
南開大學
Nankai University

# 南　开　大　学

## 网络空间安全学院

# 数据安全实验报告

---

## 对称可搜索加密方案实现

---

2111408 周钰宸

年级：2021 级

专业：信息安全

指导教师：刘哲理

2024 年 5 月 20 日

# 摘要

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。

本次实验首先基于正向索引的对称可搜索加密的密码原语，选择了对 Representation Engineering: A Top-Down Approach to AI Transparency[1] 这一论文的效果进行了复现。将其具体应用到了安全邮件系统中的加密存储、关键词搜索和解密场景下。复现效果能够完整包括加密、陷门生成、检索和解密四个过程。

在这之后我又基于倒排索引机制的对称可搜索加密的密码原语，对加密、陷门生成、检索和解密四个过程进行了简单的复现。

最后对实验结果进行了详尽的分析，确保其完成了 SE 所需的安全性能与效果，对其进行了简单评估。



**关键字：数据安全　对称可搜索加密　正向索引　SWP 方案**


# 目录

# 一、 实验要求

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。

虽然上述实验要求仅要求对正向索引和倒排索引二选一，但我对两种实现方式都进行了模拟。

本次实验参考上述实验要求，首先选择了**正向索引机制的可搜索加密（SE）的开山之作：** Representation Engineering: A Top-Down Approach to AI Transparency[1]. 基于 Dawn Song 等人的工作，本次实验基于其研究成果进行了复现，将其具体应用到了**安全邮件系统中**的加密存储、关键词搜索和解密场景下。**能够完整地模拟加密、陷门生成、检索和解密四个过程**。

而后我基于**倒排索引机制的可搜索加密（SE）**进行了简单的代码实现，也**能够完整地模拟加密、陷门生成、检索和解密四个过程**。

# 二、 实验原理

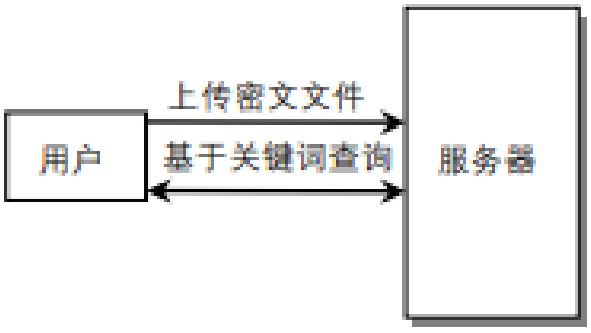
## （一） 可搜索加密

可搜索加密（Searchable Encryption, SE）是一种加密技术，允许用户在不解密的情况下对加密数据进行关键词搜索。其核心思想是在保证数据机密性的同时，提供一定程度的搜索功能。可搜索加密在保护数据隐私和安全的同时，提高了数据的可用性和便利性。

可搜索加密按照**按照应用模型分类**，主要分为四类：单用户—单服务器模型，多用户—单接收者（单服务器）模型，单用户—多接收者（单服务器）模型以及多用户—多接收者（单服务器）模型

### 1. 单用户—单服务器模型

在单用户—单服务器模型中，用户加密个人文件并将其存储到不可信的服务器。只有该用户具备基于关键词检索的能力，服务器无法获取明文文件和待检索关键词的信息。




(a) 单用户单服务器模型

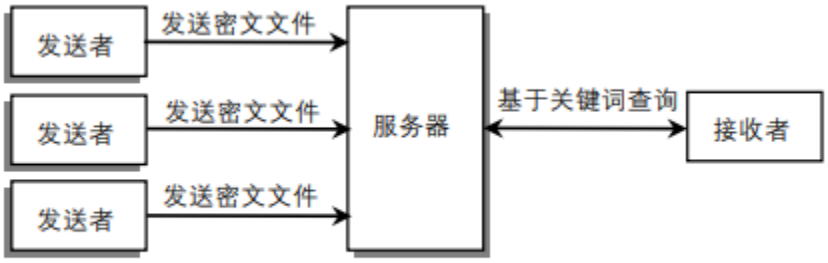
图 1

NKU

具体流程如图1所示。

**2. 多用户—单接收者（单服务器）模型**

在多用户—单接收者模型中，多个发送者加密文件后，将其上传至不可信的服务器，以达到与单个接收者传送数据的目的。只有接收者具备基于关键词检索的能力，服务器无法获取明文文件信息。与单用户模型不同，多用户—单服务器模型要求发送者和接收者不能是同一用户。



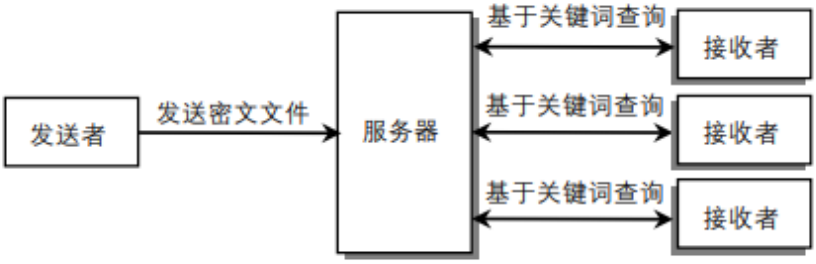

(b) 多对一单服务器模型

图 2

具体流程如图2所示。

**3. 单用户—多接收者（单服务器）模型**

在单用户—多接收者模型中，单个发送者将加密文件上传至不可信服务器，然后多个接收者共享数据。此模型适用于数据共享场景，如云存储中的共享文件。




(c) 一对多单服务器模型

图 3

具体流程如图3所示。

**4. 多用户—多接收者（单服务器）模型**

在多用户—多接收者模型中，任意用户都可成为接收者，通过访问控制和认证策略后，具备关键词的密文检索方式提取共享文件的能力。只有合法的用户具备基于关键词检索的能力，服务器无法获取明文文件信息，具备广阔的应用前景。


(d) 多用户单服务器模型

图 4

如图4所示，**作为本次实验的重点，其具体分为四个过程：加密过程，陷门生成过程，检索过程和解密过程**。具体过程如下：

1. **（1）加密过程**：
   - 每个用户生成自己的加密密钥 $K_i$。
   - 用户利用 $K_i$ 对文件进行加密，生成密文 $C_i$。
   - 用户将加密后的文件 $C_i$ 上传至服务器。
2. **（2）陷门生成过程**：
   - 任意合法接收者对关键词 $w$ 进行加密，生成加密关键词 $w'$，即生成关键词的陷门（令牌）。
   - 陷门 $w'$ 不能泄露关键词 $w$ 的任何信息。
3. **（3）检索过程**：
   - 接收者向服务器提交加密关键词 $w'$。
   - 服务器以关键词陷门 $w'$ 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 $C_i$。
   - 服务器只能判断密文文件是否包含某个特定关键词，而无法获得更多信息。
4. **（4）解密过程**：
   - 接收者使用密钥 $K_i$ 解密服务器返回的密文文件 $C_i$，获得查询结果。

## （二） 对称可搜索加密

可搜索加密（SE）如果按照使用的**密码原语分类**，可以分为对称可搜索加密（Symmetric Searchable Encryption, SSE)和非对称可搜索加密(Public Key Encryption with Keyword Search, PEKS)。

在这里将会重点介绍**对称可搜索加密（SE）的相关原理，为后面实验的进行做好充分准备**。

对称可搜索加密（Symmetric Searchable Encryption, SSE）是一种在对称加密框架下实现可搜索功能的加密方法。SSE 允许数据拥有者在对数据进行加密的同时，仍然能够对加密数据进行关键词搜索。其定义和基本流程如下：

NKU

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

$$\texttt{SSE} = (\texttt{KeyGen}, \texttt{Encrypt}, \texttt{Trapdoor}, \texttt{Search}, \texttt{Decrypt})$$

其中:

- $K = \texttt{KeyGen}(\lambda)$: 输入安全参数 $\lambda$, 输出随机产生的密钥 $K$;
- $(I, C) = \texttt{Encrypt}(K, D)$: 输入对称密钥 $K$ 和明文文件集 $D = \{D_1, D_2, \ldots, D_n\}$, 输出索引 $I$ 和密文文件集 $C = \{C_1, C_2, \ldots, C_n\}$。对于无索引的 SSE 方案, $I = \emptyset$;
- $T_w = \texttt{Trapdoor}(K, W)$: 输入对称密钥 $K$ 和关键词 $W$, 输出关键词陷门 $T_w$;
- $D(W) = \texttt{Search}(I, T_w)$: 输入索引 $I$ 和陷门 $T_w$, 输出包含 $W$ 的文件标识符构成的集合 $D(W)$;
- $D_i = \texttt{Decrypt}(K, C_i)$: 输入对称密钥 $K$ 和密文文件 $C_i$, 输出相应明文文件 $D_i$。

如果对称可搜索加密方案 SSE 是正确的, 那么对于 $\texttt{KeyGen}(\lambda)$ 和 $\texttt{Encrypt}(K, D)$ 输出的 $K$ 和 $(I, C)$, 都有 $\texttt{Search}(I, \texttt{Trapdoor}(K, W)) = D(W)$ 和 $\texttt{Decrypt}(K, C_i) = D_i$ 成立。这里 $C_i \in C$, $i = 1, 2, \ldots, n$。

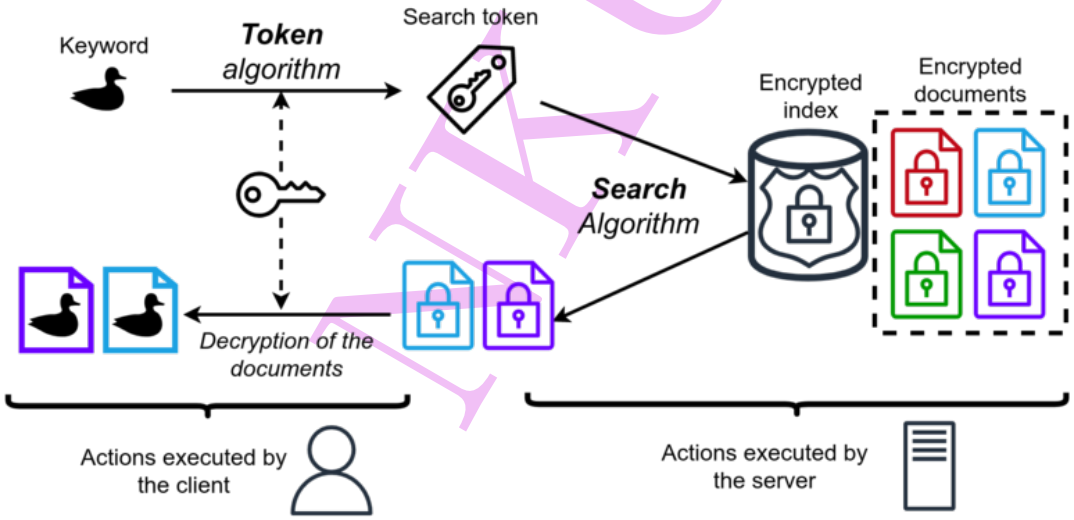



图 5: Symmetric Searchable Encryption(SSE)

**如图5所示, 基于上述定义, 对称可搜索加密流程如下:**

1. **(1) 加密过程**:
   - 用户执行 KeyGen 算法生成对称密钥 $K$。
   - 使用 $K$ 加密明文文件集 $D_i$, 将加密结果 $C_i$ 上传至服务器。
2. **(2) 陷门生成过程**:
   - 用户对关键词 $W$ 使用密钥 $K$ 生成陷门 $T_w$ (也称为令牌), 要求 $T_w$ 不能泄露关键词 $W$ 的任何信息。

3. **（3）检索过程**:

   - 用户向服务器提交关键词陷门 $T_w$。
   - 服务器使用 $T_w$ 检索到文件标识符集合 $D(W)$，并根据 $D(W)$ 中标识符将提取密文文件 $C_i$ 返回用户。

4. **（4）解密过程**:

   - 用户使用密钥 $K$ 解密服务器返回的密文文件 $C_i$，获得目标明文文件 $D_i$。

## （三） SWP 方案

在工程上来说，为了实现对称可搜索加密方案，需要采用**正向索引和倒排索引的方式。本次实验对两种索引机制都进行了实现**。

### 1. 正向索引

正向索引基本构造思路是：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。

**Dawn Song 等人于 2000 年提出了第一个实用的可搜索加密方案 SWP[1] 就是基于的正向索引**。

该方案通过使用特殊的两层加密结构来搜索加密数据，该结构允许使用顺序扫描来搜索密文。核心的想法是分别加密每个单词，然后将一个哈希值（具有特殊格式）嵌入密文中。为了进行搜索，服务器可以提取该哈希值并检查该值是否具有这种特殊格式（表示查询与加密数据的匹配）。

