块，增强加密的安全性。

```
1 def encrypt(self, plaintext):
2     padder = padding.PKCS7(algorithms.AES.block_size).padder()
3     padded_data = padder.update(plaintext.encode()) + padder.finalize()
4     cipher = Cipher(algorithms.AES(self.key), modes.CBC(self.iv), self.backend)
5     encryptor = cipher.encryptor()
6     return encryptor.update(padded_data) + encryptor.finalize()
```

加密方法（encrypt）执行以下步骤：

1. 使用 PKCS7 填充方式将明文进行填充，确保数据块大小正确。
2. 创建一个 Cipher 实例，用指定的密钥和 IV 进行 AES 加密。
3. 通过 Cipher 的加密器（encryptor），将填充后的数据转换为密文。

(三) 陷门生成 (Create Index)

陷门函数是搜索加密数据时使用的一种方法，它能够在不解密数据的情况下进行索引和查询。这里的“陷门”是通过对加密后的文档创建索引来实现的。创建索引（create_index）的过程如下：

```
def create_index(self, encrypted_document):
    index = {}
    for i, encrypted_word in enumerate(encrypted_document):
        word = self.decrypt(encrypted_word).decode()
        for char in word:
            if char not in index:
                index[char] = []
            index[char].append(i)
    return index
```

1. 对每个加密的单词解密，得到明文单词。
2. 遍历每个单词的字符，将字符及其在文档中的位置记录在索引字典中。这样，每个字符都关联到包含该字符的所有单词的位置。

(四) 检索 (Retrieve Documents)

检索功能通过陷门（索引）快速找到包含特定关键词的文档。这在加密的环境中尤为重要，因为不能直接在加密文本上执行搜索。

```
def retrieve_documents(self, keyword, index):
    documents_sets = [set(index[char]) for char in keyword if char in index]
    return list(set.intersection(*documents_sets)) if documents_sets else []
```

1. 对每个查询关键词的字符，查找其在索引中的出现位置。
2. 使用集合交集操作找出同时包含所有查询关键词字符的文档索引。

(五) 解密 (Decrypt)

解密是将密文转换回原始的明文。这一过程使用与加密相同的密钥和 IV。

```
def decrypt(self, ciphertext):
    cipher = Cipher(algorithms.AES(self.key), modes.CBC(self.iv), self.backend)
    decryptor = cipher.decryptor()
    padded_plaintext = decryptor.update(ciphertext) + decryptor.finalize()
    unpadder = padding.PKCS7(algorithms.AES.block_size).unpadder()
    return unpadder.update(padded_plaintext) + unpadder.finalize()
```

解密方法（decrypt）执行以下步骤：

1. 创建一个 Cipher 实例，与加密过程相同，但使用解密器（decryptor）。
2. 输入密文，通过解密器将其转换为填充后的明文。

3. 使用 PKCS7 解填充器移除填充，得到原始明文数据。

通过以上步骤，系统能够安全地处理文档的存储和检索，而用户只能通过合法的加密和解密过程访问原始数据。这保证了数据的机密性和完整性，同时支持基于关键词的高效检索。

三、 实验结果

下述实验结果展现了一个文档安全系统，其中文档被加密，并且可以根据加密关键词检索相应的文档。运行结果表明了系统的一些基本功能和操作流程。首先，系统生成了一些随机的五个字符长度的文档，然后对这些文档进行加密。接着，系统选择了第一个加密后的关键词作为查询条件，并且对这个关键词进行了解密操作。使用解密后的关键词在文档中检索，找到包含该关键词的文档。最后，将检索到的文档进行解密操作，并输出结果。

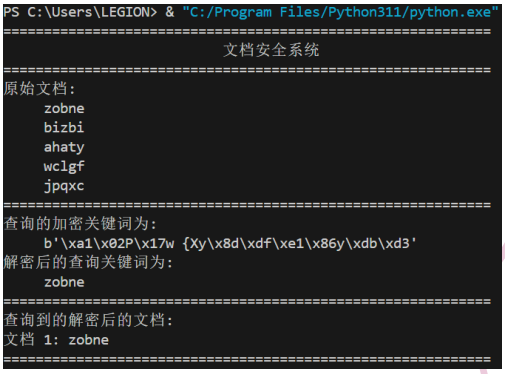

图 1: 结果 1

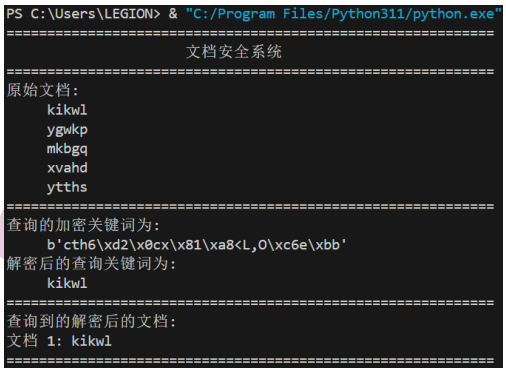

图 2: 结果 2

查询的加密关键词是由代码随机生成的，然后进行加密处理的，因此在每次运行代码时，生成的关键词和加密结果都会不同。解密后的查询关键词与原始文档中的某一个字符串相匹配，这是因为查询关键词是从加密文档中选取的。查询到的解密后的文档中包含了具体的文档内容，这些文档与原始文档中的某一个字符串完全相同，表明系统成功地根据查询关键词找到了对应的文档，并将其解密输出。**所以，结果表明了系统在加密、查询和解密过程中的正确执行**。

