南開大學
Nankai University

计算机学院和网络空间安全学院

数据安全实验报告

频率隐藏 OPE 方法实现

姓名：魏伯繁

学号：2011395

专业：信息安全

2023 年 5 月 19 日

目录

1 实验要求

参照教材 6.3.3 FHOPE 实现完成频率隐藏 OPE 方案的复现，并尝试在 client.py 中修改，完成不断插入相同数值多次的测试，观察编码树分裂和编码更新等情况。

2 保留顺序加密

保留顺序加密技术的提出是为了解决纯密码学方案的密态数据库无法支持范围查询操作的问题。保留顺序加密技术在 2004 年被 Agrawal 等人提出。保留顺序加密方案是指保留顺序密文保留原有明文顺序的加密方案。当用户进行范围查询时，仅需将范围区间的端点密文发送给云端数据库即可得到所有属于该范围的密文数据，随后，客户端对密文数据进行解密即可得到最终结果。

保留顺序加密方案泄露了密文的序，因此无法达到 IND-CPA 安全，Boldyreva 等人提出了适合于保留顺序加密方案的泄露密文序的语义安全性定义，即 IND-OCPA，也被称为保留顺序加密方案的理想安全性。

顺序保留加密是指密文保留明文的序或者密文可比较的加密技术，即如果明文 a 和 b 满足 $a < b$，那么经过加密后的密文和也满足 $Enc(a)<Enc(b)$ 。这样做的优势是密文有序，无需额外操作，可以直接支持密文上的 Max、Min、Order by 等 SQL 操作。

2.1 典型构造

BCLO 方案是第一个可证明安全性的保留顺序加密方案。在该方案中，应用超几何分布和负超几何分布来构造保留顺序加密函数 $f : X- > Y$ 即：如果 $X_1 < X_2$，则 $f(X_1) < f(X_2)$。超几何分布是统计学上一种离散概率分布。它描述了从有限 N 个物件（其中包含 M 个指定种类的物件）中抽出 n 个物件，成功抽出该指定种类的物件的次数（不放回）。称为超几何分布，是因为其形式与 “超几何函数” 的级数展式的系数有关。基于保留顺序函数和超几何分布之间的天然联系提出了 BCLO 方案。假设明文域为 D，明文范围为 $[0, M]$, 密文域为 ，密文范围为 $[0, N]$，仅考虑整型数据。如图 6.2.1 所示，在加密明文 m 的时候，首先将明文与当前明文域的中点 x 相比。若不等于明文域中点，则明文域范围缩小。当 m 大于 x 时，当前明文域范围变为 $[x, max(D)]$；当小于时，当前明文域范围变为 $[min(D), x]$。其次，根据负超几何分布的要求，在当前密文域上选择一个 $f(x)$ 作为明文 x 所对应的密文。由于确定性种子的应用，使得每一个明密文对之间存在确定性关系。在当前明文域上继续重复上述步骤，直至明文 m 是当前明文域的中点或者当前明文域仅包含时停止，这时候即得到所对应的密文。

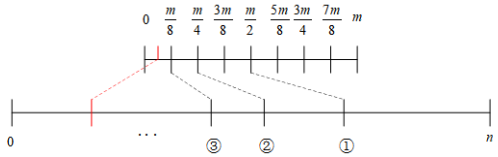

图 2.1: BCLO 示意图

2.2 顺序保留编码

顺序保留编码与顺序保留加密相对比，其密文不要求有顺序和可比较，由服务器端维护密文及其对应编码，并且服务器端还会额外维护索引树结构。

顺序保留编码通常具有较理想的安全性，可以达到 IND-OCPA 安全性。但是，顺序保留编码的客户端和服务器端存在交互。在插入操作过程中，客户端与服务器端多次交互遍历索引树生成密文编码，

