可搜索加密

计算机科学与技术 2010237 莫骐骥

随着云计算的迅速发展,用户开始将数据迁移到云端服务器,以避免繁琐的本地数据管理并获得更加便捷的服务.为了保证数据安全和用户隐私,数据一般是以密文存储在云端服务器中,但是用户将会遇到如何在密文上进行查找的难题.可搜索加密（searchable encryption,简称SE）是近年来发展的一种支持用户在密文上进行关键字查找的密码学原语,它能够为用户节省大量的网络和计算开销,并充分利用云端服务器庞大的计算资源进行密文上的关键字检索.

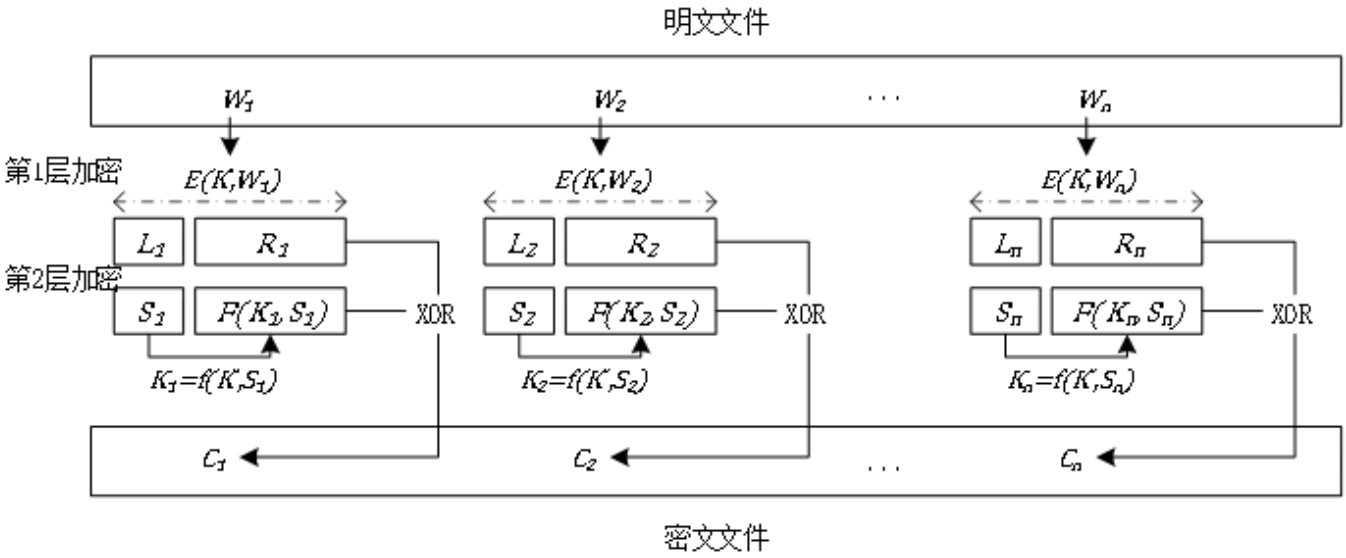
实验要求

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。

实验方案

本次实验采用基于正向索引的SWP算法,模拟可搜索加密中:加密,陷门生成,检索,解密四个过程.

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的关键词的密文文件。



代码细节

我们按加密、陷门生成、检索和解密的顺序介绍.

准备工作

- 要求每个单词第一轮加密后均为16字节,第二轮加密后左右两部分均为8字节.
- 采用AES ECB进行明文的加密.
- 采用sha256进行第二轮加密.

变量声明

```
key1 = b'computer|Science'  #第一层AES加密密钥
key2 = b'nankaiuniversity'  #第二层sha256加密密钥
aes1 = AES.new(key1, AES.MODE_ECB)  # 用于第一层加密
n = len(word_list)   # 文件单词数
random_flow = []    # 伪随机流S
encrypt_1st = []    # 第一层加密结果
encrypt_2nd = []    # 第二层加密结果
encrypt_data = []   # 加密后的数据
```

辅助函数构造

辅助函数构造:expand16B函数用于将单词通过末尾填0扩充为16字节;unexpand函数接收一个扩充后的单词,返回原单词;Bxor函数为byte数据的按位异或,返回两个byte数据按位异或的结果.

```
def expand16B(data):
    if len(data) < 16:
        data += '0' * (16 - len(data))
    return data

def unexpand(data):
    assert len(data) == 16
```

```
    res = ''
    i = 0
    while i < 16 and data[i] != '0':
        res += data[i]
        i += 1
    return res

def Bxor(bytes_a, bytes_b):
    parts = []
    for b1, b2 in zip(bytes_a, bytes_b):
        parts.append(bytes([b1 ^ b2]))
    return b''.join(parts)
```