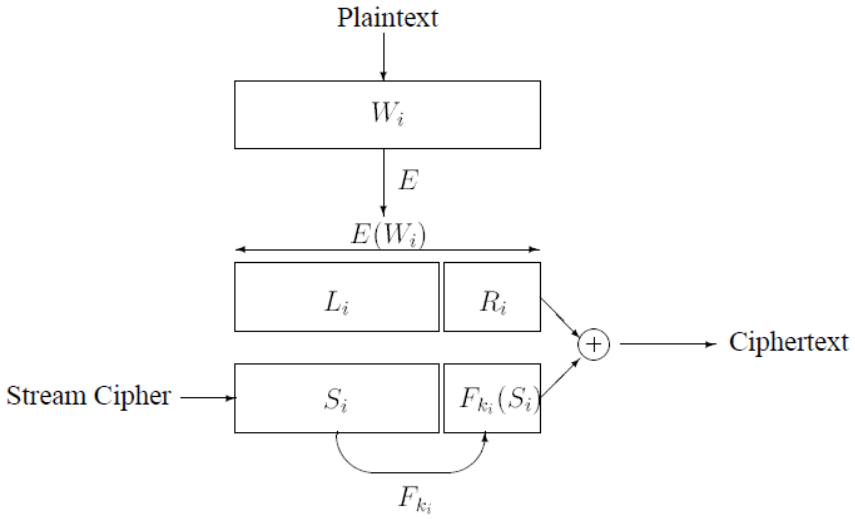



图 6: The Final Scheme

如图6所示，展示了对于一个文本如何通过加密方式完成可信的搜索。

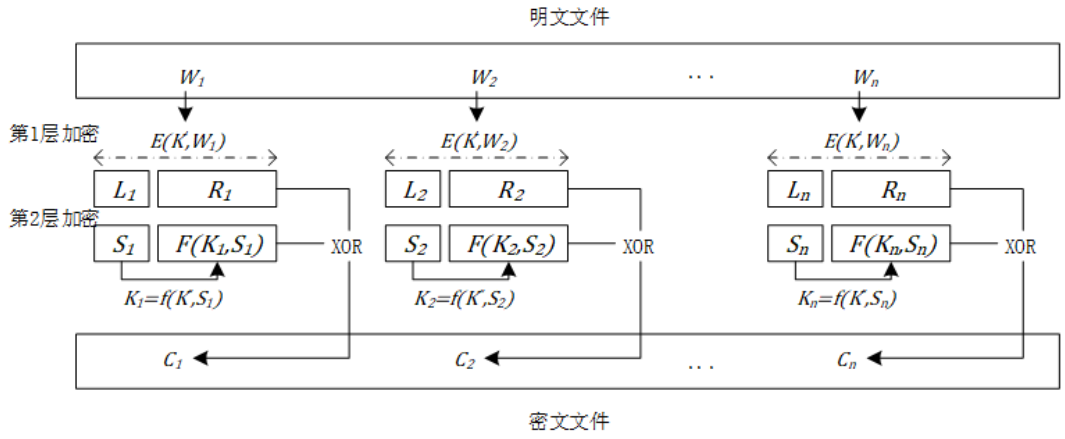



图 7: SWP 方案整体流程图

如图7所示，若将明文文件看成一个个单词组成的词素进行分词，那么整体流程会更加清晰地如图显示。

**正向索引的可搜索加密方案 SWP 整体流程**

SWP 方案在预处理过程中根据文件长度产生伪随机流 $S_1, S_2, \ldots, S_n$（$n$ 为待加密文件中的“单词”个数），然后采用两次加密。在第一层，使用分组密码 $E$ 逐个加密文件单词；在第二层，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：$L_i = f(K', L_i)$ 和 $R_i = E(K', W_i) \oplus f(K, L_i)$，其中 $f$ 为伪随机函数，$K'$ 和 $K$ 为对称密钥。
2. 基于 $L_i$ 生成二进制字符串 $F(K, S_i)$。这里，$k_i = f(K, L_i)$，$L_i$ 为符号串连接，$F$ 和 $f$ 为伪随机函数。
3. 异或 $E(K', W_i)$ 和 $F(K, S_i)$ 以形成 $W_i$ 的密文。

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。服务器根据 $T_w$ 解密并搜索文件中的所有单词，计算 $XOR(E(K', W) = S|T$，判断 $(K, S)$ 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，能够支持交控检索（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，$\alpha$ 为文件中出现的位置，仍按图7所示流程，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；
  - 在安全性方面存在较大的威胁。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。

### 2. 倒排索引

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。

**倒排索引**

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。

# 三、 实验准备

## （一） 实验环境

| | |
|---|---|
| 虚拟机软件 | VMware Workstation 17 Pro |
| 虚拟机操作系统 | Ubuntu 20.04.6 LTS amd64 |
| 实验工具 1 | Anaconda 23.7.4 |
| 实验工具 2 | Python 3.11.5 |

表 1: 本次实验环境及工具

## （二） 实验项目框架

```
/Lab06
├── forwardIndex/ 包含有本次实验的正向索引的 SWP 方案实现
│   ├── encryptUtil.py 包含有加密相关的工具函数
│   └── demo.py 通过调用 encryptUtil.py，模拟某用户对不可信邮箱服务器的基于 SWP 方案的正向索引可搜索加密演示
└── InvertedIndex/ 包含有本次实验的倒排索引的简单实现
    └── main.py 包含倒排索引的可搜索加密的代码实现与演示
```

# 四、 实验过程

本次实验的过程**分为两个部分**，第一部分是对基于 SWP 方案的正向索引可搜索加密的实现，第二部分是简单的基于倒排索引的可搜索加密的加密的实现。**两种不同的实现方法都包含了加密、陷门生成、检索和解密四个过程**。

## （一） 正向索引机制 SWP 方案

### 1. 实验代码

接下来会使用 Python **代码基于7的逻辑的正向索引可搜索加密的实现，首先给出的是 encryptUtil.py 即加密工具类的实现**。

a. **分割字符串函数**

- 函数名：split_str
- 处理逻辑：使用正则表达式将输入字符串按照单词边界进行分割，得到单词列表。
- 具体代码实现：

```
def split_str(str):
    word_list = re.split(r"\b[\.,\s\n\r\n]+?\b", str)
    return word_list
```

b. **使用 DES 加密函数**

- 函数名：encrypt
- 处理逻辑：使用 DES 算法加密字符串，转换为字节后进行加密，结果为十六进制字符串。
- 具体代码实现：

```
def encrypt(s, DES_KEY):
    fin_str = ""
    s = s.encode()
    iv = b'abcdefgh'
    des_obj = des(DES_KEY, pyDes.CBC, iv, pad=None,
    ↪  padmode=pyDes.PAD_PKCS5)
    while len(fin_str) < 64:
        secret_bytes = des_obj.encrypt(s)
        secret_hex = secret_bytes.hex()
        secret_str = str(secret_hex)
        s = secret_str
        fin_str = fin_str + secret_str
    return fin_str
```

c. **使用 DES 解密函数**

- 函数名：decrypt
- 处理逻辑：使用 DES 算法解密十六进制字符串，转换为字节后进行解密，结果为原始字符串。
- 具体代码实现：

```
def decrypt(secret_bytes, DES_KEY):
    secret_bytes = bytes.fromhex(secret_bytes)
    iv = b'abcdefgh'
    des_obj = des(DES_KEY, pyDes.CBC, iv, pad=None,
    ↪  padmode=pyDes.PAD_PKCS5)
    s = des_obj.decrypt(secret_bytes)
    return s.decode()
```

NKU

d. **字符串左右分割函数**

- 函数名：split_str_2
- 处理逻辑：将字符串按长度均分为左右两部分。
- 具体代码实现：

```
def split_str_2(str):
    left = str[:len(str)//2]
    right = str[len(str)//2:]
    return left, right
```

e. **使用 HMAC-MD5 加密函数**

- 函数名：hmac_md5
- 处理逻辑：使用 HMAC-MD5 算法对消息进行加密。
- 具体代码实现：

```
def hmac_md5(msg, key):
    return hmac.new(key.encode(), msg.encode(),
    ↪  hashlib.md5).hexdigest()
```

f. **生成随机数函数**

- 函数名：gen_random
- 处理逻辑：使用随机种子生成一个 32 位的随机数。
- 具体代码实现：

```
def gen_random(seed):
    res = ""
    dic = ["0123456789abcde"]
    random.seed(seed)
    for i in range(32):
        r = random.randint(0,14)
        res = res + dic[0][r]
    return res
```

g. **字符串异或操作函数**

- 函数名：strXOR
- 处理逻辑：对两个等长的十六进制字符串进行异或操作。
- 具体代码实现：

NKU

```python
def strXOR(str1, str2):
    if len(str1) != len(str2):
        print('ERROR: the length of two strings must be equal')
        print(len(str1), len(str2))
        return ""
    standard = len(str1) + 2
    num1 = int(str1, 16)
    num2 = int(str2, 16)
    bin1 = bin(num1)
    bin2 = bin(num2)
    res = int(bin1, 2) ^ int(bin2, 2)
    res = hex(res)
    if len(res) < standard:
        res = str(res)
        res = res[2:]
        while len(res) < standard - 2:
            res = '0' + res
        res = '0x' + res
    return res
```

h. **加密流水线函数**

- 函数名：Encrypt_pipeline
- 处理逻辑：完成从字符串加密到生成密文的整个流程。
- 具体代码实现：

```python
def Encrypt_pipeline(word, seed, DES_KEY, HASH_KEY):
    print()
    print('====================ENCRYPT PROCESS====================')
    if word == "":
        print("THIS IS AN EXAMPLE")
        word_list = split_str(test_strs[1])
        print('[ENCRYPT_LOG] word:', word_list[2])
        word = encrypt(word_list[2], DES_KEY)
        print('[ENCRYPT_LOG] word:', word)
    else:
        print('[ENCRYPT_LOG] word:', word)
        word = encrypt(word, DES_KEY)
        print('[ENCRYPT_LOG] word:', word)
    left, right = split_str_2(word)
    print('[ENCRYPT_LOG] left,right:', left, right)
    Ki = hmac_md5(left, HASH_KEY)
    print('[ENCRYPT_LOG] Ki:', Ki)
```

NKU

```
    ran = gen_random(seed)
    print('[ENCRYPT_LOG] Si:', ran)
    Fki = hmac_md5(ran, Ki)
    print('[ENCRYPT_LOG] Fki:', Fki)
    res = str(ran) + str(Fki)
    print('[ENCRYPT_LOG] join_res:', res)
    Ci = strXOR(res, word)
    print('[ENCRYPT_LOG] Ci:', Ci)
    return Ci, word, Ki, ran
```

i. **搜索流水线函数**

- 函数名：Search_pipeline
- 处理逻辑：对密文进行搜索验证，判断是否匹配。
- 具体代码实现：

```
def Search_pipeline(Ci, Xi, ki, HASH_KEY):
    print()
    print('====================SEARCH PROCESS====================')
    Ti = strXOR(Ci, Xi)[2:]
    print('[SEARCH_LOG] Ti:', Ti)
    left, right = split_str_2(Ti)
    print('[SEARCH_LOG] left,right:', left, right)
    cal_Right = hmac_md5(left, ki)
    print('[SEARCH_LOG] cal_Right:', cal_Right)
    print('[SEARCH_LOG] right:', right)
    if right == cal_Right:
        print('[SEARCH_LOG] Search Success!')
        return True
    else:
        return False
```

j. **解密流水线函数**

- 函数名：Decrypt_pipeline
- 处理逻辑：完成从密文解密到还原原始字符串的整个流程。
- 具体代码实现：

```
def Decrypt_pipeline(Si, Ci, Ki, DES_KEY, HASH_KEY):
    print()
    print('====================DECRYPT PROCESS====================')
    print('[DECRYPT_LOG] Ci:', Ci)
    Cil, Cir = split_str_2(Ci)
    print('[DECRYPT_LOG] Cil,Cir:', Cil, Cir)
```

NKU

```
    Li = strXOR(Si, Cil)
    print('[DECRYPT_LOG] Li:', Li)
    Fki = hmac_md5(Si, Ki)
    print('[DECRYPT_LOG] Fki:', Fki)
    Ri = strXOR(Cir, Fki)
    print('[DECRYPT_LOG] Ri:', Ri)
    X = Li + Ri
    print('[DECRYPT_LOG] X:', X)
    word = decrypt(str(X)[2:18], DES_KEY)
    print('[DECRYPT_LOG] word:', word)

