《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：

频率隐藏 OPE 方案实现

实验要求：

参照教材 6.3.3 FH-OPE 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。

实验过程：

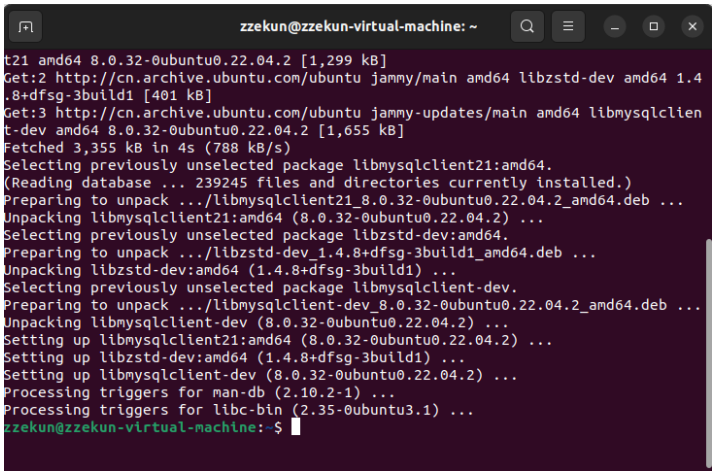
1. 配置相应的 VMware 实验环境

（1） 首先安装与配置 MySQL

使用如下命令完成 MySQL 以及其开发组件的安装：

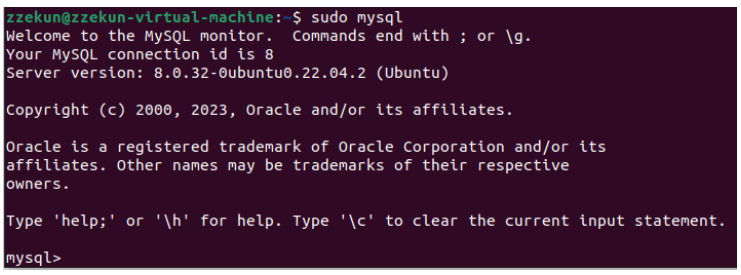
sudo apt install mysql-server libmysqlclient-dev

完成对应组件的安装部分：



接着使用如下命令创建 user 用户，我们先使用 sudo mysql 命令以 root 身份登录数据库，接着使用如下命令完成 user 用户的创建：

CREATE USER 'username'@'host' IDENTIFIED BY 'password';



```
mysql> CREATE USER 'user'@'%' IDENTIFIED BY '123456';
Query OK, 0 rows affected (0.09 sec)

mysql>
```

完成了 user 用户的创建后我们可以使用 grant 命令授予用户不同的权限，我们在这里直接将所有数据库上的所有权限授予给用户 user 即可，我们的 user 用户的密码就设置为简单的 123456 即可。

```
mysql> grant all on *.* to 'user'@'%';
Query OK, 0 rows affected (0.01 sec)

mysql>
```

接着我们使用命令 CREATE database test_db; 创建一个名为 test_db 的数据库

```
mysql> CREATE database test_db;
Query OK, 1 row affected (0.03 sec)

mysql>
```

至此我们的实验用的 test_db 数据库以及实验用的 user 用户就已经全部创建完毕，接下来进行的是 python 环境的完善。

（2） 完善实验的 python 环境

因为 Ubuntu22.04 虚拟机已经自带了 python3 环境，所以我们无需对 python3 进行额外的安装，我们需要使用 pip 工具下载 python pycryptodome 等第三方库，该库对各种加密函数提供相应的支持，所以我们需要使用命令：

sudo apt install python3-pip

下载 pip 工具，下载完成后，输入 pip 命令可以看到：

```
zzekun@zzekun-virtual-machine:~$ pip

Usage:
  pip <command> [options]

Commands:
  install                     Install packages.
  download                    Download packages.
  uninstall                   Uninstall packages.
  freeze                      Output installed packages in requirements format.
  list                        List installed packages.
  show                        Show information about installed packages.
  check                       Verify installed packages have compatible dependen
cies.
  config                      Manage local and global configuration.
  search                      Search PyPI for packages.
  cache                       Inspect and manage pip's wheel cache.
  index                       Inspect information available from package indexes
.
  wheel                       Build wheels from your requirements.
  hash                        Compute hashes of package archives.
  completion                  A helper command used for command completion.
```