四、 实验心得与体会

这个实验让我深入理解了可搜索加密方案的原理和实现。通过实现加密、陷门生成、检索和解密四个过程，我对数据安全的保护和检索有了更深入的了解。

首先，在加密过程中，我学会了如何使用 AES 加密算法和 CBC 模式来保护数据的机密性。AES 是一种强大的对称加密算法，而 CBC 模式能够增强加密的安全性。我了解到在加密过程中，密钥和初始化向量的选择对加密结果的安全性至关重要。其次，在陷门生成阶段，我学会了如何利用索引的方式在加密文档中进行快速搜索。通过创建正向索引，我可以在不解密数据的情况下，快速地找到包含特定关键词的文档。这种方式有效地保护了数据的隐私性，同时又保证了搜索的高效性。然后，在检索阶段，我学会了如何利用生成的陷门来进行文档检索。通过对加密关键词的解密和索引的查询，我可以准确地找到包含特定关键词的文档。这种检索方式不仅保护了数据的安全性，还提高了搜索的速度和效率。最后，在解密阶段，我学会了如何将密文转换回原始的明文。通过使用与加密相同的密钥和初始化向量，我可以安全地解密数据，并获取原始的文档内容。这种解密方式保证了数据的完整性和可读性。

本次实验让我对数据安全和检索有了更深入的了解，同时也提高了我的编程能力和实践经验。通过实现可搜索加密方案，我不仅学会了理论知识，还掌握了实际操作的技能，这对我今后的学习和工作都将有很大的帮助。

附 件

实验源代码

```python
import os
import random
import string
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import padding

class DocumentSecurity:
    def __init__(self, key=None):
        self.backend = default_backend()
        self.key = key or os.urandom(32) # 生成一个256位随机密钥
        self.iv = os.urandom(16) # 生成一个随机初始化向量
        self.letters = string.ascii_lowercase # 定义字母表

    def generate_random_string(self, length):
        """生成指定长度的随机字符串。"""
        return ''.join(random.choice(self.letters) for _ in range(length))

    def encrypt(self, plaintext):
        """使用AES加密文本。"""
        padder = padding.PKCS7(algorithms.AES.block_size).padder()
        padded_data = padder.update(plaintext.encode()) + padder.finalize()
        cipher = Cipher(algorithms.AES(self.key), modes.CBC(self.iv), self.backend)
        encryptor = cipher.encryptor()
        return encryptor.update(padded_data) + encryptor.finalize()

    def decrypt(self, ciphertext):
        """使用AES解密文本。"""
        cipher = Cipher(algorithms.AES(self.key), modes.CBC(self.iv), self.backend)
        decryptor = cipher.decryptor()
        padded_plaintext = decryptor.update(ciphertext) + decryptor.finalize()
        unpadder = padding.PKCS7(algorithms.AES.block_size).unpadder()
        return unpadder.update(padded_plaintext) + unpadder.finalize()

    def encrypt_document(self, document):
        """对文档中的每个关键字使用AES加密。"""
        return [self.encrypt(word) for word in document]

    def decrypt_document(self, encrypted_document):
        """解密加密文档中的每个关键字。"""
```

```
41          return [self.decrypt(word).decode() for word in encrypted_document]
42
43      def create_index(self, encrypted_document):
44          """为加密文档创建一个前向索引，将每个字符映射到文档索引中。"""
45          index = {}
46          for i, encrypted_word in enumerate(encrypted_document):
47              word = self.decrypt(encrypted_word).decode()  # 解密以用于索引
48              for char in word:
49                  if char not in index:
50                      index[char] = []
51                  index[char].append(i)
52          return index
53
54      def retrieve_documents(self, keyword, index):
55          """通过索引检索包含特定关键字的文档。"""
56          documents_sets = [set(index[char]) for char in keyword if char in index]
57          return list(set.intersection(*documents_sets)) if documents_sets else []
58
59  def print_divider():
60      print("=" * 60)
61
62  def print_title(title):
63      print_divider()
64      print(f"{title:^50}")
65      print_divider()
66
67  def print_document(document):
68      print("原始文档:")
69      for word in document:
70          print(f"{' '*5}{word}")
71      print_divider()
72
73  def print_query_info(query, decrypted_query):
74      print("查询的加密关键词为:")
75      print(f"{' '*5}{query}")
76      print("解密后的查询关键词为:")
77      print(f"{' '*5}{decrypted_query}")
78      print_divider()
79
80  def print_retrieved_documents(decrypted_documents):
81      print("查询到的解密后的文档:")
82      for i, doc in enumerate(decrypted_documents):
83          print(f"文档 {i+1}: {doc}")
84      print_divider()
```

```
85
86  if __name__ == "__main__":
87      ds = DocumentSecurity()
88
89      # 生成随机文档
90      document = [ds.generate_random_string(5) for _ in range(5)]
91      encrypted_document = ds.encrypt_document(document)
92
93      # 创建前向索引
94      index = ds.create_index(encrypted_document)
95
96      # 检索文档
97      query = encrypted_document[0] # 假设我们查询第一个加密的单词
98      decrypted_query = ds.decrypt(query).decode()
99      retrieved_documents = ds.retrieve_documents(decrypted_query, index)
100
101     # 解密检索到的文档
102     decrypted_documents = [ds.decrypt_document([encrypted_document[i]])[0]
103                            for i in retrieved_documents]
104
105     # 输出美化的信息
106      print_title ("文档安全系统")
107     print_document(document)
108     print_query_info(query, decrypted_query)
109     print_retrieved_documents(decrypted_documents)
```