加密

1. 在第一层，使用分组密码E逐个加密明文文件单词.
2. 在第二层,对分组密码E的输出进行处理：将密文等分为左L_i右R_i两部分,然后基于L_i生成二进制字符串$S_i||F(K_i.S_i)$，这里，$K_i = f(key2, L_i)$，$||$为符号串连接，F和f为伪随机函数(she256).
3. 异或E和$S_i||F(K_i.S_i)$以形成密文单词C_i.

```
# 第一层加密
    for i in range(n):
        temp = expand16B(word_list[i])
        encrypt_1st_i = temp.encode()
        assert len(encrypt_1st_i) == 16
        encrypt_1st.append(aes1.encrypt(encrypt_1st_i))
        if debug:
            print("encrypt_1st_i:", encrypt_1st[0])

    # 第二层加密
    for i in range(n):
        random_flow_i = get_random_bytes(8) # 伪随机流S
        random_flow.append(random_flow_i)
        K_i = hmac.new(key2, encrypt_1st[i][:8], digestmod = sha256).digest()[:16]
# 伪随机函数f生成的密钥
        F_ks_i = hmac.new(K_i, random_flow_i, digestmod = sha256).digest()[:8]   #
第二层的右半部分
        encrypt_2nd_i = random_flow_i + F_ks_i  # 左半部分和右半部分拼接
        encrypt_2nd.append(encrypt_2nd_i)
        if debug:
            print("random_flow_i:", random_flow_i)
            print("K_i:", K_i)
            print("F_ks_i:", F_ks_i)
            print("encrypt_2nd_i:", encrypt_2nd_i)

    # 生成加密数据
    for i in range(n):
        encrypt_data_i = Bxor(encrypt_1st[i], encrypt_2nd[i])
        encrypt_data.append(encrypt_data_i)
        if debug:
            print("encrypt_data_i:",encrypt_data_i)
```

陷门生成

陷门由两部分组成:输入关键词w使用第一轮加密的方法得到的E,以及第二轮加密过程中,用伪随机流生成F时的K.

$$T_w = (E(key1, w), K = f(key2, L))$$

其中L是E的左半部分.查询文件中是否包含关键词w,只需发送陷门至服务器即可.

```
key_word = input("Input key word:")
print(key_word)
# trapdoor生成
E = aes1.encrypt(expand16B(key_word).encode())
if len(E) != 16:
    raise("Key word too long.")
K = hmac.new(key2, E[:8], digestmod = sha256).digest()[:16]
if debug:
    print("E:", E)
    print("K:", K)
```

检索

服务器顺序遍历密文文件的所有单词C,计算$CxorE = S||T$,判断$F(K, S)? = T$,如果相等，C即为w在密文文件D中的密文；否则，继续计算下一个密文单词.

```
# 检索
is_found = False
for i in range(n):
    temp = Bxor(encrypt_data[i], E)
    S = temp[:8]
    T = temp[8:]
    F = hmac.new(K, S, digestmod = sha256).digest()[:8]
    if F == T:
        print("Find keyword.")
        is_found = True
        break
```

解密

对于检索到包含有w的密文文件,服务器应将此密文文件发送给用户,由用户完成对密文文件的解密,得到整个明文文件.

以用户对密文单词C的操作为例:用户将密文单词C拆解为左右等长两部分cl, cr,将cl与伪随机流S异或得到第一轮加密结果的左半部分l:

$$l = cl \bigoplus S$$

基于l,生成第二轮加密的右半部分f_{ks}:

$$ki = f(key2, l)$$

$$f_{ks} = F(ki, S)$$

将f_{ks}与cl异或得到第一轮加密的右半部分r.

$$r = cl \bigoplus f_{ks}$$

将第一轮加密的左右两部分拼接,再使用AES解密即可得到16字节的字节数据单词.将其转换成字符串,再调用unexpand函数即可得到明文单词.

以此对密文单词做上述处理,即可得到明文文件.

```
if is_found:
    decrypt_file = []
    for i in range(n):
        C = encrypt_data[i]
        cl = C[:8]
        cr = C[8:]
        l = Bxor(random_flow[i], cl)
        ki = hmac.new(key2, l, digestmod = sha256).digest()[:16]
        f_ks = hmac.new(ki, random_flow[i], digestmod = sha256).digest()[:8]
        r = Bxor(f_ks, cr)
        e = l + r
        if debug:
            print(encrypt_1st[i], e)
        decrypt_data = aes1.decrypt(e).decode()
        decrypt_file.append(unexpand(decrypt_data))
    print(decrypt_file)
```

实验结果

明文文件data.txt:

Coldplay are a British rock band formed in London in 1997. They consist of vocalist and pianist Chris Martin, guitarist Jonny Buckland, bassist Guy Berryman, drummer Will Champion and creative director Phil Harvey.They met at University College London and began playing music together from 1997 to 1998, initially calling themselves Starfish.

```
(pytorch) PS D:\Desktop\大三下\数据安全\对称可搜索加密方案\swp> & D:/anaconda3/envs/pytorch/python.exe d:/Desktop/大三下/数据安全/对称可搜索加密方案/swp/swp.py
Input key word:Coldplay
Coldplay
Find keyword.
['Coldplay', 'are', 'a', 'British', 'rock', 'band', 'formed', 'in', 'London', 'in', '1997.', 'They', 'consist', 'of', 'vocalist', 'and', 'pianist', 'Chris', 'Martin,
', 'guitarist', 'Jonny', 'Buckland,', 'bassist', 'Guy', 'Berryman,', 'drummer', 'Will', 'Champion', 'and', 'creative', 'director', 'Phil', 'Harvey.They', 'met', 'at'
, 'University', 'College', 'London', 'and', 'began', 'playing', 'music', 'together', 'from', '1997', 'to', '1998,', 'initially', 'calling', 'themselves', 'Starfish.'
]
(pytorch) PS D:\Desktop\大三下\数据安全\对称可搜索加密方案\swp>
```

输入关键词:Coldplay

输出Find keyword信息,并输出明文文件.

实验感想

通过实现swp算法,确实感受到此算法的效率问题.对每一次查询都需要遍历整个文件,占用大量服务器计算资源.