```

k. **主流程函数**

- 函数名：Pipeline
- 处理逻辑：调用加密流水线、搜索流水线和解密流水线函数，完成整个加密、搜索和解密流程。
- 代码实现：

```
def Pipeline(word="", seed=1, DES_KEY=b'1234567812345678',
↪   HASH_KEY='12345678'):
    Ci, Xi, Ki, Si = Encrypt_pipeline(word, seed, DES_KEY, HASH_KEY)
    print()
    print('====================MAIN PROCESS====================')
    print('[MAIN_LOG] Ci', Ci)
    print('[MAIN_LOG] Xi', Xi)
    print('[MAIN_LOG] Ki', Ki)
    print('[MAIN_LOG] Si', Si)
    Search_pipeline(Ci[2:], Xi, Ki, HASH_KEY)
    Decrypt_pipeline(Si, Ci, Ki, DES_KEY, HASH_KEY)
```

接下来是调用 encryptUtil.py 即加密工具类的**展示函数** demo.py。主要用于展示了一个基于SWP（Searchable Encryption with Forward Privacy）方案的加密、搜索、解密流程，**模拟了一个安全邮件系统的工作场景**。

a. **初始化加密和哈希密钥**:

- DES_KEY = "1234567812345678"
- HASH_KEY = "12345678"

b. **加密过程**

- 函数：Encrypt_email
- 处理逻辑：该函数首先将输入字符串分割成单词，然后对每个单词进行加密，生成加密后的单词和服务器存储的密文。

NKU

- 具体代码实现：

```
def Encrypt_email(ss):
    Encrypt_email = []
    Server_email = []
    words = encryptUtil.split_str(ss)
    for word in words:
        Ci, Xi, Ki, Si = encryptUtil.Encrypt_pipeline(word, seed,
        ↪ DES_KEY, HASH_KEY)
        ew = Encrypt_word(Ci, Xi, Ki, Si)
        Encrypt_email.append(ew)
        se = Server_word(Ci)
        Server_email.append(se)

    Encrypt_Emails.append(Encrypt_email)
    Server_Emails.append(Server_email)
```

c. **加密邮件的显示**：

- 函数：print_Encrypt_Emails 和 print_Server_Emails
- 描述：分别打印客户端和服务器端存储的加密邮件内容，便于查看加密结果。
- 具体代码如下：

```
def print_Encrypt_Emails():
    count = 1
    for email in Encrypt_Emails:
        print("==================displaying %s
        ↪ Client_email===============" % str(count))
        for ew in email:
            ew.ew_print()
        count += 1

def print_Server_Emails():
    count = 1
    for email in Server_Emails:
        print("==================displaying %s
        ↪ Server_email===============" % str(count))
        for sw in email:
            sw.sw_print()
        count += 1

```

d. **陷门生成过程**

- 函数：Search_emails

NKU

- 处理逻辑：该函数生成查询词的陷门，利用陷门在服务器存储的密文中进行搜索，找到匹配的密文。
- 具体代码实现：

```
def Search_emails(ss):
    index_i = -1
    index_j = -1
    for i in range(0, len(test_strs)):
        words = encryptUtil.split_str(test_strs[i])
        for j in range(0, len(words)):
            if words[j] == ss:
                index_i = i
                index_j = j
                break
    if index_i == -1 & index_j == -1:
        print("[ERROR TRACK]: DO NOT HAVE THIS WORD")
        return None, None

    ee = Encrypt_Emails[index_i][index_j]
    correct_emails = []
    for email in Server_Emails:
        for sw in email:
            if encryptUtil.Search_pipeline(sw.getCi(), ee.getXi(),
            ↪  ee.getKi(), HASH_KEY):
                correct_emails.append(email)

    return correct_emails, ee
```

e. **检索过程**

- 函数：Search_emails
- 处理逻辑：利用生成的陷门在服务器端的密文列表中进行匹配，找到对应的加密邮件。
- 具体代码实现：

```
def Search_emails(ss):
    index_i = -1
    index_j = -1
    for i in range(0, len(test_strs)):
        words = encryptUtil.split_str(test_strs[i])
        for j in range(0, len(words)):
            if words[j] == ss:
                index_i = i
                index_j = j
                break
```

NKU

```
    if index_i == -1 & index_j == -1:
        print("[ERROR TRACK]: DO NOT HAVE THIS WORD")
        return None, None

    ee = Encrypt_Emails[index_i][index_j]
    correct_emails = []
    for email in Server_Emails:
        for sw in email:
            if encryptUtil.Search_pipeline(sw.getCi(), ee.getXi(),
            ↪  ee.getKi(), HASH_KEY):
                correct_emails.append(email)

    return correct_emails, ee
```

f. **解密过程**

- 函数：Decrypt_emails
- 处理逻辑：对检索到的密文进行解密，还原出原始邮件内容。
- 具体代码实现：

```
def Decrypt_emails(ew):
    mails = []
    for mail in Decrypt_Emails:
        temp_mail = []
        for sw in mail:
            temp_mail.append(encryptUtil.Decrypt_pipeline(ew.getSi(),
            ↪  sw.getCi(), ew.getKi(), DES_KEY, HASH_KEY))
        mails.append(temp_mail)

    print("===============DISPLAY EMAIL======================")
    for email in mails:
        print("[email]", end=' ')
        for word in email:
            print(word, end=' ')
        print()
```

g. **完整流程执行**：

- 主函数：__main__
- 描述：执行加密、显示加密结果、搜索指定词、解密搜索结果的完整流程，模拟邮件加密、搜索和解密的全流程操作。
- 具体代码实现：

```
if __name__ == '__main__':
    Encrypt_emails()
```

NKU

```
    print_Encrypt_Emails()
    # At this point, all the emails are loaded. If there is a server,
    ↪ the Server_emails should be saved by the server, and the
    ↪ Encrypt_Emails should be saved by the client.
    # But here, we are not doing that. For debugging purposes, I have
    ↪ stored the CI as well, although it is not used later.
    print_Server_Emails()
    search_word = "Yuchen"
    Searched_emails, ew = Search_emails(search_word)
    if Searched_emails == None:
        print('error search_word')
        exit()
    Decrypt_Emails = Searched_emails
    print('Searching', len(Searched_emails), 'emails containing',
    ↪ search_word)
    Decrypt_emails(ew)

```

SWP 正向索引实现综述

通过这个示例，展示了如何使用 SWP 方案实现安全邮件系统中的加密存储、关键词搜索和解密操作，从而保证邮件内容的隐私性和可搜索性。

**2. 结果分析**

本次实验对于**正向索引的可搜索加密准备了三组精心设计的字符串，**它们分别是：

1. `Nankai is surely in China`
2. `Yuchen is a Nankai student`
3. `China is a great country`

上述三个字符串都各自有独特唯一标示的关键词，如 `surely`，`Yuchen`，`country` 和共有的关键词 `is`。同时句子 1 和 2 有相同的关键词 `Nankai`，1 和 3 有相同的关键词 `China`，2 和 3 有相同的关键词 `a`。

基于上述预分析，现在对于不同的关键词进行测试，以验证我们**正向索引的的 SWP 可搜索加密方案的正确性**。

```
====================DECRYPT PROCESS====================
[DECRYPT_LOG] Ci: 37a7ab88b72e06262aa175058ce07471206fae35c527f0e57003e82995cf1d57
[DECRYPT_LOG] Cil,Cir: 37a7ab88b72e06262aa175058ce07471 206fae35c527f0e57003e82995cf1d57
[DECRYPT_LOG] Li: 0x1e7b6ac9cb59a0e53dafa36c40eb00cd
[DECRYPT_LOG] Fki: 6a5745da561604d080a66b2597f94370
[DECRYPT_LOG] Ri: 0x4a38ebef9331f435f0a5830c02365e27
[DECRYPT_LOG] X: 0x1e7b6ac9cb59a0e53dafa36c40eb00cd0x4a38ebef9331f435f0a5830c02365e27
[DECRYPT_LOG] word: China
================DISPLAY EMAIL=======================
[email] Nankai is surely in China
(base) erwinzhou@erwinzhou-virtual-machine:~/Tools$
```

图 8: surely 关键词测试结果

如图8所示，可以看到使用 surely 关键词。能够唯一地返回正确的句子。与预测的结果一致。

```
====================DECRYPT PROCESS====================
[DECRYPT_LOG] Ci: f7bbfb02fe5f2f4e493fbb58b55ddd42c76745d5f0595b771f06496fa7dc1924
[DECRYPT_LOG] Cil,Cir: f7bbfb02fe5f2f4e493fbb58b55ddd42 c76745d5f0595b771f06496fa7dc1924
[DECRYPT_LOG] Li: 0xde673a438228898d5e316d317956a9fe
[DECRYPT_LOG] Fki: eae3c412430a8a42006db7d6cb63a384
[DECRYPT_LOG] Ri: 0x2d8481c7b353d1351f6bfeb96cbfbaa0
[DECRYPT_LOG] X: 0xde673a438228898d5e316d317956a9fe0x2d8481c7b353d1351f6bfeb96cbfbaa0
[DECRYPT_LOG] word: student
================DISPLAY EMAIL=======================
[email] Yuchen is a Nankai student
(base) erwinzhou@erwinzhou-virtual-machine:~/Tools$
```

图 9: Yuchen 关键词测试结果

```
====================DECRYPT PROCESS====================
[DECRYPT_LOG] Ci: 49ef3240ff36350e2c66c84c1955baca237f456dafc515d0d2d4ec1273365d97
[DECRYPT_LOG] Cil,Cir: 49ef3240ff36350e2c66c84c1955baca 237f456dafc515d0d2d4ec1273365d97
[DECRYPT_LOG] Li: 0x6033f301834193cd3b681e25d55ece76
[DECRYPT_LOG] Fki: e19ef553b3224438f9f02bfa2d0ee1c4
[DECRYPT_LOG] Ri: 0xc2e1b03e1ce751e82b24c7e85e38bc53
[DECRYPT_LOG] X: 0x6033f301834193cd3b681e25d55ece760xc2e1b03e1ce751e82b24c7e85e38bc53
[DECRYPT_LOG] word: country
================DISPLAY EMAIL=======================
[email] China is a great country
(base) erwinzhou@erwinzhou-virtual-machine:~/Tools$
```

图 10: country 关键词测试结果

如图9和10所示，使用 Yuchen 和 country 也能够产生和预测结果一致的结论。

```
====================DECRYPT PROCESS====================
[DECRYPT_LOG] Ci: f7bbfb02fe5f2f4e493fbb58b55ddd42c76745d5f0595b771f06496fa7dc1924
[DECRYPT_LOG] Cil,Cir: f7bbfb02fe5f2f4e493fbb58b55ddd42 c76745d5f0595b771f06496fa7dc1924
[DECRYPT_LOG] Li: 0xde673a438228898d5e316d317956a9fe
[DECRYPT_LOG] Fki: cfb94e82325ce88f4a93356c96d1b79a
[DECRYPT_LOG] Ri: 0x08de0b57c205b3f855957c03310daebe
[DECRYPT_LOG] X: 0xde673a438228898d5e316d317956a9fe0x08de0b57c205b3f855957c03310daebe
[DECRYPT_LOG] word: student
================DISPLAY EMAIL=======================
[email] Nankai is surely in China
[email] Yuchen is a Nankai student
(base) erwinzhou@erwinzhou-virtual-machine:~/Tools$
```

图 11: Nankai 关键词检测结果

```
====================DECRYPT PROCESS====================
[DECRYPT_LOG] Ci: 49ef3240ff36350e2c66c84c1955baca237f456dafc515d0d2d4ec1273365d97
[DECRYPT_LOG] Cil,Cir: 49ef3240ff36350e2c66c84c1955baca 237f456dafc515d0d2d4ec1273365d97
[DECRYPT_LOG] Li: 0x6033f301834193cd3b681e25d55ece76
[DECRYPT_LOG] Fki: 7a85917b9dddfda74baffb1c55b879aa
[DECRYPT_LOG] Ri: 0x59fad4163218e877997b170e268e243d
[DECRYPT_LOG] X: 0x6033f301834193cd3b681e25d55ece760x59fad4163218e877997b170e268e243d
[DECRYPT_LOG] word: country
================DISPLAY EMAIL=======================
[email] Nankai is surely in China
[email] China is a great country
(base) erwinzhou@erwinzhou-virtual-machine:~/Tools$
```

