数据安全 -- 频率隐藏OPE

学号：2013921
姓名：周延霖
专业：信息安全

一、实验名称

频率隐藏OPE

二、实验要求

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尝试，观察编码树分裂和编码更新等情况

三、实验过程

1、环境准备

> MySQL的安装与配置

- Ubuntu安装MySQL

本次实验需要通过UDF函数将其植入到MySQL中，因此需要安装MySQL以及开发组建，命令如下：

```
[sudo] apt install mysql-server libmysqlclient-dev
```

- 创建用户

首先通过如下指令以root身份登陆数据库：

```
[sudo] mysql
```

如下图所示：

```
ubuntu@laptop:~$ sudo mysql
[sudo] password for ubuntu:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 11
Server version: 8.0.30-0ubuntu0.22.04.1 (Ubuntu)

Copyright (c) 2000, 2022, 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>
```

接下来可以通过create user指令创建用户，结构如下：

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

创建完成后，可以通过grant命令授予用户不同的权限。例如创建一个用户名为user，不限制host，密码为123456的账户，并将所有数据库上的所有权限授予用户user：

如下图所示：

```
mysql> create user 'user'@'%' identified by '123456';
Query OK, 0 rows affected (0.04 sec)

mysql> grant all on *.* to 'user'@'%';
Query OK, 0 rows affected (0.03 sec)
```

- 创建数据库

使用如下指令创建数据库：

```
CREATE database <name>;
```

例如可以创建一个test_db的数据库，如下图所示：

```
mysql> drop database test_db;
Query OK, 1 row affected (0.06 sec)
```

> Python3环境

- Ubuntu安装Python3

使用APT包管理即可方便的安装python3:

```
[sudo] apt install python3
```

需要注意的是，Ubuntu16.04之后已经默认包含python3环境，无需而外安装

- 安装pip工具并安装所需包

Ubuntu初始环境仅包含python，并没有pip工具，所以需要通过如下命令安装：

```
[sudo] apt install python3-pip
```

本实验在实现时，在client需要用到pycryptodome中的加密函数，以及使用pymysql连接数据库，安装命令如下：

```
pip3 install pycryptodome pymsql
```

如果下载速度较慢，也可通过-i使用第三方源，例如使用清华源：

```
pip3 install -i https://pypi.tuna.tsinghua.edu.cn/simple pycryptodome
pymsql
```

2、编写Server端程序

- 根据FH-OPE描述，编写Node.h文件

```
#pragma once
#include <vector>
#include <map>
#include <string>
using namespace std;

class Node
{
public:
    int type;         // 用来区分InternalNode和LeafNode
    int parent_index; // 记录当前结点是父结点的第几个孩子
    Node *parent = NULL;
    virtual void rebalance(){};
    virtual long long insert(int pos, string cipher) { return 0; }; // 插入
新的密文
    virtual long long search(int pos) { return 0; };    //查找pos对应的code
};

class InternalNode : public Node
{
public:
    std::vector<int> child_num; // 子节点具有的加密值个数
    std::vector<Node *> child;  // 子节点指针
```

```
    InternalNode();
    void rebalance() override;
    long long insert(int pos, string cipher) override;
    long long search(int pos) override;
    void insert_node(int index, Node *new_node); // 插入新的Node
};

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;
};

const int M = 128;
extern Node *root;
extern long long start_update;  //  更新区间的左端点
extern long long end_update;    //  更新区间的右端点
extern std::map<string, long long> update;

void root_initial();                  // 初始化
long long get_update(string cipher); // 根据密文获取对应的更新后的code
```

