《数据安全》实验报告

姓名： 费泽锟　学号： 2014074　班级： 信安班

实验名称：

对称可搜索加密方案实现

实验要求：

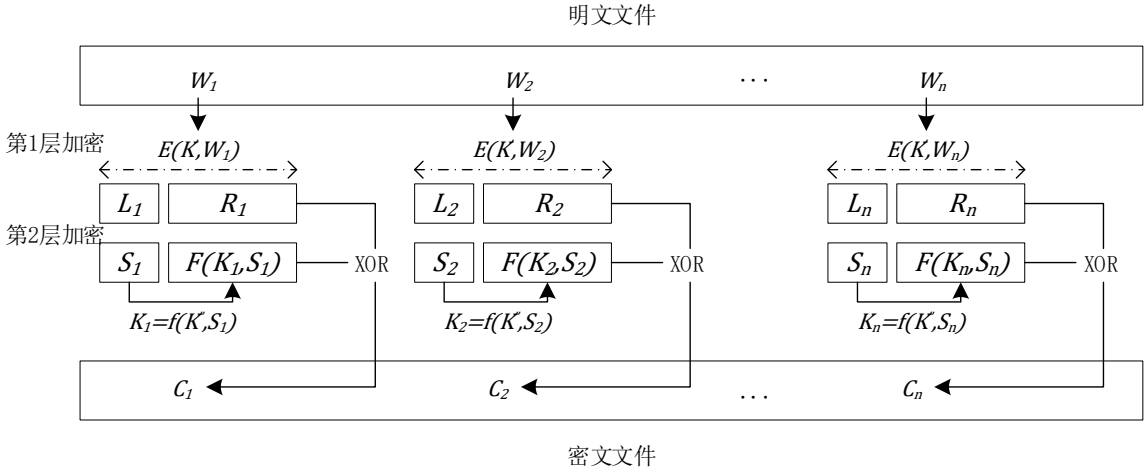
根据正向索引或者倒排索引机制，提供一种可搜索加密方案的模拟实现，应能分别完成加密、陷门生成、检索和解密四个过程。

实验过程：

1. 对于 SWP 方案的理解与探究

本次实验之中主要完成的是对 SWP 方案的模拟与复现，这里选择对于经典方案 SWP 方案进行复现的原因是，Z-IDX 方案中的布隆过滤器与倒排索引中的 SSE-1 方案中的对于数组与速查表的构造较为复杂，所以这里选择复现较为简单的 SWP 方案。

我们首先要对于 SWP 方案的原理进行学习探究，才能够成功复现。



我们先放上整体的架构图以便于我们来进行对照。

- 第一步：使用分组密码E逐个加密明文文件单词，这里我们选择的对称加密算法为 AES 加密算法，我们选择的长度为 128bit 的长度，也就是 16 字节的密钥大小，这个长度因为选择了 128bit，所以后续在处理随机序列S_i时需要进行一定的适应性改进。
- 第二步：将密文等分为L_i和R_i两部分，注意这里是等分，也就是对于 128bit 的二进制串等分为 64bit 的L_i和R_i两部分，我们在 AES 加密的过程之中无法保证二进制流为 128 的倍数，所以我们需要进行 pad 与 unpad，这里使用 pkcs7 的形式进行 padding，具体的片段代码如下图：

```
# 对明文加密
for item in message_list:
    cipher = AES.new(cipher_key, AES.MODE_CBC, iv)  # 创建一个aes对象
    text = item
    padtext = pad(text, 16, style='pkcs7')
    encrypted_text = cipher.encrypt(padtext)
    secret_messages_list.append(encrypted_text)
    print(encrypted_text)
```

- 第三步：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。这里的K''可以理解为伪随机序列生成器f的 seed，可以通过K''和L_i生成K_i,这里因为没有找到合适的伪随机序列函数 API,这里采用 SHA256 模拟f，hash 函数的结果也具有可证明的伪随机性，这里是将 seed 和L_i拼接后进行的计算。