实验中，在 client 需要使用 pycryptodome 第三方库中的加密函数，以及使用 pymysql 进行连接数据库，所以我们使用命令安装：

pip3 install pycryptodome pymsql

```
Collecting pycryptodome
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/14/58/77278d7a078241b55
b515f6073b90108125fb0d197b384a0f372c5f61c80/pycryptodome-3.17-cp35-abi3-manylinu
x_2_17_x86_64.manylinux2014_x86_64.whl (2.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 7.3 MB/s eta 0:00:00
ERROR: Could not find a version that satisfies the requirement pymsql (from vers
ions: none)
ERROR: No matching distribution found for pymsql
zzekun@zzekun-virtual-machine:~$ pip3 install -i https://pypi.tuna.tsinghua.edu.
cn/simple pycryptodome pymysql
Defaulting to user installation because normal site-packages is not writeable
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
Collecting pycryptodome
  Using cached https://pypi.tuna.tsinghua.edu.cn/packages/14/58/77278d7a078241b5
5b515f6073b90108125fb0d197b384a0f372c5f61c80/pycryptodome-3.17-cp35-abi3-manylin
ux_2_17_x86_64.manylinux2014_x86_64.whl (2.1 MB)
Collecting pymysql
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/5b/b1/bb485db528749f07d
6f11aa123e5f931f2e465a9c27945d6122bae5f7df7/PyMySQL-1.0.3-py3-none-any.whl (43 k
B)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43.7/43.7 KB 8.1 MB/s eta 0:00:00
Installing collected packages: pymysql, pycryptodome
Successfully installed pycryptodome-3.17 pymysql-1.0.3
zzekun@zzekun-virtual-machine:~$
```

至此我们的实验环境就已经搭建完毕。

~~~~~~~

2. 实验复现与修改观察编码树分裂与编码更新等情况

（1） 使用 touch 命令创建实验所需的源文件

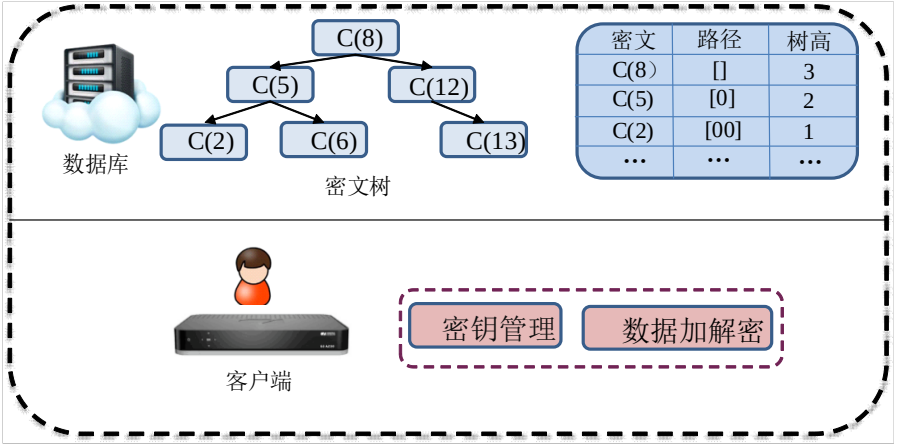
使用 touch 命令分别创建 Node.h，Node.cpp 和 UDF.cpp 文件