- 在同文件夹下编写对应的Node.cpp文件

```
#include "Node.h"
#include <array>
#include <math.h>
#include <assert.h>
#include <vector>
#include <map>
#include <fstream>

Node* root = nullptr;
long long start_update = -1;
long long end_update = -1;
std::map<string, long long> update;

InternalNode::InternalNode()
{
    this->type = 2;
```

```
    this->parent_index = -1;
    this->parent = NULL;
}

void InternalNode::insert_node(int index, Node *new_node)
{
    this->child.insert(this->child.begin() + index, new_node);

    if (new_node->type == 1)
    {
        // 如果新结点是LeafNode
        this->child_num.insert(this->child_num.begin() + index, ((LeafNode
*)new_node)->cipher.size());
        ((LeafNode *)new_node)->parent = this;
    }
    else
    {
        // 如果是InternalNode
        int res = 0;
        for (size_t i = 0; i < ((InternalNode *)new_node)-
>child_num.size(); i++)
        {
            res += ((InternalNode *)new_node)->child_num.at(i);
        }
        this->child_num.insert(this->child_num.begin() + index, res);
        ((InternalNode *)new_node)->parent = this;
    }

    // 插入的新node改变了原有node位置，因此需要重新记录parent_index和child_num
    for (int i = 0; i < this->child.size(); i++)
    {
        this->child.at(i)->parent_index = i;
        if (this->child.at(i)->type == 1)
        {
            LeafNode *tmp = (LeafNode *)this->child.at(i);
            this->child_num.at(i) = tmp->cipher.size();
        }
    }
    if (this->child.size() >= M)
    {
        this->rebalance();
    }
}

void InternalNode::rebalance()
{
    InternalNode *new_node = new InternalNode();
    // 将当前结点的后半部分数据存入new_node
    int middle = floor(this->child.size() * 0.5);
    while (middle > 0)
    {
        new_node->child.insert(new_node->child.begin(), this-
>child.at(this->child.size() - 1));
        new_node->child_num.insert(new_node->child_num.begin(), this-
```

```
>child_num.at(this->child_num.size() - 1));
        this->child.pop_back();
        this->child_num.pop_back();
        middle--;
    }
    for (int i = 0; i < new_node->child.size(); i++)
    {
        new_node->child.at(i)->parent_index = i;
        new_node->child.at(i)->parent = new_node;
    }

    if (!this->parent)
    {
        // 如果当前结点是root结点
        InternalNode *new_root = new InternalNode();
        new_root->insert_node(0, this);
        new_root->insert_node(1, new_node);
        root = new_root;
    }
    else
    {
        int res = 0;
        for (size_t i = 0; i < this->child_num.size(); i++)
        {
            res += this->child_num.at(i);
        }
        ((InternalNode *)this->parent)->child_num.at(this->parent_index) =
res;
        ((InternalNode *)this->parent)->insert_node(this->parent_index +
1, new_node);
    }
}

long long InternalNode::insert(int pos, string cipher)
{
    for (int i = 0; i < this->child.size(); i++)
    {
        if (pos > this->child_num.at(i))
        {
            pos = pos - this->child_num.at(i);
        }
        else
        {
            this->child_num.at(i)++;
            return this->child.at(i)->insert(pos, cipher);
        }
    }
    // 如果没有符合的，则放到最后一个
    this->child_num.back() = this->child_num.back()++;
    return this->child.back()->insert(pos, cipher);
}

long long InternalNode::search(int pos)
{
```

```
    int i = 0;
    for (; i < this->child.size(); i++)
    {
        if (pos < this->child_num.at(i))
        {
            return this->child.at(i)->search(pos);
        }
        else
        {
            pos = pos - this->child_num.at(i);
        }
    }
    return 0;
}

LeafNode::LeafNode()
{
    this->type = 1;
    this->parent_index = -1;
    this->parent = NULL;
}

void Recode(vector<LeafNode *> node_list)
{
    long long left_bound = node_list.at(0)->lower;
    long long right_bound = node_list.back()->upper;
    int total_cipher_num = 0;

    for (size_t i = 0; i < node_list.size(); i++)
    {
        total_cipher_num += node_list.at(i)->cipher.size();
    }

    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;
        }

        else
        {
            // 若不足以包含，则继续向左兄弟结点和右兄弟结点扩展更新区间
            if (node_list.at(0)->left_bro)
            {
                // 如果左兄弟存在，则加入更新列表
                node_list.insert(node_list.begin(), node_list.at(0)-
>left_bro);
            }

            if (node_list.back()->right_bro)
            {
                // 如果右兄弟存在，则加入更新列表
                node_list.push_back(node_list.back()->right_bro);
            }
            else
            {
                // 扩展最后一个结点的大小
                node_list.back()->upper = node_list.back()->upper * 2;
                if (node_list.back()->upper >= pow(2, 60))
                    node_list.back()->upper = pow(2, 60);
            }
            Recode(node_list);
        }
    }

    long long LeafNode::Encode(int pos)
    {
        long long left = this->lower;
        long long right = this->upper;

        if (pos > 0)
        {
            left = this->encoding.at(pos - 1);
        }
        if (pos < this->encoding.size() - 1)
        {
            right = this->encoding.at(pos + 1);
        }

        if (floor(right - left) < 2)
        {
            // 如果该区间已经没有位置，则需要recode
            std::vector<LeafNode *> node_list;
            node_list.push_back(this);
            Recode(node_list);
            // 返回特殊值0，标志已经调整，需要更新数据库
            return 0;
        }
        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);
    }
}

void LeafNode::rebalance()
{
    LeafNode *new_node = new LeafNode();
    // 将本结点后半部分数据放到new_node
    int middle = floor(this->cipher.size() * 0.5);
    while (middle > 0)
    {
        new_node->cipher.insert(new_node->cipher.begin(), this-
>cipher.back());
        new_node->encoding.insert(new_node->encoding.begin(), this-
>encoding.back());
        this->encoding.pop_back();
        this->cipher.pop_back();
        middle--;
    }
    // 将新结点连接在this结点右边
    new_node->lower = new_node->encoding.at(0);
    new_node->upper = this->upper;
    this->upper = new_node->encoding.at(0);
    if (this->right_bro)
    {
        this->right_bro->left_bro = new_node;
    }
    new_node->right_bro = this->right_bro;
    this->right_bro = new_node;
    new_node->left_bro = this;

    if (!this->parent)
    {
        // 如果this是root结点，则创建一个新的root
        InternalNode *new_root = new InternalNode();
        new_root->insert_node(0, this);
        new_root->insert_node(1, new_node);
        root = new_root;
    }
    else
    {
        ((InternalNode *)this->parent)->child_num.at(this->parent_index) =
this->cipher.size();
        ((InternalNode *)this->parent)->insert_node(this->parent_index +
1, new_node);
    }
}
```

```
long long LeafNode::insert(int pos, string cipher)
{
    // 将密文插入到对应的pos
    this->cipher.insert(this->cipher.begin() + pos, cipher);
    this->encoding.insert(this->encoding.begin() + pos, -1);
    long long cd = this->Encode(pos);
    if (this->cipher.size() >= M)
    {
        this->rebalance();
    }
    return cd;
}

long long LeafNode::search(int pos)
{
    return this->encoding.at(pos);
}

void root_initial()
{
    root = new LeafNode();
    ((LeafNode *)root)->lower = 0;
    ((LeafNode *)root)->upper = pow(2, 62);
    update.clear();
    start_update = -1;
    end_update = -1;
};

long long get_update(string cipher)
{
    if (update.count(cipher) > 0)
    {
        return update[cipher];
    }
    return 0;
}
```