  所以这里K''我们就设置为了“zzekun”，而接下来要根据K_i生成F的结果，这里的F我也使用的是 hash 函数，使用的是 MD5，计算方式是将随机序列S_i与K_i拼接后进行计算。

```
hash_key = b'zzekun'
for key, value in secret_messages_dict.items():
    secret_wrod_list = []
    for item in value:
        # 等分
        Li = item[:8] # 此时Li print出来后还是\x形式
        Ri = item[8:]
        # 使用MD5和SHA256作为伪随机函数
        Ki = f(hash_key + Li) # Ki为生成的F的密钥
        # print(Ki.hexdigest())  # str类型
        Ki = Ki.hexdigest()
        Si = random.getrandbits(16) # int类型
        Si = str(hex(Si))
        # print(Ki + Si)
        Fi = F((Ki + Si).encode())
        # 存储为(Si, Fi)用于验证
        secret_wrod_list.append((Si, Fi))
    # 存储
    secret_words_dict[key] = secret_wrod_list
```

- 第四步：异或$E(K',W_i)$和$S_i||F(K_i,S_i)$以形成W_i的密文单词。这样我们就能够得到每个文件对应的单词密文是什么了。但是这里有一个问题，那就是如果使用 AES 的 128bit 的密钥加密后的结果为 128bit，而 MD5 的运算结果也为 128bit，那么我们就无法将S_i的信息拼接存储进密文之中了，所以在模拟服务器端，这里进行一定的改进那就是使用 dict 的形式去存储(F_i,S_i)的对应关系，虽然这增大了服务器端的存储开销，但是因为使用了 hash 函数具有验证功能。

  我们也可以选择另一种方案，那就是使用 256bit 的 AES 加密，此时我们 MD5 哈希函数的结果为 128bit，那么我们就需要生成 128bit 的S_i，S_i根据每个文件中的关键词数量进行生成，如果我们采用 128bit 的 AES 加密，那么使用 16bit 甚至更少的S_i就行了，因为我们使用 dict 对其进行存储。

具体代码如下：

```
# 最后一步两个128bit数据进行逐bit异或
for i in range(1, len(message_list)+1):
    final_word_list = []
    for j in range(len(secret_messages_dict[i])):
        s1 = secret_messages_dict[i][j]
        s2 = secret_words_dict[i][j][1]
        s2 = s2.digest()
        l = bxor(s1, s2)
        final_word_list.append(l)
        # print(len(l))
        # print(type(l))
    final_secret_messages_dict[i] = final_word_list
```

- 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$E(K',W)$的左部），服务器顺序遍历密文文件的所有单词$C$，计算$C\ XOR\ E(K',W) = T$，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。（上述阐释的为 128bit 的 AES 加密）
- 至此我们就完成了对于 SWP 方案的学习与认识。

~~~~~~~

2. 具体实现细节

我们在是实现的过程之中，首先要对每个模拟文件也就是消息进行分词，在实现之中，我们考虑了标点符号的影响，这里我们模拟了如果两个!!感叹号连在一起出现的情形，如果我们使用空格进行分词，那么我们可以检索到!!这个字符串，同时我们也添加了能够单独检索单个!感叹号的关键词。

```
# 对于每条明文消息进行分词，对称加密
for i in range(len(message_list)):
    temp_word_list = message_list[i].split()
    # 为了提供这样的功能，就是对于两个感叹号!!，如果只搜索!也可以搜索到
    temp_word_list.append(b'!')
    enc_word_list = []
    for word in temp_word_list:
        cipher = AES.new(cipher_key, AES.MODE_CBC, iv)
        padword = pad(word, 16, style='pkcs7')
        # print(padword)
        # 此处将单个单词进行加密
        cipherword = cipher.encrypt(padword)
        # print(cipherword)
        enc_word_list.append(cipherword)
    secret_messages_dict[i+1] = enc_word_list
```

我们在实现 AES 加密的过程之中需要注意，要对密钥进行 pad，对消息进行 pad，同时我们在对每个关键词进行加密的过程之中，最好初始化新的 AES 对象，来保证加密的正确性，在实现的全程主要使用的是 Bytes 字符串的形式，保证了字符串长度的一致
~~~~~~~