```
zzekun@zzekun-virtual-machine:~/FHOPE$ touch Node.h
zzekun@zzekun-virtual-machine:~/FHOPE$ touch Node.cpp
zzekun@zzekun-virtual-machine:~/FHOPE$ touch UDF.cpp
zzekun@zzekun-virtual-machine:~/FHOPE$
```

我们创建一个 FHOPE 目录，创建这三个源文件，创建完成后将实验代码填入到这三个源文件之中（这里建议使用 VSCode 打开后进行复制，便于阅读代码），然后我们需要对源代码进行理解与分析。

（2） 对 FH-OPE 方案进行回顾

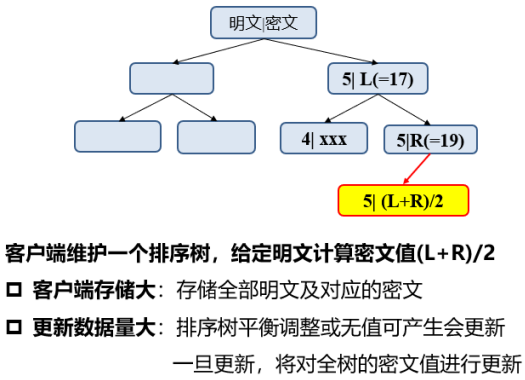
我们这里先对 FH-OPE 方案进行一个简单的回顾，以便于我们与源代码进行对照，



我们首先要对上图提到的 OPE 方案进行简单的回顾，上述提到的为 mOPE 方案，该方案虽然将明文，密文树都存储到了客户端上很难应用到现实生活之中，但是该方案的思
~~~~~~~

路也就是保留顺序编码为 FH-OPE 方案提供了基础。

下图即为 FH-OPE 方案的设计思路：



FH-OPE 方案实现的是频率隐藏保序加密，就是在 OPE 方案上为了抵抗频率以及统计攻击，而对于相同的明文得到的加密密文不一样，这里与 OPE 方案中的明文对应的密文隔一段时间更换不同，因为 OPE 方案之中相同的密文是直接返回的不会进一步进行操作。

与确定性保序加密不同之处在于密文$Enc(a)$和$Enc(b)$如果满足$Enc(a) < Enc(b)$则 a $\leq$ b。所以 FH-OPE 方案实现的就是在二叉树上在能够插入的位置上随机插入相等的明文对应的密文，该密文为(L+R)/2，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。

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t的保留顺序编码对应密文，则该节点的明文就是密文对应的明文。

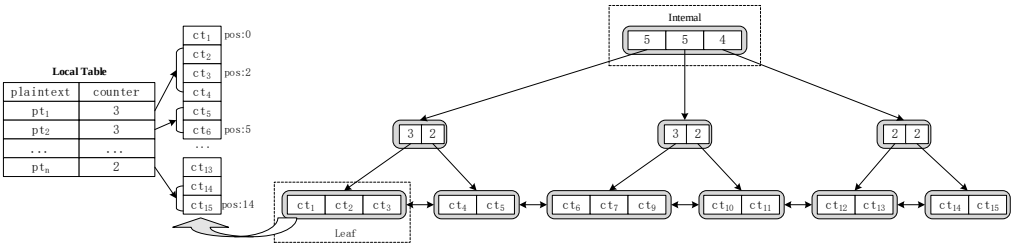
（3） 对源代码进行阅读分析

我们首先来看看 Node.h 中的代码，我们可以看到对树节点的定义，在我们对该树节点的所具有的属性进行查看的时候，我们发现本实验复现的应该是五交互的 FH-OPE 方案，而不是使用平衡二叉树的有交互的 FH-OPE 方案。

```
class LeafNode : public Node
{
public:
    std::vector<std::string> cipher; // 密文
    std::vector<long long> encoding; // 编码
    LeafNode *left_bro = NULL;       // 左兄弟节点
    LeafNode *right_bro = NULL;      // 右兄弟节点
    long long lower = -1;
    long long upper = -1;

    LeafNode();
    long long Encode(int pos);
    void rebalance() override;
    long long insert(int pos, string cipher) override;
    long long search(int pos) override;
};
```