图 12: China 关键词检测结果

```
====================DECRYPT PROCESS====================
[DECRYPT_LOG] Ci: 49ef3240ff36350e2c66c84c1955baca237f456dafc515d0d2d4ec1273365d97
[DECRYPT_LOG] Cil,Cir: 49ef3240ff36350e2c66c84c1955baca 237f456dafc515d0d2d4ec1273365d97
[DECRYPT_LOG] Li: 0x6033f301834193cd3b681e25d55ece76
[DECRYPT_LOG] Fki: d5ccbe618ec1bc5121df249c77e163f9
[DECRYPT_LOG] Ri: 0xf6b3fb0c2104a981f30bc88e04d73e6e
[DECRYPT_LOG] X: 0x6033f301834193cd3b681e25d55ece760xf6b3fb0c2104a981f30bc88e04d73e6e
[DECRYPT_LOG] word: country
================DISPLAY EMAIL=======================
[email] Yuchen is a Nankai student
[email] China is a great country
(base) erwinzhou@erwinzhou-virtual-machine:~/Tools$
```

图 13: a 关键词检测结果

1 和 2 相同的关键词 Nankai 检测结果如11所示。1 和 3 相同的关键词 China 检测结果如12所示。2 和 3 相同的关键词 a 检测结果如13所示。可以看到均与预测结果一致。

```
====================DECRYPT PROCESS====================
[DECRYPT_LOG] Ci: 49ef3240ff36350e2c66c84c1955baca237f456dafc515d0d2d4ec1273365d97
[DECRYPT_LOG] Cil,Cir: 49ef3240ff36350e2c66c84c1955baca 237f456dafc515d0d2d4ec1273365d97
[DECRYPT_LOG] Li: 0x6033f301834193cd3b681e25d55ece76
[DECRYPT_LOG] Fki: 0a8dcf1744944034c3d2e3922ab48ed3
[DECRYPT_LOG] Ri: 0x29f28a7aeb5155e411060f805982d344
[DECRYPT_LOG] X: 0x6033f301834193cd3b681e25d55ece760x29f28a7aeb5155e411060f805982d344
[DECRYPT_LOG] word: country
================DISPLAY EMAIL=======================
[email] Nankai is surely in China
[email] Yuchen is a Nankai student
[email] China is a great country
(base) erwinzhou@erwinzhou-virtual-machine:~/Tools$
```

图 14: is 关键词检测结果

如图14所示，采用三个句子都有的关键词 is 结果也与预期一致。**所有结果均与预期一致，证明了实现的正确性。能够完整地模拟加密、陷门生成、检索和解密四个过程。**

## （二） 倒排索引机制简单实现

### 1. 实验代码

#### a. 加密过程

- 函数：encrypt
- 描述：使用哈希函数对输入单词进行加密，返回加密后的十六进制字符串。
- 具体代码实现：

```python
def encrypt(self, word):
    # 使用哈希函数对单词进行加密
    return hashlib.sha256(word.encode()).hexdigest()
```

#### b. 陷门生成过程

- 函数：generate_trapdoor
- 描述：使用哈希函数对加密后的单词生成陷门，返回陷门的十六进制字符串。
- 具体代码实现：

```python
def generate_trapdoor(self, encrypted_word):
    # 使用哈希函数对密文单词进行陷门生成
    return hashlib.sha256(encrypted_word.encode()).hexdigest()
```

#### c. 索引过程

- 函数：index_document
- 描述：对输入文档进行分词，对每个单词进行加密和陷门生成，并分别存储在正向索引和倒排索引中。
- 具体代码实现：

NKU

```
def index_document(self, document):
    # 对文档进行加密和索引
    for word in document.split():
        encrypted_word = self.encrypt(word)
        trapdoor = self.generate_trapdoor(encrypted_word)

        if encrypted_word not in self.forward_index:
            self.forward_index[encrypted_word] = set()
        self.forward_index[encrypted_word].add(word)

        if trapdoor not in self.inverted_index:
            self.inverted_index[trapdoor] = set()
        self.inverted_index[trapdoor].add(encrypted_word)
```

d. **搜索过程**

- 函数：search
- 描述：对查询单词进行加密和陷门生成，并在倒排索引中查找对应的陷门，返回匹配的原始单词集合。
- 具体代码实现：

```
def search(self, query):
    # 对查询单词进行加密和陷门生成，并在倒排索引中查找对应的陷门
    encrypted_query = self.encrypt(query)
    trapdoor = self.generate_trapdoor(encrypted_query)

    if trapdoor in self.inverted_index:
        encrypted_results = self.inverted_index[trapdoor]
        results = set()
        for encrypted_word in encrypted_results:
            results.update(self.forward_index[encrypted_word])
        return results
    else:
        return set()
```

e. **示例过程：**

- 具体代码如下：

```
# 示例
scheme = SearchableEncryptionScheme()
document = "This is a sample document with some words"
scheme.index_document(document)
```

NKU

```
5
6 query = "sample"
7 results = scheme.search(query)
8 print(f"Results for query '{query}': {results}")
```

### 2. 结果分析

这里测试样例使用的是字符串："YuchenZhou is a nb student at Nankai University in China"。接下来会展示其分别检测确实存在和不存在的单词，对比结果。

```
(base) erwinzhou@erwinzhou-virtual-machine:~/Tools$ /home/erwinzhou/anaconda3/bin/python /home/erwinzhou/lab/labcodes/DataSecurity/Lab06/invertedIndex/main.py
Encrypted results for query 'YuchenZhou': {'b552675fbee014b9150f537db30e9365'}
Decrypted results for query 'YuchenZhou': {'YuchenZhou'}
(base) erwinzhou@erwinzhou-virtual-machine:~/Tools$
```

图 15: Yuchen 关键词检测结果

如图15所示，当使用 Yuchen 这个确实存在在句子中的关键词时候，返回了正确的结果还有加密的字符串。这与预期结果一致。

```
(base) erwinzhou@erwinzhou-virtual-machine:~/Tools$ /home/erwinzhou/anaconda3/bin/python /home/erwinzhou/lab/labcodes/DataSecurity/Lab06/invertedIndex/main.py
No results found for query 'ChenggongLiu'
(base) erwinzhou@erwinzhou-virtual-machine:~/Tools$
```

图 16: ChenggongLiu 关键词检测结果

如图16所示，当使用 ChenggongLiu 这个不存在句子中的管检测进行搜索时候，返回了不存在的提示信息，这也与预期结果一致。

**SWP 倒排索引实现综述**

通过这个实例，可以看到所有结果均与预期一致，证明了实现的正确性。能够完整地模拟加密、陷门生成、检索和解密四个过程。

**到此我们本次实验全部结束，总体来说实验较为成功。**

## 五、 实验总结与思考

本次实验我通过将课堂上老师讲授的可搜索加密，正向索引和倒排索引的数据安全知识在课后进一步进行了复习，并额外查找了前人的研究论文成果。跟着参考资料和老师讲解视频的思路，对论文进行了研读。

实验要求仅要求对正向索引和倒排索引二选一，**但我对两种实现方式都进行了模拟**。具体而言，工作包括：

1. 熟悉了 Linux 环境中使用 Anaconda 和 python 进行编程；
2. 通过实践，加深了对可搜索加密的相关原理和实现流程的理解；

3. **对于正向索引机制的可搜索加密机制，**阅读了文献 Representation Engineering: A Top-Down Approach to AI Transparency[1]. 并基于其原理将其具体应用到了**安全邮件系统中**的加密存储、关键词搜索和解密场景下。**能够完整地模拟加密、陷门生成、检索和解密四个过程**。
4. **对于倒排索引机制的可搜索加密机制，**进行了简单的代码实现，也**能够完整地模拟加密、陷门生成、检索和解密四个过程**。
5. 最后我还对两种索引机制的可搜索加密机制的实验结果进行了详尽的分析，证明了实现的算法正确性和安全性。

总的来说，本次实验我收获颇丰，由于没有提供参考代码，更让我加强了实践代码编程实现数据安全算法的能力。希望在最后的大作业中继续努力，将数据安全领域的知识熟记于心，并在实验中反复巩固，不断探索。

# 六、 附录

## (一) 正向索引机制 SWP 方案完整代码

**1. encryptUtil.py**

```python
import re
from pyDes import triple_des as des, PAD_PKCS5, ECB
import pyDes
import base64
import hashlib
import hmac
import random

test_strs = [
    "Nankai is surely in China",
    "Yuchen is a Nankai student",
    "China is a great country",
]

# Split the string using regular expressions
def split_str(str):
    word_list = re.split(r"\b[\.,\s\n\r\n]+?\b", str)
    return word_list

# Encrypt using DES
def encrypt(s, DES_KEY):
    fin_str = ""
    s = s.encode() # Convert Chinese characters to bytes
    iv = b'abcdefgh' # Define the initialization vector, length is 8 characters
    ↪ (64 bits)
```

NKU

```
25      des_obj = des(DES_KEY, pyDes.CBC, iv, pad=None, padmode=pyDes.PAD_PKCS5)  #
        ↪ Initialize a des object with the key, encryption mode, offset, and
        ↪ padding mode
26      while len(fin_str) < 64:
27          secret_bytes = des_obj.encrypt(s)  # Encrypt using the object's encrypt
            ↪ method
28          secret_hex = secret_bytes.hex()
29          secret_str = str(secret_hex)
30          s = secret_str
31          fin_str = fin_str + secret_str
32      return fin_str
33
34  # Decrypt using DES
35  def decrypt(secret_bytes, DES_KEY):
36      secret_bytes = bytes.fromhex(secret_bytes) # Convert Chinese characters to
        ↪ bytes
37      iv = b'abcdefgh' # Define the initialization vector, length is 8 characters
        ↪ (64 bits)
38      des_obj = des(DES_KEY, pyDes.CBC, iv, pad=None, padmode=pyDes.PAD_PKCS5)  #
        ↪ Initialize a des object with the key, encryption mode, offset, and
        ↪ padding mode
39      s = des_obj.decrypt(secret_bytes)  # Decrypt using the object's decrypt
        ↪ method
40      return s.decode()
41
42  # Split into left and right sides
43  def split_str_2(str):
44      left = str[:len(str)//2]
45      right = str[len(str)//2:]
46      return left, right
47
48  # Encrypt using hmac_MD5
49  def hmac_md5(msg, key):
50      return hmac.new(key.encode(), msg.encode(), hashlib.md5).hexdigest()
51
52  # Generate random number using a random seed
53  def gen_random(seed):
54      res = ""
55      dic = ["0123456789abcde"]
56      random.seed(seed)
57      for i in range(32):
58          r = random.randint(0,14)
59          res = res + dic[0][r]
60      return res
```

NKU

```python

# Perform XOR operation
def strXOR(str1, str2):
    if len(str1) != len(str2):
        print('ERROR: the length of two strings must be equal')
        print(len(str1), len(str2))
        return ""
    standard = len(str1) + 2
    num1 = int(str1, 16)
    num2 = int(str2, 16)
    bin1 = bin(num1)
    bin2 = bin(num2)
    res = int(bin1, 2) ^ int(bin2, 2)
    res = hex(res)
    # This is a special case where if the result starts with 0, it will result in a value that is less than the original bit length.
    if len(res) < standard:
        res = str(res)
        res = res[2:]
        while len(res) < standard - 2:
            res = '0' + res
        res = '0x' + res
    return res

# Encryption algorithm pipeline
def Encrypt_pipeline(word, seed, DES_KEY, HASH_KEY):
    print()
    print('====================ENCRYPT PROCESS====================')
    if word == "":
        print("THIS IS AN EXAMPLE")
        word_list = split_str(test_strs[1])
        print('[ENCRYPT_LOG] word:', word_list[2])
        word = encrypt(word_list[2], DES_KEY) # 64 bits
        print('[ENCRYPT_LOG] word:', word)
    else:
        print('[ENCRYPT_LOG] word:', word)
        word = encrypt(word, DES_KEY) # 64 bits
        print('[ENCRYPT_LOG] word:', word)
    left, right = split_str_2(word)
    print('[ENCRYPT_LOG] left,right:', left, right)
    Ki = hmac_md5(left, HASH_KEY) # 32 bits
    print('[ENCRYPT_LOG] Ki:', Ki)
    ran = gen_random(seed) # Generate a 32-bit random number
    print('[ENCRYPT_LOG] Si:', ran)
```