- 为能够在MySQL中使用，在UDF.cpp文件中编写udf函数

```
#include "Node.h"
#include "mysql/mysql.h"
#include <string.h>

extern "C"
{
    // 插入
    bool FHInsert_init(UDF_INIT *initid, UDF_ARGS *args, char *message);
    long long FHInsert(UDF_INIT *initid, UDF_ARGS *args, char *is_null,
char *error);
    // 搜索
    bool FHSearch_init(UDF_INIT *const initid, UDF_ARGS *const args, char
*const message);
```

```
    long long FHSearch(UDF_INIT *const initid, UDF_ARGS *const args,
                       char *const result, unsigned long *const length,
                       char *const is_null, char *const error);

    // 更新
    bool FHUpdate_init(UDF_INIT *initid, UDF_ARGS *args, char *message);
    long long FHUpdate(UDF_INIT *initid, UDF_ARGS *args, char *is_null,
char *error);
    // 更新范围
    bool FHStart_init(UDF_INIT *initid, UDF_ARGS *args, char *message);
    long long FHStart(UDF_INIT *initid, UDF_ARGS *args, char *is_null,
char *error);
    bool FHEnd_init(UDF_INIT *initid, UDF_ARGS *args, char *message);
    long long FHEnd(UDF_INIT *initid, UDF_ARGS *args, char *is_null, char
*error);
}

static char * getba(UDF_ARGS *const args, int i, double &len)
{
    len = args->lengths[i];
    return args->args[i];
}

/*插入*/
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;
}

bool FHInsert_init(UDF_INIT *initid, UDF_ARGS *args, char *message)
{
    start_update = -1;
    end_update = -1;
    update.clear();
    if (root == nullptr)
    {
        root_initial();
    }
    return 0;
}

/*搜索*/
long long FHSearch(UDF_INIT *const initid, UDF_ARGS *const args, char
*const result,
                   unsigned long *const length, char *const is_null, char
```

```
*const error)
{
    int pos = *(int *)(args->args[0]);
    if (pos < 0)
        return 0;
    return root->search(pos);
}
bool FHSearch_init(UDF_INIT *const initid, UDF_ARGS *const args, char
*const message)
{
    return 0;
}

long long FHUpdate(UDF_INIT *initid, UDF_ARGS *args, char *is_null, char
*error)
{
    double keyLen;
    char *const keyBytes = getba(args, 0, keyLen);
    const std::string cipher = std::string(keyBytes, keyLen);
    long long update_code = get_update(cipher);
    return update_code;
}

bool FHUpdate_init(UDF_INIT *initid, UDF_ARGS *args, char *message)
{
    return 0;
}

long long FHStart(UDF_INIT *initid, UDF_ARGS *args, char *is_null, char
*error)
{
    return start_update;
}

bool FHStart_init(UDF_INIT *initid, UDF_ARGS *args, char *message)
{
    return 0;
}

long long FHEnd(UDF_INIT *initid, UDF_ARGS *args, char *is_null, char
*error)
{
    return end_update;
}

bool FHEnd_init(UDF_INIT *initid, UDF_ARGS *args, char *message)
{
    return 0;
}
```