我们可以看到，其中分为了叶节点和非叶节点，其中非叶节点不会存储具体的密文，而是会存储对应孩子节点的数目以及指针。具体的架构图如下：



无交互的 FH-OPE 方案通过存储在本地的本地表进行 pos 的确定和索引树的插入，利用 B+树实现保留顺序编码，因为 B+树可以减少密文的重新编码，实现局部编码的更新，而不至于像平衡二叉树那样需要对树结构进行重构。

我们对于五交互的局部更新方案进行回顾，无交互的方案采用区域编码策略：每个叶节点值区间为$(a,b]$，默认新节点编码策略：$(L+R)/2$，L为左邻居的编码，R为右邻居的编码。节点内编码更新策略：区间$(a,b]$、密文个数为c，更新后$[1*(a+(b-a)/c),\cdots,c*(a+(b-a)/c)]$。在这种情况下，树的平衡调整不会引起编码的更新，插入可能引发节点内数据密文编码更新，但是其他节点不发生更新。

我们继续阅读分析源代码：

```
if (this->child.size() >= M)
{
    this->rebalance();
}
```

我们在 Node.cpp 文件之中能够看到如上的代码，也就是 B+树子节点的指针数量达到了 M 之后就需要进行编码树的重构，但是这里的 M 在 Node.h 之中的设置为 128，还是较大的。这里是我们可能需要后续进行稍作修改的。

```
if ((right_bound - left_bound) > total_cipher_num)
{
    // 如果当前的更新区间，足以包含待放的pos
    start_update = left_bound;
    end_update = right_bound;
    // 计算间隔量，使code均匀分布
    long long frag = floor((right_bound - left_bound) / total_cipher_num);
    assert(frag >= 1);
    long long cd = left_bound;
    for (size_t i = 0; i < node_list.size(); i++)
    {
        node_list.at(i)->lower = cd;
        for (int j = 0; j < node_list.at(i)->encoding.size(); j++)
        {
            node_list.at(i)->encoding.at(j) = cd;
            update.insert(make_pair(node_list.at(i)->cipher.at(j), cd));
            cd = cd + frag;
        }
        node_list.at(i)->upper = cd;
    }
    node_list.back()->upper = right_bound;
}
```

这里我们能够看到具体的局部重新编码策略，也就是如果我们取到的节点编码的更新区间足以放下所有的密文的话，那么就可以不同调整，如果更新区间不够的话就需要向左兄弟节点和右兄弟节点扩展更新区间，也就是 B+树的扩展策略。

```
else
{
    // 否则直接以left和right的平均值向上取整作为新的code
    unsigned long long re = right;
    long long frag = (right - left) / 2;
    re = re - frag;
    this->encoding.at(pos) = re;
    return this->encoding.at(pos);
}
```

编码策略：当空间充足时，直接以 left 和 right 的平均值向上取整作为新的 code

```
long long FHInsert(UDF_INIT *initid, UDF_ARGS *args, char *is_null, char *error)
{
    int pos = *(int *)(args->args[0]);
    double keyLen;
    char *const keyBytes = getba(args, 1, keyLen);
    const std::string cipher = std::string(keyBytes, keyLen);
    long long start_update = -1;
    long long end_update = -1;
    update.clear();
    long long re = root->insert(pos, cipher);
    return re;
```

而在 UDF.cpp 之中则包含着使用了节点索引树的插入和搜索等初始化以及自定义函数。其中 insert 函数的第一个参数为 pos 也就是插入的位置，还有输入的第二个参数为 key，并且会通过 getba 函数进行处理，获得 key 的长度，这里从后续来看，这里的 key 应该就是明文对应的密文了，即为 cipher