NKU

```
    Fki = hmac_md5(ran, Ki)
    print('[ENCRYPT_LOG] Fki:', Fki)
    res = str(ran) + str(Fki)
    print('[ENCRYPT_LOG] join_res:', res)
    Ci = strXOR(res, word)
    print('[ENCRYPT_LOG] Ci:', Ci)
    return Ci, word, Ki, ran

# Search algorithm pipeline
def Search_pipeline(Ci, Xi, ki, HASH_KEY):
    print()
    print('====================SEARCH PROCESS====================')
    Ti = strXOR(Ci, Xi)[2:]
    print('[SEARCH_LOG] Ti:', Ti)
    left, right = split_str_2(Ti)
    print('[SEARCH_LOG] left,right:', left, right)
    cal_Right = hmac_md5(left, ki)
    print('[SEARCH_LOG] cal_Right:', cal_Right)
    print('[SEARCH_LOG] right:', right)
    if right == cal_Right:
        print('[SEARCH_LOG] Search Success!')
        return True
    else:
        return False

def Decrypt_pipeline(Si, Ci, Ki, DES_KEY, HASH_KEY):
    print()
    print('====================DECRYPT PROCESS====================')
    # Ci = Ci[2:]
    print('[DECRYPT_LOG] Ci:', Ci)
    Cil, Cir = split_str_2(Ci)
    print('[DECRYPT_LOG] Cil,Cir:', Cil, Cir)
    Li = strXOR(Si, Cil)
    print('[DECRYPT_LOG] Li:', Li)
    Fki = hmac_md5(Si, Ki)
    print('[DECRYPT_LOG] Fki:', Fki)
    Ri = strXOR(Cir, Fki)
    print('[DECRYPT_LOG] Ri:', Ri)
    X = Li + Ri
    print('[DECRYPT_LOG] X:', X)
    word = decrypt(str(X)[2:18], DES_KEY)
    print('[DECRYPT_LOG] word:', word)
    return word

```

NKU

```
148 def Pipeline(word="", seed=1, DES_KEY=b'1234567812345678',
    ↪ HASH_KEY='12345678'):
149     Ci, Xi, Ki, Si = Encrypt_pipeline(word, seed, DES_KEY, HASH_KEY)
150     print()
151     print('====================MAIN PROCESS====================')
152     print('[MAIN_LOG] Ci', Ci)
153     print('[MAIN_LOG] Xi', Xi)
154     print('[MAIN_LOG] Ki', Ki)
155     print('[MAIN_LOG] Si', Si)
156     Search_pipeline(Ci[2:], Xi, Ki, HASH_KEY)
157     Decrypt_pipeline(Si, Ci, Ki, DES_KEY, HASH_KEY)
158 
159 if __name__ == '__main__':
160     Pipeline()
161 
```

**2. demp.py**

```
1  import encryptUtil
2  
3  test_strs = [
4      "Nankai is surely in China",
5      "Yuchen is a Nankai student",
6      "China is a great country",
7  ]
8  
9  seed = 1
10 DES_KEY = "1234567812345678"
11 HASH_KEY = "12345678"
12 Encrypt_Emails = []
13 Server_Emails = []
14 Decrypt_Emails = []
15 
16 class Encrypt_word:
17     def __init__(self, Ci, Xi, Ki, Si):
18         self.Ci = Ci
19         self.Xi = Xi
20         self.Ki = Ki
21         self.Si = Si
22     def getCi(self):
23         if self.Ci[0:2] == "0x":
24             return self.Ci[2:]
25         return self.Ci
```

NKU

```
26      def getXi(self):
27          if self.Xi[0:2] == '0x':
28              return self.Xi[2:]
29          return self.Xi
30      def getKi(self):
31          if self.Ki[0:2] == '0x':
32              return self.Ki[2:]
33          return self.Ki
34      def getSi(self):
35          if self.Si[0:2] == '0x':
36              return self.Si[2:]
37          return self.Si
38      def ew_print(self):
39          print("========================================")
40          print("[CHECK ENCRYPT_WORD] CI = ", self.Ci)
41          print("[CHECK ENCRYPT_WORD] XI = ", self.Xi)
42          print("[CHECK ENCRYPT_WORD] KI = ", self.Ki)
43          print("[CHECK ENCRYPT_WORD] SI = ", self.Si)
44          print("========================================")
45
46  class Server_word:
47      def __init__(self, Ci):
48          self.Ci = Ci
49      def getCi(self):
50          if self.Ci[0:2] == "0x":
51              return self.Ci[2:]
52          return self.Ci
53      def sw_print(self):
54          print("========================================")
55          print("[CHECK SERVER_WORD] CI = ", self.Ci)
56          print("========================================")
57
58  def print_Encrypt_Emails():
59      count = 1
60      for email in Encrypt_Emails:
61          print("==================displaying %s  Client_email==============" %
            ↪  str(count))
62          for ew in email:
63              ew.ew_print()
64          count += 1
65
66  def print_Server_Emails():
67      count = 1
68      for email in Server_Emails:
```

NKU

```
69          print("==================displaying %s  Server_email==============" %
            ↪ str(count))
70          for sw in email:
71              sw.sw_print()
72          count += 1
73
74  def Encrypt_email(ss):
75      Encrypt_email = []
76      Server_email = []
77      words = encryptUtil.split_str(ss)
78      for word in words:
79          Ci, Xi, Ki, Si = encryptUtil.Encrypt_pipeline(word, seed, DES_KEY,
            ↪ HASH_KEY)
80          ew = Encrypt_word(Ci, Xi, Ki, Si)
81          Encrypt_email.append(ew)
82          se = Server_word(Ci)
83          Server_email.append(se)
84
85      Encrypt_Emails.append(Encrypt_email)
86      Server_Emails.append(Server_email)
87
88  def Encrypt_emails():
89      for email in test_strs:
90          Encrypt_email(email)
91
92  def Search_emails(ss):
93      # Directly check if the word exists in any of the email lists and retrieve
        ↪ the corresponding Encrypt_word object
94      '''
95      In fact, at this step, the user should directly input the corresponding CI,
        ↪ XI, KI, SI.
96      But for a simple demo, it is not practical to interact with the user for
        ↪ hundreds of hex values.
97      It is better to search directly in all the existing emails.
98      '''
99      index_i = -1
100     index_j = -1
101     for i in range(0, len(test_strs)):
102         words = encryptUtil.split_str(test_strs[i])
103         for j in range(0, len(words)):
104             if words[j] == ss:
105                 index_i = i
106                 index_j = j
107                 break
```

NKU

```
108     if index_i == -1 & index_j == -1:
109         print("[ERROR TRACK]: DO NOT HAVE THIS WORD")
110         return None, None
111 
112     ee = Encrypt_Emails[index_i][index_j]
113     correct_emails = []
114     for email in Server_Emails:
115         for sw in email:
116             if encryptUtil.Search_pipeline(sw.getCi(), ee.getXi(), ee.getKi(),
                ↪  HASH_KEY):
117                 correct_emails.append(email)
118 
119     return correct_emails, ee
120 
121 def Decrypt_emails(ew):
122     mails = []
123     for mail in Decrypt_Emails:
124         temp_mail = []
125         for sw in mail:
126             temp_mail.append(encryptUtil.Decrypt_pipeline(ew.getSi(),
                ↪  sw.getCi(), ew.getKi(), DES_KEY, HASH_KEY))
127         mails.append(temp_mail)
128 
129     print("===============DISPLAY EMAIL=====================")
130     for email in mails:
131         print("[email]", end=' ')
132         for word in email:
133             print(word, end=' ')
134         print()
135 
136 
137 if __name__ == '__main__':
138     Encrypt_emails()
139     print_Encrypt_Emails()
140     # At this point, all the emails are loaded. If there is a server, the
        ↪  Server_emails should be saved by the server, and the Encrypt_Emails
        ↪  should be saved by the client.
141     # But here, we are not doing that. For debugging purposes, I have stored
        ↪  the CI as well, although it is not used later.
142     print_Server_Emails()
143     search_word = "Nankai"
144     Searched_emails, ew = Search_emails(search_word)
145     if Searched_emails == None:
146         print('error search_word')
```

NKU

```
147         exit()
148     Decrypt_Emails = Searched_emails
149     print('Searching', len(Searched_emails), 'emails containing', search_word)
150     Decrypt_emails(ew)
151
```

### 3. 完整日志信息展示

下面提供了**以 surely 作为关键词的详细日志信息，以验证程序正确性**。

```
1  ====================ENCRYPT PROCESS====================
2  [ENCRYPT_LOG] word: Nankai
3  [ENCRYPT_LOG] word:
   ↪ 1c3883f514b279b38fd5fec64f3a464de4c0ee2c9b69228a1d0398b02080e20c
4  [ENCRYPT_LOG] left,right: 1c3883f514b279b38fd5fec64f3a464d
   ↪ e4c0ee2c9b69228a1d0398b02080e20c
5  [ENCRYPT_LOG] Ki: 0ba5b5ba2289769ad2e37eeafc444b15
6  [ENCRYPT_LOG] Si: 29dcc1417c77a6c3170ed669cc0b74bc
7  [ENCRYPT_LOG] Fki: cfb94e82325ce88f4a93356c96d1b79a
8  [ENCRYPT_LOG] join_res:
   ↪ 29dcc1417c77a6c3170ed669cc0b74bccfb94e82325ce88f4a93356c96d1b79a
9  [ENCRYPT_LOG] Ci:
   ↪ 0x35e442b468c5df7098db28af833132f12b79a0aea935ca055790addcb6515596
10
11 ====================ENCRYPT PROCESS====================
12 [ENCRYPT_LOG] word: is
13 [ENCRYPT_LOG] word:
   ↪ c3082096d63be756f63feaf57f3418b97cff33c4527ae13ea940e76b3ea0c0aa
14 [ENCRYPT_LOG] left,right: c3082096d63be756f63feaf57f3418b9
   ↪ 7cff33c4527ae13ea940e76b3ea0c0aa
15 [ENCRYPT_LOG] Ki: 95d0306ff700a3a64492686ee43224a5
16 [ENCRYPT_LOG] Si: 29dcc1417c77a6c3170ed669cc0b74bc
17 [ENCRYPT_LOG] Fki: 0a8dcf1744944034c3d2e3922ab48ed3
18 [ENCRYPT_LOG] join_res:
   ↪ 29dcc1417c77a6c3170ed669cc0b74bc0a8dcf1744944034c3d2e3922ab48ed3
19 [ENCRYPT_LOG] Ci:
   ↪ 0xead4e1d7aa4c4195e1313c9cb33f6c057672fcd316eea10a6a9204f914144e79
20
21 ====================ENCRYPT PROCESS====================
22 [ENCRYPT_LOG] word: surely
23 [ENCRYPT_LOG] word:
   ↪ 4bd84c59136b4a1a624886e8247e4c0e7e59826789d36c86a6ccd98acbd53fc1
24 [ENCRYPT_LOG] left,right: 4bd84c59136b4a1a624886e8247e4c0e
   ↪ 7e59826789d36c86a6ccd98acbd53fc1
```

NKU

```
[ENCRYPT_LOG] Ki: be47d8eb92880503bfb8e80a8e066650
[ENCRYPT_LOG] Si: 29dcc1417c77a6c3170ed669cc0b74bc
[ENCRYPT_LOG] Fki: 6a5745da561604d080a66b2597f94370
[ENCRYPT_LOG] join_res:
↪ 29dcc1417c77a6c3170ed669cc0b74bc6a5745da561604d080a66b2597f94370
[ENCRYPT_LOG] Ci:
↪ 0x62048d186f1cecd975465081e87538b2140ec7bddfc56856266ab2af5c2c7cb1

====================ENCRYPT PROCESS====================
[ENCRYPT_LOG] word: in
[ENCRYPT_LOG] word:
↪ ffed9600b6bbedb467abcb2c69670641f9c82119f3b6bf123a3b8db2f32652a8
[ENCRYPT_LOG] left,right: ffed9600b6bbedb467abcb2c69670641
↪ f9c82119f3b6bf123a3b8db2f32652a8
[ENCRYPT_LOG] Ki: 01cfd891da03def4c6b90690d40b08c5
[ENCRYPT_LOG] Si: 29dcc1417c77a6c3170ed669cc0b74bc
[ENCRYPT_LOG] Fki: c8185242d7f7635f4aeec642783fac1d
[ENCRYPT_LOG] join_res:
↪ 29dcc1417c77a6c3170ed669cc0b74bcc8185242d7f7635f4aeec642783fac1d
[ENCRYPT_LOG] Ci:
↪ 0xd6315741cacc4b7770a51d45a56c72fd31d0735b2441dc4d70d54bf08b19feb5

====================ENCRYPT PROCESS====================
[ENCRYPT_LOG] word: China
[ENCRYPT_LOG] word:
↪ 1e7b6ac9cb59a0e53dafa36c40eb00cd5aea3f4e58fa0d423bac1335c07764fd
[ENCRYPT_LOG] left,right: 1e7b6ac9cb59a0e53dafa36c40eb00cd
↪ 5aea3f4e58fa0d423bac1335c07764fd
[ENCRYPT_LOG] Ki: 7e518d3f89b9293ffb74c4e9143eed62
[ENCRYPT_LOG] Si: 29dcc1417c77a6c3170ed669cc0b74bc
[ENCRYPT_LOG] Fki: 7a85917b9dddfda74baffb1c55b879aa
[ENCRYPT_LOG] join_res:
↪ 29dcc1417c77a6c3170ed669cc0b74bc7a85917b9dddfda74baffb1c55b879aa
[ENCRYPT_LOG] Ci:
↪ 0x37a7ab88b72e06262aa175058ce07471206fae35c527f0e57003e82995cf1d57

====================ENCRYPT PROCESS====================
[ENCRYPT_LOG] word: Yuchen
[ENCRYPT_LOG] word:
↪ 035c982be595e246afbe20c485f6b756868f35ef51401cffd03e22e94ce054ee
[ENCRYPT_LOG] left,right: 035c982be595e246afbe20c485f6b756
↪ 868f35ef51401cffd03e22e94ce054ee
[ENCRYPT_LOG] Ki: 560b03cf4a91b1d6368bec2634cd3d20
[ENCRYPT_LOG] Si: 29dcc1417c77a6c3170ed669cc0b74bc
```