3、编译生成动态链接库

- 使用如下命令编译生成动态链接库libzidx.so

```
g++ -shared -fPIC UDF.cpp FH-OPE.cpp -lcrypto -o libope.so
```

- 将其拷贝到MySQL的文件夹

```
sudo cp libope.so /usr/lib/mysql/plugin/
```

4、导入MySQL

- 编写sql文件

```
-- 如果原来有同名表，则删除
drop table if exists example;
-- 创建名为example的表
create table example
(
    encoding   bigint,
    ciphertext varchar(512)
);

-- 删除已有的函数
drop function if exists FHInsert;
drop function if exists FHSearch;
drop function if exists FHUpdate;
drop function if exists FHStart;
drop function if exists FHEnd;

-- 创建函数
create function FHInsert RETURNS INTEGER SONAME 'libfhope.so';
create function FHSearch RETURNS INTEGER SONAME 'libfhope.so';
create function FHUpdate RETURNS INTEGER SONAME 'libfhope.so';
create function FHStart RETURNS INTEGER SONAME 'libfhope.so';
create function FHEnd RETURNS INTEGER SONAME 'libfhope.so';

-- 创建插入数据的存储过程
drop procedure if exists pro_insert;
delimiter $$
create procedure pro_insert(IN pos int, IN ct varchar(512))
BEGIN
    DECLARE i BIGINT default 0;
    SET i = FHInsert(pos, ct);
    insert into example values (i, ct);
    if i = 0 then
        -- 树结构中更新了编码，同步更新数据库中的信息
        update example
        set encoding = FHUpdate(ciphertext)
        where (encoding >= FHStart() and encoding < FHEnd())
```

```
            or (encoding = 0);
    end if;
END $$
delimiter ;
```

- 登陆

```
mysql -u<用户名> -p<密码>
```

例如登陆用户user，密码为123456：

```
ubuntu@laptop:~$ 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 14
Server version: 8.0.30-0ubuntu0.22.04.1 (Ubuntu)

Copyright (c) 2000, 2022, 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>
```

- 切换数据库

```
use <数据库名>
```

切换到test_db数据库：

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

- 导入sql文件

```
source <文件地址>
```

文件地址应为sql文件的绝对地址，例如：

```
mysql> source /home/ubuntu/OPEUDF/load.sql
Query OK, 0 rows affected, 1 warning (0.01 sec)

Query OK, 0 rows affected (0.09 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, 1 warning (0.01 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 (0.02 sec)

Query OK, 0 rows affected (0.01 sec)

Query OK, 0 rows affected (0.01 sec)

Query OK, 0 rows affected (0.02 sec)

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

Query OK, 0 rows affected (0.02 sec)
```