好了我们对于这三个源文件的代码做了什么已经有了基本的了解，我们来继续进行实验的复现，我们使用如下命令编译 so 动态链接库，注意这与教材之中有点不同，需要使用 Node.cpp 而不是 FH-OPE.cpp：

g++ -shared -fPIC UDF.cpp Node.cpp -lcrypto -o libfhope.so

```
zzekun@zzekun-virtual-machine:~/FHOPE$ g++ -shared -fPIC UDF.cpp FH-OPE.cpp -lcr
ypto -o libope.so
cc1plus: fatal error: FH-OPE.cpp: No such file or directory
compilation terminated.
zzekun@zzekun-virtual-machine:~/FHOPE$ g++ -shared -fPIC UDF.cpp Node.cpp -lcryp
to -o libope.so
zzekun@zzekun-virtual-machine:~/FHOPE$
```

成功生成.so 动态链接库文件。

接下来将其拷贝到 MySQL 的文件夹：

sudo cp libfhope.so /usr/lib/mysql/plugin/

我们完成了文件复制，接下来将其中的函数导入到 MySQL 中。

```
root@zzekun-virtual-machine:/usr/lib/mysql/plugin# ls
adt_null.so                        keyring_udf.so
auth_socket.so                     libmemcached.so
component_audit_api_message_emit.so libope.so
component_keyring_file.so          libpluginmecab.so
component_log_filter_dragnet.so    locking_service.so
component_log_sink_json.so         mypluglib.so
component_log_sink_syseventlog.so  mysql_clone.so
```

接着我们创建一个 load.sql 文件创建插入数据的存储过程，树结构中更新了编码同步更新数据库的信息。接着我们登录 MySQL 进行操作。

```
zzekun@zzekun-virtual-machine:~/FHOPE$ mysql -uuser -p123456
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 11
Server version: 8.0.32-0ubuntu0.22.04.2 (Ubuntu)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

```
mysql> use test_db
Database changed
```

在 source 使用 sql 文件的时候需要注意修改一点，将 sql 文件之中的 libfhope.so 字样改为 libope.so 字符串，或者编译阶段将 so 文件名进行修改。

```
Query OK, 0 rows affected, 1 warning (0.00 sec)

Query OK, 0 rows affected, 1 warning (0.00 sec)

Query OK, 0 rows affected, 1 warning (0.00 sec)

Query OK, 0 rows affected, 1 warning (0.01 sec)

Query OK, 0 rows affected (0.01 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.01 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql>
```

至此我们对 MySQL 数据库的操作就已经完成了，接下对 client 客户端进行模拟操作。

使用 touch 命令创建 client.py，将实验指导书的代码复制进去，我们就可以进行 client 客户端模拟了。通过 python3 指令可直接执行 py 文件，可以看到 search 命令的结果，也就是 b-p 开头的字符串，及其明文：

```
zzekun@zzekun-virtual-machine:~/FHOPE$ python3 client.py
ciphtertext: POveqIRFID/RY8ENupA8AFE4tthQkLVwhEbCj39PwC5Nf5FylGcKXGeipIlWoxC/ plaintext: banana
ciphtertext: oKU6WsarMhAcChoEzgJ4xWD3MkoM07PwhKLjxWfGg65WIYLHQlkNybk85aimic0y plaintext: orange
ciphtertext: VXaFNLsuR+MVeOKtb1ERN71VLuRH/+Mv47LoYM8ObLDgzvWLfJAOrDkaJa7TAgzO plaintext: cherry
ciphtertext: C1UsZWC0CxdH4z66BOXLKWZw6D2YkiapGk3dxlDc0zjaAYoAcfh2VSkpzC2Q/JZc plaintext: cherry
ciphtertext: /ASuBJ1JkvHKAsZF6LT86kfkYSesJGlDt+N2IBoczzXuotxP705M0yXttkUgWbHA plaintext: orange
zzekun@zzekun-virtual-machine:~/FHOPE$
```

至此我们对于实验 6.3.3 复现就已经全部完成了，下一部分是对 client 部分的代码的一点点修改与编码分析。