更新编码表并将编码插入数据库。在查询操作过程中，客户端与服务器进行交互得到基于编码表的密文对应编码，然后再进行范围查询。

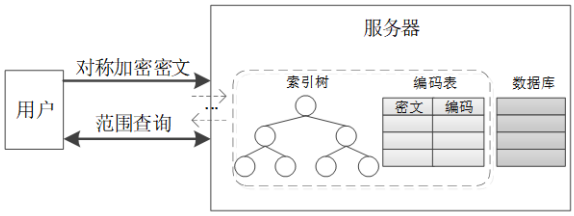

图 2.2: BCLO 示意图

Popa 等人是第一个实现 IND-OCPA 安全性的保留顺序加密方案（mOPE)。不同于一般的加密方案，Popa 等人将其定位为保留顺序编码方案。简单来讲，就是每一个明文对应一个保留顺序编码而不是一个保留顺序密文。在该方案中，采用语义安全的对称加密方式对明文进行加密，并将密文存储在客户端的二叉排序树中。通过客户端与服务器的多次交互，按照明文对二叉树中的节点进行排列。在此基础上，应用类似于霍夫曼编码的技术，对每个二叉树中的节点进行编码，即路径即为明文的保留顺序编码。

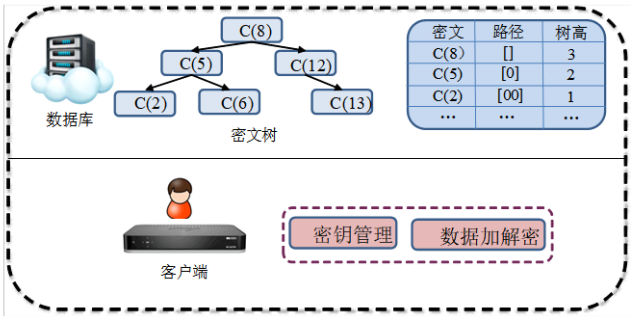

图 2.3: mOPE 方案

3 频率隐藏保序加密

2015 年，Kerschbaum 对现有的保留顺序编码方案进行了改进，提出了频率隐藏的保留顺序编码方案，进一步提高了安全性。该方案隐藏相同明文出现的频率，在一定程度上提升了方案的安全性，并抵御了一部分利用明文频率发起的攻击。该方案在客户端维护一个二叉排序树，将明文插入到二叉排序树中。通过参数的设定来减少排序树的调整。但是排序树一旦发生调整，将带来巨大的性能消耗。不仅如此，客户端的大存储，也使得该方案不实用。

当解密密文的时候，从根节点开始对索引树进行遍历，直到当前节点的保留顺序编码等于密文，则该节点的明文就是密文对应的明文。如果索引树进行重新调整，需要先将明文进行升序排列，然后将索引树组织成一个二叉平衡搜索树。在这个过程中，保证索引树的中序遍历结果与明文的升序排列相同。

3.1 无交互的 FH-OPE 方案

2021 年 Li 等人提出了一个小客户端存储且无额外交互的频率隐藏 OPE 方案，该方案主要针对易受频率攻击的字段提供了频率隐藏安全性，即来自小明文域中的大量数据记录。该方案同时为了降

低了编码更新的频率，提出了一个带有编码策略的 B+ 树。

为了在无额外交互下运行算法，该方案设置了一个本地表作为客户端存储，记录明文以及出现次数。每当新的明文被加密时，在本地表的帮助下，找出有多少现有的明文值小于 pt 和等于 pt 的。因此，很容易确定相应该明文在 B+ 树中的随机顺序。该方案改进了 B+ 树，每个中间节点不再存储关键词，而是存储节点后代中包含的密文的数量。从而完成在 1 次交互下将密文插入 B+ 树中。