5、编写Client端程序并测试

> 编写Client部分代码，存入client.py

此部分代码已经放入附录中

> 运行

通过python3指令可直接执行py文件

```
python3 client.py
```

```
ubuntu@laptop:~/OPEUDF$ python3 client.py
ciphtertext: cIbvZj5cu/o99NDyZ0KjLzft0fmWX5lL6vLXsgUuLEtaOjRsAvmpPRBexg/LI+a/ plaintext: banana
ciphtertext: v0ZHEERstzm3LuJg+3qGViixrYvUX5PKmwi2y/jrfGI+yxvxH29cEjwi2S26B8Wu plaintext: orange
ciphtertext: OdNCr/SpXjMzQaZ/iFa2V/7Rn6NqWU786IemFM0x0gtytEA4Y09lBIQXMm60Xf9O plaintext: cherry
ciphtertext: yLajfw+f5FrccGS4pHKnNXSFJvn7C8YY03efMfrW/sfySS4yD6YC0a/06FPq7K7S plaintext: cherry
ciphtertext: 99sfLbVSSGQBA3ppHtjIA20XM6YUGFwbbR4YhvyOJnkpSwRvrgW/cF+dGarqRFQ4 plaintext: orange
```

可以看到实验结果与预期相符，打印了密文及解密后的明文

四、心得体会

在本次实验中，首先学习到了频率隐藏保序加密的相关概念，以及mysql的一些使用方法

还了解到udf的概念及简单编写方法并最后将FH-OPE在实验中复现出来

最后通过本次实验对所学到的理论知识进行相应的应用，期待自己未来更好的发展，心想事成、万事胜意、未来可期

五、附录——client.py完整代码

```
import pymysql
import random
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad
import base64

local_table = {}
key = get_random_bytes(16)
base_iv = get_random_bytes(16)

def AES_ENC(plaintext, iv):
    # AES加密
    aes = AES.new(key, AES.MODE_CBC, iv=iv)
    padded_data = pad(plaintext, AES.block_size, style='pkcs7')
    ciphertext = aes.encrypt(padded_data)
    return ciphertext

def AES_DEC(ciphertext, iv):
    # AES解密
    aes = AES.new(key, AES.MODE_CBC, iv=iv)
    padded_data = aes.decrypt(ciphertext)
    plaintext = unpad(padded_data, AES.block_size, style='pkcs7')
    return plaintext

def Random_Encrypt(plaintext):
    # 随机生成iv来保证加密结果的随机性
    iv = get_random_bytes(16)
    ciphertext = AES_ENC(iv + AES_ENC(plaintext.encode('utf-8'), iv),
base_iv)
    ciphertext = base64.b64encode(ciphertext)
    return ciphertext.decode('utf-8')

def Random_Decrypt(ciphertext):
    plaintext = AES_DEC(base64.b64decode(ciphertext.encode('utf-8'))
,base_iv)
    plaintext = AES_DEC(plaintext[16:],plaintext[:16])
    return plaintext.decode('utf-8')

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])
    if plaintext in local_table:
        local_table[plaintext] += 1
        return random.randint(presum, presum + local_table[plaintext] - 1)
    else:
```

```
        local_table[plaintext] = 1
        return presum

def GetLeftPos(plaintext):
    return sum([v for k, v in local_table.items() if k < plaintext])

def GetRightPos(plaintext):
    return sum([v for k, v in local_table.items() if k <= plaintext])

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()
    cur.execute(f"call pro_insert({CalPos(plaintext)},'{ciphertext}')")
    conn.commit()
    conn.close()

def Search(left, right):
    # 搜索[left,right]中的信息
    left_pos = GetLeftPos(left)
    right_pos = GetRightPos(right)
    # 连接数据库
    conn = pymysql.connect(host='localhost', user='user',
                           passwd='123456', database='test_db')
    cur = conn.cursor()
    cur.execute(
        f"select ciphertext 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])}")

if __name__ == '__main__':
    # 插入明文，同时设置了一部分重复的内容
    for ciphertext in ['apple', 'pear', 'banana', 'orange', 'cherry',
'apple', 'cherry', 'orange']:
        Insert(ciphertext)

    # 假设我们搜索b和p之间的数据
    Search('b', 'p')
```