~~~~~~~

3. 实验内容分析与总结

我们首先来观察这个 example 数据表都存了一些什么数据，因为实验的要求是通过修改 client.py 来观察对应的编码树的更新与分裂的情形，如果想要进一步去查看每一个
~~~~~~~

节点对应在树中的自身或者父节点的 depth，那么就需要这么做：先在 UDF 之中设计一个新的能够返回 depth 的用户自定义函数，并且在 Node.h 与 Node.cpp 之中支持这样的查询操作，这显然无法仅仅通过修改 client.py 来实现，所以我们采取另一种策略，那就是通过 encoding 的编码策略进行观察和分析。

我们在 client.py 之中补充一段查询语句如下：

select ciphertext, encoding from example where encoding >= FHSearch({left_pos}) and encoding < FHSearch({right_pos})

```
    print(f"ciphtertext: {x[0]} plaintext: {Random_Decrypt(x[0])} ")
# here we can show the plaintext encoding
cur.execute(
    f"select ciphertext, encoding from example where encoding >= FHSearch({left_pos}) and encoding < FHSearch({right_pos})")
rest = cur.fetchall()
for x in rest:
    print(f"ciphtertext: {x[0]} plaintext: {Random_Decrypt(x[0])} encoding: {x[1]}")
```

这样我们就能够查看到每个节点的 encoding 了，我们仍然需要修改一个地方，那就是之前提到的 M 参数，如果设置 M 为 128 那么其实很难看到分裂的现象，我们可以把这个 M 设置为 4，这样能够看到较大的 encoding 变化，我们重复的插入“banana”字符串可以看到查询的结果如下：