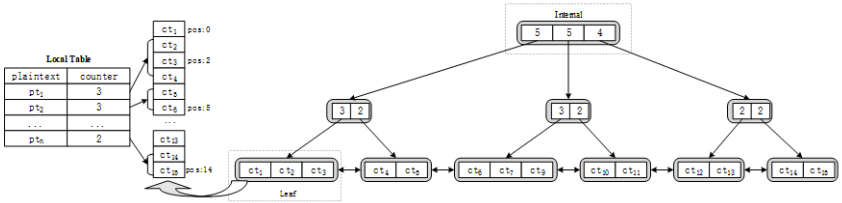

图 3.4: FH-OPE 示意图

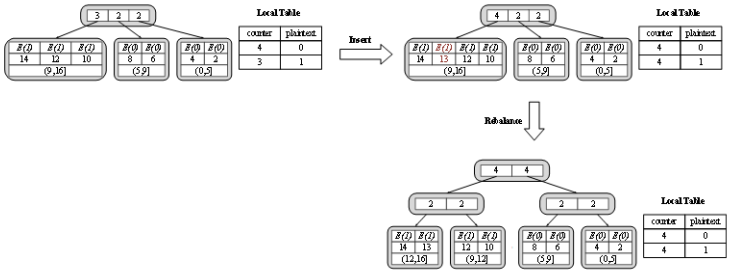

图 3.5: FH-OPE 示意图

4 实验探究

为了更好的完成实验探究任务，我们通过在源代码重要位置添加输出日志语句的方法进行补充，在计算位置的时候，我们首先根据每个名词在之前是否出现过进行判断，如果没出现过就直接输出没出现过以及其选择的位置，如果出现过就看一下目前可以选择的范围是哪里然后最后选择了哪里。

```
def CalPos(plaintext):
    # 插入 plaintext，返回对应的 Pos
    presum = sum([v for k, v in local_table.items() if k < plaintext])
    print("[CALPOS LOG] presum=",presum,end=' ')
    if plaintext in local_table:
        local_table[plaintext] += 1
        ret = random.randint(presum, presum + local_table[plaintext] - 1)
        print("plaintext exist before,range:[",presum,",",presum + local_table[plaintext] - 1,
        print("final choose:",ret)
        return ret
    else:
        local_table[plaintext] = 1
```

```
            print('plain test not exist before,final choose',presum)
            return presum
```

当插入元素时可以按照如下方式进行输出，这样就可以获得插入的明文的具体的内容以及其对应的插入位置

```
def Insert(plaintext):
    ciphertext = Random_Encrypt(plaintext)
    # 连接数据库
    conn = pymysql.connect(host='localhost', user='user',
                           passwd='123456', database='test_db')
    cur = conn.cursor()
    res = CalPos(plaintext)
    print("===[INSERT LOG] text = ",plaintext,end=' ')
    print('postion = ',res,"  ====")
    cur.execute(f"call pro_insert({res},'{ciphertext}')")
    conn.commit()
    conn.close()
```

4.1 实验效果

我们可以通过构建一个非常巧妙构造的输入序列来推理 B+ 树的生成过程，首先，我们插入 4 个 apple，然后插入一个比 apple 对应的值要大的 banana 来看 banana 是否插入到了 apple 的后面，然后我们再插入四个 apple，之后再插入一个 banana 观察前面的 apple 是否按照要求插到最前面了，然后最后我们插入一个介于 apple 和 banana 的内容，最后再插入一个 banana 观察 banana 节点位置的变化情况

```
if __name__ == '__main__':
    # 插入明文，同时设置了一部分重复的内容
    test_str = ['apple', 'apple', 'apple', 'apple', 'banana','apple', 'apple', 'apple', 'apple
    for ciphertext in test_str:
        Insert(ciphertext)
    print("==[MAIN LOG] local_table = ",local_table)
    # 假设我们搜索 b 和 p 之间的数据
    Search('aaaa', 'zzzzz')
```