NKU

```
[ENCRYPT_LOG] Fki: eae3c412430a8a42006db7d6cb63a384
[ENCRYPT_LOG] join_res:
↪ 29dcc1417c77a6c3170ed669cc0b74bceae3c412430a8a42006db7d6cb63a384
[ENCRYPT_LOG] Ci:
↪ 0x2a80596a99e24485b8b0f6ad49fdc3ea6c6cf1fd124a96bdd053953f8783f76a

======================ENCRYPT PROCESS======================
[ENCRYPT_LOG] word: is
[ENCRYPT_LOG] word:
↪ c3082096d63be756f63feaf57f3418b97cff33c4527ae13ea940e76b3ea0c0aa
[ENCRYPT_LOG] left,right: c3082096d63be756f63feaf57f3418b9
↪ 7cff33c4527ae13ea940e76b3ea0c0aa
[ENCRYPT_LOG] Ki: 95d0306ff700a3a64492686ee43224a5
[ENCRYPT_LOG] Si: 29dcc1417c77a6c3170ed669cc0b74bc
[ENCRYPT_LOG] Fki: 0a8dcf1744944034c3d2e3922ab48ed3
[ENCRYPT_LOG] join_res:
↪ 29dcc1417c77a6c3170ed669cc0b74bc0a8dcf1744944034c3d2e3922ab48ed3
[ENCRYPT_LOG] Ci:
↪ 0xead4e1d7aa4c4195e1313c9cb33f6c057672fcd316eea10a6a9204f914144e79

======================ENCRYPT PROCESS======================
[ENCRYPT_LOG] word: a
[ENCRYPT_LOG] word:
↪ f6d7c397808b6980145f8b3f95970f0237979dc4976745b654f85538d2e26ffb
[ENCRYPT_LOG] left,right: f6d7c397808b6980145f8b3f95970f02
↪ 37979dc4976745b654f85538d2e26ffb
[ENCRYPT_LOG] Ki: 67cb1debc0b68eb489b573427b441db4
[ENCRYPT_LOG] Si: 29dcc1417c77a6c3170ed669cc0b74bc
[ENCRYPT_LOG] Fki: d5ccbe618ec1bc5121df249c77e163f9
[ENCRYPT_LOG] join_res:
↪ 29dcc1417c77a6c3170ed669cc0b74bcd5ccbe618ec1bc5121df249c77e163f9
[ENCRYPT_LOG] Ci:
↪ 0xdf0b02d6fcfccf4303515d56599c7bbee25b23a519a6f9e7752771a4a5030c02

======================ENCRYPT PROCESS======================
[ENCRYPT_LOG] word: Nankai
[ENCRYPT_LOG] word:
↪ 1c3883f514b279b38fd5fec64f3a464de4c0ee2c9b69228a1d0398b02080e20c
[ENCRYPT_LOG] left,right: 1c3883f514b279b38fd5fec64f3a464d
↪ e4c0ee2c9b69228a1d0398b02080e20c
[ENCRYPT_LOG] Ki: 0ba5b5ba2289769ad2e37eeafc444b15
[ENCRYPT_LOG] Si: 29dcc1417c77a6c3170ed669cc0b74bc
[ENCRYPT_LOG] Fki: cfb94e82325ce88f4a93356c96d1b79a
```

NKU

```
[ENCRYPT_LOG] join_res:
↪ 29dcc1417c77a6c3170ed669cc0b74bccfb94e82325ce88f4a93356c96d1b79a
[ENCRYPT_LOG] Ci:
↪ 0x35e442b468c5df7098db28af833132f12b79a0aea935ca055790addcb6515596

====================ENCRYPT PROCESS====================
[ENCRYPT_LOG] word: student
[ENCRYPT_LOG] word:
↪ de673a438228898d5e316d317956a9fe7ee40e03464362383d4e5385f3b957ad
[ENCRYPT_LOG] left,right: de673a438228898d5e316d317956a9fe
↪ 7ee40e03464362383d4e5385f3b957ad
[ENCRYPT_LOG] Ki: 7c39554c4141c9008077accb12b39ecc
[ENCRYPT_LOG] Si: 29dcc1417c77a6c3170ed669cc0b74bc
[ENCRYPT_LOG] Fki: b9834bd6b61a394f22481aea54654e89
[ENCRYPT_LOG] join_res:
↪ 29dcc1417c77a6c3170ed669cc0b74bcb9834bd6b61a394f22481aea54654e89
[ENCRYPT_LOG] Ci:
↪ 0xf7bbfb02fe5f2f4e493fbb58b55ddd42c76745d5f0595b771f06496fa7dc1924

====================ENCRYPT PROCESS====================
[ENCRYPT_LOG] word: China
[ENCRYPT_LOG] word:
↪ 1e7b6ac9cb59a0e53dafa36c40eb00cd5aea3f4e58fa0d423bac1335c07764fd
[ENCRYPT_LOG] left,right: 1e7b6ac9cb59a0e53dafa36c40eb00cd
↪ 5aea3f4e58fa0d423bac1335c07764fd
[ENCRYPT_LOG] Ki: 7e518d3f89b9293ffb74c4e9143eed62
[ENCRYPT_LOG] Si: 29dcc1417c77a6c3170ed669cc0b74bc
[ENCRYPT_LOG] Fki: 7a85917b9dddfda74baffb1c55b879aa
[ENCRYPT_LOG] join_res:
↪ 29dcc1417c77a6c3170ed669cc0b74bc7a85917b9dddfda74baffb1c55b879aa
[ENCRYPT_LOG] Ci:
↪ 0x37a7ab88b72e06262aa175058ce07471206fae35c527f0e57003e82995cf1d57

====================ENCRYPT PROCESS====================
[ENCRYPT_LOG] word: is
[ENCRYPT_LOG] word:
↪ c3082096d63be756f63feaf57f3418b97cff33c4527ae13ea940e76b3ea0c0aa
[ENCRYPT_LOG] left,right: c3082096d63be756f63feaf57f3418b9
↪ 7cff33c4527ae13ea940e76b3ea0c0aa
[ENCRYPT_LOG] Ki: 95d0306ff700a3a64492686ee43224a5
[ENCRYPT_LOG] Si: 29dcc1417c77a6c3170ed669cc0b74bc
[ENCRYPT_LOG] Fki: 0a8dcf1744944034c3d2e3922ab48ed3
[ENCRYPT_LOG] join_res:
↪ 29dcc1417c77a6c3170ed669cc0b74bc0a8dcf1744944034c3d2e3922ab48ed3
```

NKU

```
[ENCRYPT_LOG] Ci:
    ↪ 0xead4e1d7aa4c4195e1313c9cb33f6c057672fcd316eea10a6a9204f914144e79

====================ENCRYPT PROCESS====================
[ENCRYPT_LOG] word: a
[ENCRYPT_LOG] word:
    ↪ f6d7c397808b6980145f8b3f95970f0237979dc4976745b654f85538d2e26ffb
[ENCRYPT_LOG] left,right: f6d7c397808b6980145f8b3f95970f02
    ↪ 37979dc4976745b654f85538d2e26ffb
[ENCRYPT_LOG] Ki: 67cb1debc0b68eb489b573427b441db4
[ENCRYPT_LOG] Si: 29dcc1417c77a6c3170ed669cc0b74bc
[ENCRYPT_LOG] Fki: d5ccbe618ec1bc5121df249c77e163f9
[ENCRYPT_LOG] join_res:
    ↪ 29dcc1417c77a6c3170ed669cc0b74bcd5ccbe618ec1bc5121df249c77e163f9
[ENCRYPT_LOG] Ci:
    ↪ 0xdf0b02d6fcfccf4303515d56599c7bbee25b23a519a6f9e7752771a4a5030c02

====================ENCRYPT PROCESS====================
[ENCRYPT_LOG] word: great
[ENCRYPT_LOG] word:
    ↪ 69de8f1be916112a1a577030756ee389e3cad913dac2b6dfc0cc1e5b550fb340
[ENCRYPT_LOG] left,right: 69de8f1be916112a1a577030756ee389
    ↪ e3cad913dac2b6dfc0cc1e5b550fb340
[ENCRYPT_LOG] Ki: 72a27be9390c158dc6f253ece7d1d4c9
[ENCRYPT_LOG] Si: 29dcc1417c77a6c3170ed669cc0b74bc
[ENCRYPT_LOG] Fki: 29457a66f1e8539aa0e24caf78308da6
[ENCRYPT_LOG] join_res:
    ↪ 29dcc1417c77a6c3170ed669cc0b74bc29457a66f1e8539aa0e24caf78308da6
[ENCRYPT_LOG] Ci:
    ↪ 0x40024e5a9561b7e90d59a659b9659735ca8fa3752b2ae545602e52f42d3f3ee6

====================ENCRYPT PROCESS====================
[ENCRYPT_LOG] word: country
[ENCRYPT_LOG] word:
    ↪ 6033f301834193cd3b681e25d55ece76c2e1b03e1ce751e82b24c7e85e38bc53
[ENCRYPT_LOG] left,right: 6033f301834193cd3b681e25d55ece76
    ↪ c2e1b03e1ce751e82b24c7e85e38bc53
[ENCRYPT_LOG] Ki: 33752513b3a1657961ebcf00899ee8d3
[ENCRYPT_LOG] Si: 29dcc1417c77a6c3170ed669cc0b74bc
[ENCRYPT_LOG] Fki: e19ef553b3224438f9f02bfa2d0ee1c4
[ENCRYPT_LOG] join_res:
    ↪ 29dcc1417c77a6c3170ed669cc0b74bce19ef553b3224438f9f02bfa2d0ee1c4
[ENCRYPT_LOG] Ci:
    ↪ 0x49ef3240ff36350e2c66c84c1955baca237f456dafc515d0d2d4ec1273365d97
```

NKU

```
==================displaying 1  Client_email==============
========================================
[CHECK ENCRYPT_WORD] CI =
↪  0x35e442b468c5df7098db28af833132f12b79a0aea935ca055790addcb6515596
[CHECK ENCRYPT_WORD] XI =
↪  1c3883f514b279b38fd5fec64f3a464de4c0ee2c9b69228a1d0398b02080e20c
[CHECK ENCRYPT_WORD] KI =  0ba5b5ba2289769ad2e37eeafc444b15
[CHECK ENCRYPT_WORD] SI =  29dcc1417c77a6c3170ed669cc0b74bc
========================================
========================================
[CHECK ENCRYPT_WORD] CI =
↪  0xead4e1d7aa4c4195e1313c9cb33f6c057672fcd316eea10a6a9204f914144e79
[CHECK ENCRYPT_WORD] XI =
↪  c3082096d63be756f63feaf57f3418b97cff33c4527ae13ea940e76b3ea0c0aa
[CHECK ENCRYPT_WORD] KI =  95d0306ff700a3a64492686ee43224a5
[CHECK ENCRYPT_WORD] SI =  29dcc1417c77a6c3170ed669cc0b74bc
========================================
========================================
[CHECK ENCRYPT_WORD] CI =
↪  0x62048d186f1cecd975465081e87538b2140ec7bddfc56856266ab2af5c2c7cb1
[CHECK ENCRYPT_WORD] XI =
↪  4bd84c59136b4a1a624886e8247e4c0e7e59826789d36c86a6ccd98acbd53fc1
[CHECK ENCRYPT_WORD] KI =  be47d8eb92880503bfb8e80a8e066650
[CHECK ENCRYPT_WORD] SI =  29dcc1417c77a6c3170ed669cc0b74bc
========================================
========================================
[CHECK ENCRYPT_WORD] CI =
↪  0xd6315741cacc4b7770a51d45a56c72fd31d0735b2441dc4d70d54bf08b19feb5
[CHECK ENCRYPT_WORD] XI =
↪  ffed9600b6bbedb467abcb2c69670641f9c82119f3b6bf123a3b8db2f32652a8
[CHECK ENCRYPT_WORD] KI =  01cfd891da03def4c6b90690d40b08c5
[CHECK ENCRYPT_WORD] SI =  29dcc1417c77a6c3170ed669cc0b74bc
========================================
========================================
[CHECK ENCRYPT_WORD] CI =
↪  0x37a7ab88b72e06262aa175058ce07471206fae35c527f0e57003e82995cf1d57
[CHECK ENCRYPT_WORD] XI =
↪  1e7b6ac9cb59a0e53dafa36c40eb00cd5aea3f4e58fa0d423bac1335c07764fd
[CHECK ENCRYPT_WORD] KI =  7e518d3f89b9293ffb74c4e9143eed62
[CHECK ENCRYPT_WORD] SI =  29dcc1417c77a6c3170ed669cc0b74bc
========================================
==================displaying 2  Client_email==============
========================================
```