```
ciphtertext: BxY53xT0rfCUfwTfJqikw/AEgRTDxFCZyOH7sFcNDg2WkaS2WRU+H4CQAzkFFD3w plaintext: banana encoding: 180143985094819840
ciphtertext: 5aZEsnmaCYGafjDfBaPY7tAVmFFtHqQK7jrfxxgoEIhk4JxMi6rM5hjwcSyFUAZT plaintext: banana encoding: 162129586585337856
ciphtertext: 7QxHThy0RCsWcTa+0UY5sUYs4qz6gcK61u2CJBaJscEMcTcxgbYlJBHl/Iu36KqZ plaintext: banana encoding: 198158383604301824
ciphtertext: nrnJxkgvMdC491CR7DHN7vnaw14Lj4Sww715wMhEFKO7gY2C5DxHE/rNwUPek+Mf plaintext: banana encoding: 153122387330596864
ciphtertext: iRmGF3uKjXecnZCP71luswdnoBnFkA8ytPdByWrwexH0bbo65cT5uEBuH04MBaIq plaintext: banana encoding: 171136785840078848
ciphtertext: ry7HWUDQPUugQl+J5vcfeFjMI1WSaZTVXD66CN468gRDcE6T1t+ERzLYWGkdshve plaintext: banana encoding: 166633186212708352
ciphtertext: m1hn4Z+W2CQgjVMSyo1ZYmmq8HoCYfgiUL19ZNHZrvQQCwjf6pxNYcFySXnfYhQK plaintext: banana encoding: 157625986957967360
ciphtertext: Y1Yq0HVkYBAzN5w5D6JmTZFITJKinN5cwgO2ghRRbmiyxIPnSiDYcrwhZ902MMDr plaintext: banana encoding: 175640385467449344
ciphtertext: DYMywBbcJBgI/eKWgT0N5gS6JhGcQMQPWbzOSyBXI+1wUufBqnQVIp2kioaj0MhX plaintext: banana encoding: 207165582859042816
ciphtertext: C0oInEfjgPp/SbUuOiYVTQ2fvKSghAqNAA4hT3cKKxvnMiRCP/6TyKngVlc5CcKE plaintext: banana encoding: 202661983231672320
ciphtertext: zkgniWEGBEjeU1Vew6SQfDhU+gBWYLgQzzULBn8zHwb62KYsJ0zsvJAlta6fxg19 plaintext: banana encoding: 211669182486413312
ciphtertext: o6WEFy/9WzBDmM/wdWvHqoOibWRr8E8+9N4lAEeEiRmsIN3Rg9AnFMcbaQMfpAnE plaintext: banana encoding: 209417382672728064
ciphtertext: wQECzJR7E/CVZjHBYSew0jiJySvmbNqZxkTMSxhICSxEKYW5HxLyAF/QK7tesTBJ plaintext: banana encoding: 155374187144282112
ciphtertext: IaOPejSd3g78maD8Jp3XUxQ/0k7935cF1DzNz5QOwlHf46kO2LaEtFMZjNMYd/mc plaintext: banana encoding: 156500087051124736
ciphtertext: 45nwTGHUTzRKK6jAXRGnKdx0GEriBhmcURFL9UK8IC7i3Nej1jGuMJKZB0auzpzy plaintext: banana encoding: 155937137097703424
ciphtertext: aid/UVxyIWGJ7HSQpvRLjvygFmoH/mx61v44gBUkQ9z3uSRoNaBBFTaA/CGzXOjW plaintext: banana encoding: 213920982300098560
ciphtertext: 1J/jlmjAm9qBijdHWHBBfbF9pa8wbSEn0NWY2HWQUDAXLOz9tU4AAC697Rbo0p2i plaintext: banana encoding: 208291482765885440
ciphtertext: RrLCVjLc0kTPBzaQMO8O/4yqnouUKdf2XmNZthReI/MKYzI5Rf8ZfkMRdA2ukVF6 plaintext: banana encoding: 177892185281134592
ciphtertext: c4pB3iQyp9I2C4OiJiNnkofCrWRX/oz5MwED6XjHBjHC98tHcFGE/+jbCtMhPcfP plaintext: banana encoding: 164381386399023104
ciphtertext: RGwLH6l5XjEBHzCziyU3XBUtZuwWWwvTjgGwFwBEUj26C3KsXwOuD2WcGaCXXsAn plaintext: banana encoding: 156218612074414080
ciphtertext: JHs/dXQZAxls4LImxFrVPp7TiI1DFzf3Bq4GBOpYAEpH5DBxjzlZDVQo/MzLyrcq plaintext: banana encoding: 163255486492180480
ciphtertext: gnvClAKT0PBm/1igLWQ1v8SmgpI7hnqPbM4mkJ8rFWsMPrXeKqJs2gXsmh3JRNOX plaintext: banana encoding: 154248287237439488
ciphtertext: F3U5xnO6XHDlMvPvWFa6SjsGTuUGghnpGrKWAW+czMDvlistueEGTJwiADeSsuPy plaintext: banana encoding: 215046882206941184
ciphtertext: /pB/EFWy2KqM6MxkR/HoLvVd1OPydiSYAbwtwVwKUDdnwxcELAzVKSmB7m2VmAn5 plaintext: banana encoding: 163818436445601792
```

可以观察到 banana 的 encoding 从 15 为首到 21 为首不等，且其中覆盖着 15-16-17-18-19-20 为首的 encoding，这里的为首因为数值过大只是用前两位进行表示，从这也就表明了无交互的 FH-OPE 方案的保留顺序编码策略。

心得体会：

OPE（Order-Preserving Encryption）是一种保留排序顺序的加密方法。在 OPE 中，加密后的密文和原明文的排序顺序是一致的。但是，OPE 无法隐藏明文中的频率信息，因为相同的明文将加密为相同的密文，而密文的数量可以被统计。

为了解决这个问题，FH-OPE（Frequency-Hiding OPE）方案被提出。FH-OPE 将 OPE 和 FHE（Fully Homomorphic Encryption）相结合，加密的密文不同且插入的位置无法被统计。

FH-OPE 方案是一种保留排序顺序并隐藏频率信息的加密方法。通过 FHE 和 OPE 的结合，实现了频率信息的隐藏，但是也增加了计算和存储的复杂度。在实验中，我们成功地实现了 FH-OPE 方案，此外，我们还发现，由于不同数值被映射到不同的明文空间中，插入相同数值多次会导致编码树的分裂和更新，并观察到了编码树的分裂和更新情况。