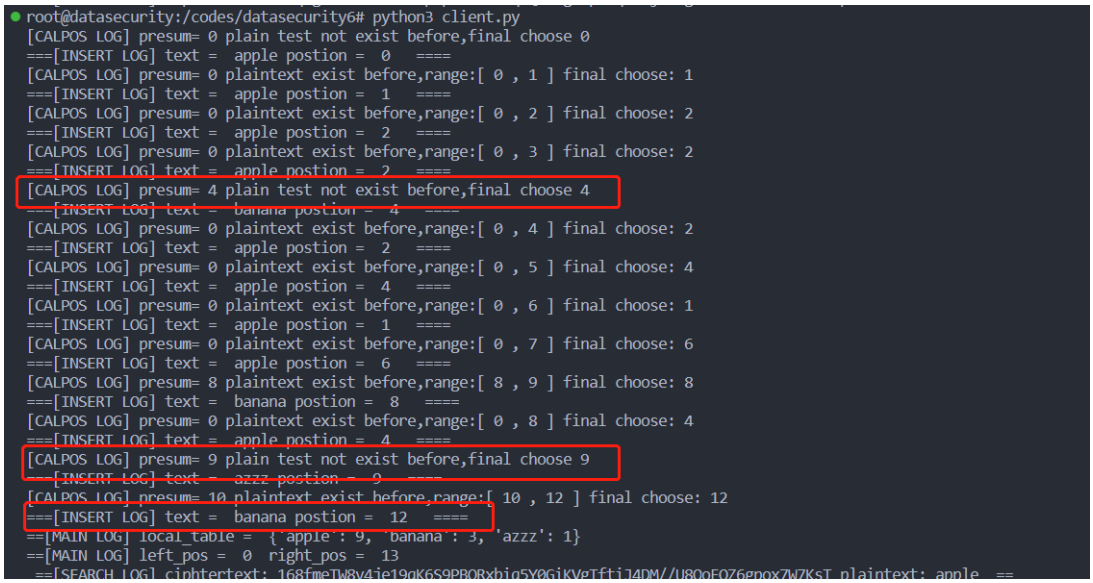

图 4.6: 实验结果示意图

接下来我们分析实验结果，首先插入 apple 发现其序号排列的 0、1、2、3，接下来插入 banana 则节点进入 4，因为 banana 大于 apple，所以其排列在序号 4 的位置，接下来我们再插入四个 apple 不难观察到其将 banana 原来的位置占据了，于是之后我们再插入一个 banana 发现他插入到了 8，而可选的范围是 [8,9] 说明 banana 正常的推后了，因为前面 0-7 都是 apple，接下来我们再插入一个 apple，此时 apple 的序号是 [0,8], 然后我们再插入一个'azzz', 因为 azzz 介于 apple 和 banana 之间，所以他回味与 [0,8] 之后，且之前没出现过 azzz，所以他没有选择的余地，只能选择 9，果然最后实验结果如图所示，azzz 插入到了 9 的位置，然后我们再插入一个 banana 进行验证发现其位置范围变味了 [10,12], 正确的后移了一位。

这样的排列方法是符合保留顺序加密的要求的，当然可以将对应内容选出：

图 4.7: 实验结果示意图

在数据安全课程中，我学习了保留顺序加密的明文插入过程，并通过已有的代码进行了实验。在这个过程中，我深刻了解了保留顺序加密的原理和实现方法，并从实践中收获了一些有趣的经验。

首先，我对保留顺序加密算法进行了深入的研究和学习，了解了其工作原理和设计思路。我研究了一些相关的论文和文章，并分析了已有的算法实现。在理解了这些基础知识之后，我开始运用已有的代码进行实验。

我要很好的设计一个测试样例来保证我能够观察到树的变化以及其编码变化，通过实践，我了解到这种加密方案可以很好地保护数据，同时也保留了数据的顺序，适用于很多场景。

在这个实验中，我获得了很多有意义的经验和体验。首先，我深入了解了保留顺序加密的基本原理和特点，并学会了如何在实践中进行实现。同时，我还学会了如何使用一些现有的安全技术和库，例

如加密算法和数据库。最后，我学会了如何设计和执行针对安全性的攻击，并同时了解到了如何保护数据免受攻击。

总之，通过这个实验，我充分了解了保留顺序加密算法，并掌握了如何在实践中进行实现和测试。这个实验不仅扩展了我的技能和知识，同时也加深了我对数据安全的认识。