NKU

```
[CHECK ENCRYPT_WORD] CI =
↪ 0x2a80596a99e24485b8b0f6ad49fdc3ea6c6cf1fd124a96bdd053953f8783f76a
[CHECK ENCRYPT_WORD] XI =
↪ 035c982be595e246afbe20c485f6b756868f35ef51401cffd03e22e94ce054ee
[CHECK ENCRYPT_WORD] KI = 560b03cf4a91b1d6368bec2634cd3d20
[CHECK ENCRYPT_WORD] SI = 29dcc1417c77a6c3170ed669cc0b74bc
========================================
========================================
[CHECK ENCRYPT_WORD] CI =
↪ 0xead4e1d7aa4c4195e1313c9cb33f6c057672fcd316eea10a6a9204f914144e79
[CHECK ENCRYPT_WORD] XI =
↪ c3082096d63be756f63feaf57f3418b97cff33c4527ae13ea940e76b3ea0c0aa
[CHECK ENCRYPT_WORD] KI = 95d0306ff700a3a64492686ee43224a5
[CHECK ENCRYPT_WORD] SI = 29dcc1417c77a6c3170ed669cc0b74bc
========================================
========================================
[CHECK ENCRYPT_WORD] CI =
↪ 0xdf0b02d6fcfccf4303515d56599c7bbee25b23a519a6f9e7752771a4a5030c02
[CHECK ENCRYPT_WORD] XI =
↪ f6d7c397808b6980145f8b3f95970f0237979dc4976745b654f85538d2e26ffb
[CHECK ENCRYPT_WORD] KI = 67cb1debc0b68eb489b573427b441db4
[CHECK ENCRYPT_WORD] SI = 29dcc1417c77a6c3170ed669cc0b74bc
========================================
========================================
[CHECK ENCRYPT_WORD] CI =
↪ 0x35e442b468c5df7098db28af833132f12b79a0aea935ca055790addcb6515596
[CHECK ENCRYPT_WORD] XI =
↪ 1c3883f514b279b38fd5fec64f3a464de4c0ee2c9b69228a1d0398b02080e20c
[CHECK ENCRYPT_WORD] KI = 0ba5b5ba2289769ad2e37eeafc444b15
[CHECK ENCRYPT_WORD] SI = 29dcc1417c77a6c3170ed669cc0b74bc
========================================
========================================
[CHECK ENCRYPT_WORD] CI =
↪ 0xf7bbfb02fe5f2f4e493fbb58b55ddd42c76745d5f0595b771f06496fa7dc1924
[CHECK ENCRYPT_WORD] XI =
↪ de673a438228898d5e316d317956a9fe7ee40e03464362383d4e5385f3b957ad
[CHECK ENCRYPT_WORD] KI = 7c39554c4141c9008077accb12b39ecc
[CHECK ENCRYPT_WORD] SI = 29dcc1417c77a6c3170ed669cc0b74bc
========================================
==========displaying 3 Client_email==========
========================================
[CHECK ENCRYPT_WORD] CI =
↪ 0x37a7ab88b72e06262aa175058ce07471206fae35c527f0e57003e82995cf1d57
```

NKU

```
[CHECK ENCRYPT_WORD] XI =
↪ 1e7b6ac9cb59a0e53dafa36c40eb00cd5aea3f4e58fa0d423bac1335c07764fd
[CHECK ENCRYPT_WORD] KI = 7e518d3f89b9293ffb74c4e9143eed62
[CHECK ENCRYPT_WORD] SI = 29dcc1417c77a6c3170ed669cc0b74bc
========================================
========================================
[CHECK ENCRYPT_WORD] CI =
↪ 0xead4e1d7aa4c4195e1313c9cb33f6c057672fcd316eea10a6a9204f914144e79
[CHECK ENCRYPT_WORD] XI =
↪ c3082096d63be756f63feaf57f3418b97cff33c4527ae13ea940e76b3ea0c0aa
[CHECK ENCRYPT_WORD] KI = 95d0306ff700a3a64492686ee43224a5
[CHECK ENCRYPT_WORD] SI = 29dcc1417c77a6c3170ed669cc0b74bc
========================================
========================================
[CHECK ENCRYPT_WORD] CI =
↪ 0xdf0b02d6fcfccf4303515d56599c7bbee25b23a519a6f9e7752771a4a5030c02
[CHECK ENCRYPT_WORD] XI =
↪ f6d7c397808b6980145f8b3f95970f0237979dc4976745b654f85538d2e26ffb
[CHECK ENCRYPT_WORD] KI = 67cb1debc0b68eb489b573427b441db4
[CHECK ENCRYPT_WORD] SI = 29dcc1417c77a6c3170ed669cc0b74bc
========================================
========================================
[CHECK ENCRYPT_WORD] CI =
↪ 0x40024e5a9561b7e90d59a659b9659735ca8fa3752b2ae545602e52f42d3f3ee6
[CHECK ENCRYPT_WORD] XI =
↪ 69de8f1be916112a1a577030756ee389e3cad913dac2b6dfc0cc1e5b550fb340
[CHECK ENCRYPT_WORD] KI = 72a27be9390c158dc6f253ece7d1d4c9
[CHECK ENCRYPT_WORD] SI = 29dcc1417c77a6c3170ed669cc0b74bc
========================================
========================================
[CHECK ENCRYPT_WORD] CI =
↪ 0x49ef3240ff36350e2c66c84c1955baca237f456dafc515d0d2d4ec1273365d97
[CHECK ENCRYPT_WORD] XI =
↪ 6033f301834193cd3b681e25d55ece76c2e1b03e1ce751e82b24c7e85e38bc53
[CHECK ENCRYPT_WORD] KI = 33752513b3a1657961ebcf00899ee8d3
[CHECK ENCRYPT_WORD] SI = 29dcc1417c77a6c3170ed669cc0b74bc
========================================
==========displaying 1 Server_email==========
========================================
[CHECK SERVER_WORD] CI =
↪ 0x35e442b468c5df7098db28af833132f12b79a0aea935ca055790addcb6515596
========================================
========================================
```

NKU

```
[CHECK SERVER_WORD] CI =
↪ 0xead4e1d7aa4c4195e1313c9cb33f6c057672fcd316eea10a6a9204f914144e79
========================================
========================================
[CHECK SERVER_WORD] CI =
↪ 0x62048d186f1cecd975465081e87538b2140ec7bddfc56856266ab2af5c2c7cb1
========================================
========================================
[CHECK SERVER_WORD] CI =
↪ 0xd6315741cacc4b7770a51d45a56c72fd31d0735b2441dc4d70d54bf08b19feb5
========================================
========================================
[CHECK SERVER_WORD] CI =
↪ 0x37a7ab88b72e06262aa175058ce07471206fae35c527f0e57003e82995cf1d57
========================================
==================displaying 2  Server_email===============
========================================
[CHECK SERVER_WORD] CI =
↪ 0x2a80596a99e24485b8b0f6ad49fdc3ea6c6cf1fd124a96bdd053953f8783f76a
========================================
========================================
[CHECK SERVER_WORD] CI =
↪ 0xead4e1d7aa4c4195e1313c9cb33f6c057672fcd316eea10a6a9204f914144e79
========================================
========================================
[CHECK SERVER_WORD] CI =
↪ 0xdf0b02d6fcfccf4303515d56599c7bbee25b23a519a6f9e7752771a4a5030c02
========================================
========================================
[CHECK SERVER_WORD] CI =
↪ 0x35e442b468c5df7098db28af833132f12b79a0aea935ca055790addcb6515596
========================================
========================================
[CHECK SERVER_WORD] CI =
↪ 0xf7bbfb02fe5f2f4e493fbb58b55ddd42c76745d5f0595b771f06496fa7dc1924
========================================
==================displaying 3  Server_email===============
========================================
[CHECK SERVER_WORD] CI =
↪ 0x37a7ab88b72e06262aa175058ce07471206fae35c527f0e57003e82995cf1d57
========================================
========================================
[CHECK SERVER_WORD] CI =
↪ 0xead4e1d7aa4c4195e1313c9cb33f6c057672fcd316eea10a6a9204f914144e79
```

NKU

```
281 ========================================
282 ========================================
283 [CHECK SERVER_WORD] CI =
    ↪ 0xdf0b02d6fcfccf4303515d56599c7bbee25b23a519a6f9e7752771a4a5030c02
284 ========================================
285 ========================================
286 [CHECK SERVER_WORD] CI =
    ↪ 0x40024e5a9561b7e90d59a659b9659735ca8fa3752b2ae545602e52f42d3f3ee6
287 ========================================
288 ========================================
289 [CHECK SERVER_WORD] CI =
    ↪ 0x49ef3240ff36350e2c66c84c1955baca237f456dafc515d0d2d4ec1273365d97
290 ========================================
291
292 ====================SEARCH PROCESS====================
293 [SEARCH_LOG] Ti:
    ↪ 7e3c0eed7bae956afa93ae47a74f7eff552022c920e6a683f15c74567d846a57
294 [SEARCH_LOG] left,right: 7e3c0eed7bae956afa93ae47a74f7eff
    ↪ 552022c920e6a683f15c74567d846a57
295 [SEARCH_LOG] cal_Right: 346d923c4f7f418362d64ce0eb33346b
296 [SEARCH_LOG] right: 552022c920e6a683f15c74567d846a57
297
298 ====================SEARCH PROCESS====================
299 [SEARCH_LOG] Ti:
    ↪ a10cad8eb9270b8f8379ba749741200b082b7eb49f3dcd8ccc5edd73dfc171b8
300 [SEARCH_LOG] left,right: a10cad8eb9270b8f8379ba749741200b
    ↪ 082b7eb49f3dcd8ccc5edd73dfc171b8
301 [SEARCH_LOG] cal_Right: 6e86196631e5819a4c9046a6cb8996ae
302 [SEARCH_LOG] right: 082b7eb49f3dcd8ccc5edd73dfc171b8
303
304 ====================SEARCH PROCESS====================
305 [SEARCH_LOG] Ti:
    ↪ 29dcc1417c77a6c3170ed669cc0b74bc6a5745da561604d080a66b2597f94370
306 [SEARCH_LOG] left,right: 29dcc1417c77a6c3170ed669cc0b74bc
    ↪ 6a5745da561604d080a66b2597f94370
307 [SEARCH_LOG] cal_Right: 6a5745da561604d080a66b2597f94370
308 [SEARCH_LOG] right: 6a5745da561604d080a66b2597f94370
309 [SEARCH_LOG] Search Success!
310
311 ====================SEARCH PROCESS====================
312 [SEARCH_LOG] Ti:
    ↪ 9de91b18d9a7016d12ed9bad81123ef34f89f13cad92b0cbd619927a40ccc174
313 [SEARCH_LOG] left,right: 9de91b18d9a7016d12ed9bad81123ef3
    ↪ 4f89f13cad92b0cbd619927a40ccc174
```