来完成逐 bit 异或，实现方式如下：

```python
# use xor for bytes
def bxor(b1, b2):
    result = b''
    for b1, b2 in zip(b1, b2):
        result += bytes([b1 ^ b2])
    return result
```

~~~~~~~

3. 实验结果

我们的实验中的模拟消息如下：

```python
message_list = [b'Hello Alice !! Here is zzekun',
                b'Hello Alice !! Here is Bob',
                b'Hello Bob !! Here is zzekun',
                b'Hello Carol !! Here is zzekun',
                b'Hello Dean !! Here is zzekun',
                b'Hello Eva !! Here is zzekun',
                b'Hello world !! Here is zzekun']
```

如果我们选择的搜索关键词为“Alice”，那么我们应当获得的消息为第一条和第二条，我们的具体过程可在 log 信息之中观察到，log 信息展示如下：

```
C:\Users\25747\AppData\Local\Programs\Python\Python37\python.exe C:/Users/25747/PycharmProjects/data-security/SSE.py
b'zkzkzkzkzkzkzkzk'
search_Ki.digest:  03c131df9a4ddb349f993f9fae353dcc397fc0f0a7f4ca1d056ee066bac5e856
search_word:  b' \xed\xcd\xd8\xf0\xfd\xe1j\x81\xb1F\xa3\xa2\xe2\xf5\xac'
[1, 2]
b'\xc6\x95\xa4z\x94K\xee-\xb4\xdc\x82\x1cF\x90\xd8\x173\xb9\xff\xb1\xf2\xbd71\x17\xdc\\\x90\x0cb[5'
b'\xc6\x95\xa4z\x94K\xee-\xb4\xdc\x82\x1cF\x90\xd8\x17 i2\xfa\x06\xb6\x1eF/\xda\xca$b\xffE\xf7'
b'\xf7\xd0h\x88/+/e\x8f\x11M\xbdV\x82bq\x96\x8d"y:\x03\xe8\x06\x8b<\xf5\xb6\xef\xa7GV'
b'\x94c\xe7PZM\x8b-\xca1\x0e\x1e\x82I\x97\x9c\xab%~\xdb!R*4\xca`\x15^=\xac\x85\xa9'
b'N\x9aledI\xac\x01H\x8f\x9d\xff\x98l\xfb\xf0#\n\xec\x1d\x95\x8a0\xad\x8a\xb8\xd0\xee\x13\xffy4'
b']\x19\xd3O9\x1f\x05|(T\xd9@\xa9F\xc6\xd9\xb7\x9bC\xbb\x8e\xb8\x1f\x04\x8b\xec\xa7\xe6\xf1|\x10\x05'
b'I\x0b\xdb\xf0\xfc8D\xb6\xd1\x9e\x89g{\xd7\xbf\x19\xfd\xa0\xe3=\xcb\xa9\xf6\x9b\xbb\xa1\xba\x91\xed\xdb\xd4``'
b'Hello Alice !! Here is zzekun'
b'Hello Alice !! Here is Bob'

进程已结束，退出代码为 0
```

可以看到我们选择的密钥为：zkzkzkzkzkzkzkzk

我们生成的陷门 E 为 search_word 和 search_Ki

搜索到的消息索引为[1, 2]

对应的消息使用 AES 加密后的密文列表为接下来 7 行

最后为本地得到的解密明文为：

b’Hello Alice !! Here is zzekun’

b’Hello Alice !! Here is Bob’

具体的加密，陷门生成，检索和解密的过程的细节就不再阐述，具体代码可见链接：https://github.com/FZaKK/Data-Security
~~~~~~~

心得体会：

正向索引对称可搜索加密是一种基于关键字的加密方法，其中明文数据被分成若干段，并按照关键字进行索引。在加密过程中，明文数据被分成若干段，并对每一段进行加密。在搜索时，用户将关键字发送给服务器进行检索。服务器会将这些关键字与已加密的加密匹配，然后将匹配的加密数据返回给用户。用户再对返回的加密数据进行解密，得到明文数据。

在本次实验之中，复现了2000年提出的基于正向索引的对称可搜索加密方案SWP方案，因为伪随机序列生成器的问题，这里进行了一部分适应性的改进，完成了加密、陷门生成、检索和解密的四个过程，充分了解了相关的实验原理。