NKU

```
[SEARCH_LOG] cal_Right: 3535c86b1e65d53dc976e1fd7f3cf2c4
[SEARCH_LOG] right: 4f89f13cad92b0cbd619927a40ccc174

====================SEARCH PROCESS====================
[SEARCH_LOG] Ti:
↪ 7c7fe7d1a4454c3c48e9f3eda89e387f5e362c524cf49c63d6cf31a35e1a2296
[SEARCH_LOG] left,right: 7c7fe7d1a4454c3c48e9f3eda89e387f
↪ 5e362c524cf49c63d6cf31a35e1a2296
[SEARCH_LOG] cal_Right: ce83de0c639f54fdbc395479ec03c0f8
[SEARCH_LOG] right: 5e362c524cf49c63d6cf31a35e1a2296

====================SEARCH PROCESS====================
[SEARCH_LOG] Ti:
↪ 615815338a890e9fdaf870456d838fe41235739a9b99fa3b769f4cb54c56c8ab
[SEARCH_LOG] left,right: 615815338a890e9fdaf870456d838fe4
↪ 1235739a9b99fa3b769f4cb54c56c8ab
[SEARCH_LOG] cal_Right: 4a77dcaaf97d9124a22cfc66ade08e48
[SEARCH_LOG] right: 1235739a9b99fa3b769f4cb54c56c8ab

====================SEARCH PROCESS====================
[SEARCH_LOG] Ti:
↪ a10cad8eb9270b8f8379ba749741200b082b7eb49f3dcd8ccc5edd73dfc171b8
[SEARCH_LOG] left,right: a10cad8eb9270b8f8379ba749741200b
↪ 082b7eb49f3dcd8ccc5edd73dfc171b8
[SEARCH_LOG] cal_Right: 6e86196631e5819a4c9046a6cb8996ae
[SEARCH_LOG] right: 082b7eb49f3dcd8ccc5edd73dfc171b8

====================SEARCH PROCESS====================
[SEARCH_LOG] Ti:
↪ 94d34e8fef9785596119dbbe7de237b09c02a1c290759561d3eba82e6ed633c3
[SEARCH_LOG] left,right: 94d34e8fef9785596119dbbe7de237b0
↪ 9c02a1c290759561d3eba82e6ed633c3
[SEARCH_LOG] cal_Right: 90c199909c37c311f23c51bc5142f3c6
[SEARCH_LOG] right: 9c02a1c290759561d3eba82e6ed633c3

====================SEARCH PROCESS====================
[SEARCH_LOG] Ti:
↪ 7e3c0eed7bae956afa93ae47a74f7eff552022c920e6a683f15c74567d846a57
[SEARCH_LOG] left,right: 7e3c0eed7bae956afa93ae47a74f7eff
↪ 552022c920e6a683f15c74567d846a57
[SEARCH_LOG] cal_Right: 346d923c4f7f418362d64ce0eb33346b
[SEARCH_LOG] right: 552022c920e6a683f15c74567d846a57

====================SEARCH PROCESS====================
```

NKU

```
[SEARCH_LOG] Ti:
↪ bc63b75bed3465542b773db09123914cb93ec7b2798a37f1b9ca90e56c0926e5
[SEARCH_LOG] left,right: bc63b75bed3465542b773db09123914c
↪ b93ec7b2798a37f1b9ca90e56c0926e5
[SEARCH_LOG] cal_Right: 3d97f78c1acc5ef81fb1e411b1fbfbc3
[SEARCH_LOG] right: b93ec7b2798a37f1b9ca90e56c0926e5

====================SEARCH PROCESS====================
[SEARCH_LOG] Ti:
↪ 7c7fe7d1a4454c3c48e9f3eda89e387f5e362c524cf49c63d6cf31a35e1a2296
[SEARCH_LOG] left,right: 7c7fe7d1a4454c3c48e9f3eda89e387f
↪ 5e362c524cf49c63d6cf31a35e1a2296
[SEARCH_LOG] cal_Right: ce83de0c639f54fdbc395479ec03c0f8
[SEARCH_LOG] right: 5e362c524cf49c63d6cf31a35e1a2296

====================SEARCH PROCESS====================
[SEARCH_LOG] Ti:
↪ a10cad8eb9270b8f8379ba749741200b082b7eb49f3dcd8ccc5edd73dfc171b8
[SEARCH_LOG] left,right: a10cad8eb9270b8f8379ba749741200b
↪ 082b7eb49f3dcd8ccc5edd73dfc171b8
[SEARCH_LOG] cal_Right: 6e86196631e5819a4c9046a6cb8996ae
[SEARCH_LOG] right: 082b7eb49f3dcd8ccc5edd73dfc171b8

====================SEARCH PROCESS====================
[SEARCH_LOG] Ti:
↪ 94d34e8fef9785596119dbbe7de237b09c02a1c290759561d3eba82e6ed633c3
[SEARCH_LOG] left,right: 94d34e8fef9785596119dbbe7de237b0
↪ 9c02a1c290759561d3eba82e6ed633c3
[SEARCH_LOG] cal_Right: 90c199909c37c311f23c51bc5142f3c6
[SEARCH_LOG] right: 9c02a1c290759561d3eba82e6ed633c3

====================SEARCH PROCESS====================
[SEARCH_LOG] Ti:
↪ 0bda0203860afdf36f1120b19d1bdb3bb4d62112a2f989c3c6e28b7ee6ea0127
[SEARCH_LOG] left,right: 0bda0203860afdf36f1120b19d1bdb3b
↪ b4d62112a2f989c3c6e28b7ee6ea0127
[SEARCH_LOG] cal_Right: 503b08639acb006558e3badc04f43168
[SEARCH_LOG] right: b4d62112a2f989c3c6e28b7ee6ea0127

====================SEARCH PROCESS====================
[SEARCH_LOG] Ti:
↪ 02377e19ec5d7f144e2e4ea43d2bf6c45d26c70a2616795674183598b8e36256
[SEARCH_LOG] left,right: 02377e19ec5d7f144e2e4ea43d2bf6c4
↪ 5d26c70a2616795674183598b8e36256
```

NKU

```
[SEARCH_LOG] cal_Right: 9d3bcae1ec5bf6c6dd51263ca5259bab
[SEARCH_LOG] right: 5d26c70a2616795674183598b8e36256
Searching 1 emails containing surely

====================DECRYPT PROCESS====================
[DECRYPT_LOG] Ci:
↪ 35e442b468c5df7098db28af833132f12b79a0aea935ca055790addcb6515596
[DECRYPT_LOG] Cil,Cir: 35e442b468c5df7098db28af833132f1
↪ 2b79a0aea935ca055790addcb6515596
[DECRYPT_LOG] Li: 0x1c3883f514b279b38fd5fec64f3a464d
[DECRYPT_LOG] Fki: 6a5745da561604d080a66b2597f94370
[DECRYPT_LOG] Ri: 0x412ee574ff23ced5d736c6f921a816e6
[DECRYPT_LOG] X:
↪ 0x1c3883f514b279b38fd5fec64f3a464d0x412ee574ff23ced5d736c6f921a816e6
[DECRYPT_LOG] word: Nankai

====================DECRYPT PROCESS====================
[DECRYPT_LOG] Ci:
↪ ead4e1d7aa4c4195e1313c9cb33f6c057672fcd316eea10a6a9204f914144e79
[DECRYPT_LOG] Cil,Cir: ead4e1d7aa4c4195e1313c9cb33f6c05
↪ 7672fcd316eea10a6a9204f914144e79
[DECRYPT_LOG] Li: 0xc3082096d63be756f63feaf57f3418b9
[DECRYPT_LOG] Fki: 6a5745da561604d080a66b2597f94370
[DECRYPT_LOG] Ri: 0x1c25b90940f8a5daea346fdc83ed0d09
[DECRYPT_LOG] X:
↪ 0xc3082096d63be756f63feaf57f3418b90x1c25b90940f8a5daea346fdc83ed0d09
[DECRYPT_LOG] word: is

====================DECRYPT PROCESS====================
[DECRYPT_LOG] Ci:
↪ 62048d186f1cecd975465081e87538b2140ec7bddfc56856266ab2af5c2c7cb1
[DECRYPT_LOG] Cil,Cir: 62048d186f1cecd975465081e87538b2
↪ 140ec7bddfc56856266ab2af5c2c7cb1
[DECRYPT_LOG] Li: 0x4bd84c59136b4a1a624886e8247e4c0e
[DECRYPT_LOG] Fki: 6a5745da561604d080a66b2597f94370
[DECRYPT_LOG] Ri: 0x7e59826789d36c86a6ccd98acbd53fc1
[DECRYPT_LOG] X:
↪ 0x4bd84c59136b4a1a624886e8247e4c0e0x7e59826789d36c86a6ccd98acbd53fc1
[DECRYPT_LOG] word: surely

====================DECRYPT PROCESS====================
[DECRYPT_LOG] Ci:
↪ d6315741cacc4b7770a51d45a56c72fd31d0735b2441dc4d70d54bf08b19feb5
```

NKU

```
[DECRYPT_LOG] Cil,Cir: d6315741cacc4b7770a51d45a56c72fd
↪ 31d0735b2441dc4d70d54bf08b19feb5
[DECRYPT_LOG] Li: 0xffed9600b6bbedb467abcb2c69670641
[DECRYPT_LOG] Fki: 6a5745da561604d080a66b2597f94370
[DECRYPT_LOG] Ri: 0x5b8736817257d89df07320d51ce0bdc5
[DECRYPT_LOG] X:
↪ 0xffed9600b6bbedb467abcb2c696706410x5b8736817257d89df07320d51ce0bdc5
[DECRYPT_LOG] word: in

====================DECRYPT PROCESS====================
[DECRYPT_LOG] Ci:
↪ 37a7ab88b72e06262aa175058ce07471206fae35c527f0e57003e82995cf1d57
[DECRYPT_LOG] Cil,Cir: 37a7ab88b72e06262aa175058ce07471
↪ 206fae35c527f0e57003e82995cf1d57
[DECRYPT_LOG] Li: 0x1e7b6ac9cb59a0e53dafa36c40eb00cd
[DECRYPT_LOG] Fki: 6a5745da561604d080a66b2597f94370
[DECRYPT_LOG] Ri: 0x4a38ebef9331f435f0a5830c02365e27
[DECRYPT_LOG] X:
↪ 0x1e7b6ac9cb59a0e53dafa36c40eb00cd0x4a38ebef9331f435f0a5830c02365e27
[DECRYPT_LOG] word: China
================DISPLAY EMAIL=======================
[email] Nankai is surely in China
```

## (二) 倒排索引机制简单实现完整代码

```python
import hashlib
import os
import hmac
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes

class SymmetricSearchableEncryption:
    def __init__(self):
        self.forward_index = {}
        self.inverted_index = {}
        self.key = get_random_bytes(16)  # AES key must be either 16, 24, or 32
        ↪ bytes long

    def encrypt_word(self, word):
        # Encrypt the word using AES
        cipher = AES.new(self.key, AES.MODE_ECB)
        encrypted_word = cipher.encrypt(pad(word.encode(), AES.block_size))
```

NKU

```
18          return encrypted_word.hex()
19
20      def decrypt_word(self, encrypted_word):
21          # Decrypt the word using AES
22          cipher = AES.new(self.key, AES.MODE_ECB)
23          decrypted_word = unpad(cipher.decrypt(bytes.fromhex(encrypted_word)),
            ↪  AES.block_size)
24          return decrypted_word.decode()
25
26      def generate_trapdoor(self, encrypted_word):
27          # Generate trapdoor for the encrypted word using HMAC with SHA-256 and
            ↪  the secret key
28          return hmac.new(self.key, encrypted_word.encode(),
            ↪  hashlib.sha256).hexdigest()
29
30      def index_document(self, document):
31          # Encrypt and index the document
32          for word in document.split():
33              encrypted_word = self.encrypt_word(word)
34              trapdoor = self.generate_trapdoor(encrypted_word)
35
36              if encrypted_word not in self.forward_index:
37                  self.forward_index[encrypted_word] = set()
38              self.forward_index[encrypted_word].add(word)
39
40              if trapdoor not in self.inverted_index:
41                  self.inverted_index[trapdoor] = set()
42              self.inverted_index[trapdoor].add(encrypted_word)
43
44      def search(self, query):
45          # Encrypt the query word and generate trapdoor, then search in the
            ↪  inverted index
46          encrypted_query = self.encrypt_word(query)
47          trapdoor = self.generate_trapdoor(encrypted_query)
48
49          if trapdoor in self.inverted_index:
50              encrypted_results = self.inverted_index[trapdoor]
51              return encrypted_results
52          else:
53              return None  # No results found
54
55      def decrypt_results(self, encrypted_results):
56          # Decrypt the search results
57          decrypted_results = set()
```

NKU

```
58          if encrypted_results is None:
59              return {"No results found"}
60          for encrypted_word in encrypted_results:
61              try:
62                  decrypted_word = self.decrypt_word(encrypted_word)
63                  decrypted_results.add(decrypted_word)
64              except (ValueError, KeyError):
65                  decrypted_results.add("Decryption error or word not found")
66          return decrypted_results
67
68  # Example usage
69  scheme = SymmetricSearchableEncryption()
70  document = "This is a sample document with some words"
71  scheme.index_document(document)
72
73  query = "sample"
74  encrypted_results = scheme.search(query)
75  if encrypted_results is None:
76      print(f"No results found for query '{query}'")
77  else:
78      print(f"Encrypted results for query '{query}': {encrypted_results}")
79
80      decrypted_results = scheme.decrypt_results(encrypted_results)
81      print(f"Decrypted results for query '{query}': {decrypted_results}")
82
```

NKU

# 参考文献

[1] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Proceeding 2000 IEEE symposium on security and privacy. S&P 2000*, pages 44–55. IEEE, 2000.

NKU