SEAL应用实践

- 姓名：陈睿颖
- 学号：2013544
- 专业：计算机科学与技术

1. 实验内容

参考教材实验2.3，实现将三个数的密文发送到服务器完成 $x^3 + yz$ 的运算

2. 实验环境

本实验使用的是WSL Ubuntu22.04，IDE为VSCode远程连接WSL。

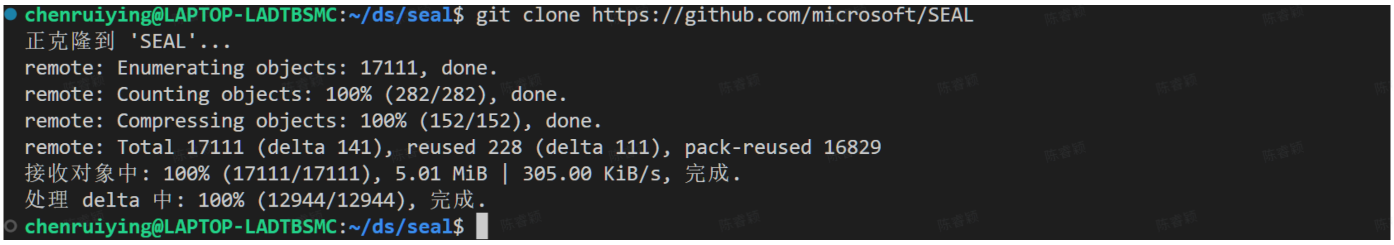
3. 实验步骤

3.1 SEAL库的安装

1. git clone 加密库资源 在 Ubuntu 的 home 文件夹下建立文件夹 seal，进入该文件夹后，打开终端，输入命令：

```
git clone https://github.com/microsoft/SEAL
```

运行完毕，将在 seal 文件夹下自动建立 SEAL 这个新文件夹。



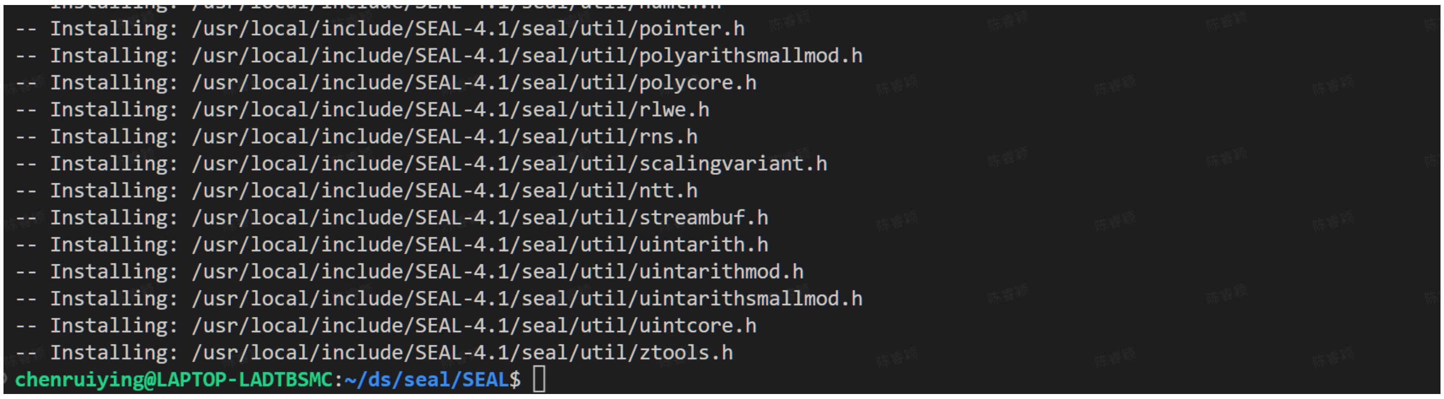
```
chenruiying@LAPTOP-LADTBSMC:~/ds/seal$ git clone https://github.com/microsoft/SEAL
正克隆到 'SEAL'...
remote: Enumerating objects: 17111, done.
remote: Counting objects: 100% (282/282), done.
remote: Compressing objects: 100% (152/152), done.
remote: Total 17111 (delta 141), reused 228 (delta 111), pack-reused 16829
接收对象中: 100% (17111/17111), 5.01 MiB | 305.00 KiB/s, 完成.
处理 delta 中: 100% (12944/12944), 完成.
chenruiying@LAPTOP-LADTBSMC:~/ds/seal$
```

2. 编译和安装

输入命令:

```
cd SEAL
cmake .
make
sudo make install
```

最后显示如下:



```
-- Installing: /usr/local/include/SEAL-4.1/seal/util/pointer.h
-- Installing: /usr/local/include/SEAL-4.1/seal/util/polyarithsmallmod.h
-- Installing: /usr/local/include/SEAL-4.1/seal/util/polycore.h
-- Installing: /usr/local/include/SEAL-4.1/seal/util/rlwe.h
-- Installing: /usr/local/include/SEAL-4.1/seal/util/rns.h
-- Installing: /usr/local/include/SEAL-4.1/seal/util/scalingvariant.h
-- Installing: /usr/local/include/SEAL-4.1/seal/util/ntt.h
-- Installing: /usr/local/include/SEAL-4.1/seal/util/streambuf.h
-- Installing: /usr/local/include/SEAL-4.1/seal/util/uintarith.h
-- Installing: /usr/local/include/SEAL-4.1/seal/util/uintarithmod.h
-- Installing: /usr/local/include/SEAL-4.1/seal/util/uintarithsmallmod.h
-- Installing: /usr/local/include/SEAL-4.1/seal/util/uintcore.h
-- Installing: /usr/local/include/SEAL-4.1/seal/util/ztools.h
chenruiying@LAPTOP-LADTBSMC:~/ds/seal/SEAL$
```

安装完毕!

3.2 基于 CKKS 方案构建一个基于云服务器的算力协助完成客户端的某种运算

3.2.1 标准化构建流程

共有以下几个步骤:

1. 选择 CKKS 参数 parms

2. 生成 CKKS 框架 context
3. 构建 CKKS 模块 keygenerator、encoder、encryptor、evaluator 和 decryptor
4. 使用 encoder 将数据 n 编码为明文 m
5. 使用 encryptor 将明文 m 加密为密文 c
6. 使用 evaluator 对密文 c 运算为密文 c’
7. 使用 decryptor 将密文 c’ 解密为明文 m’
8. 使用 encoder 将明文 m’ 解码为数据 n

3.2.2 示例代码

同态加密算法最直观的应用是云计算，其基本流程为：

1. 发送方利用公钥 pk 加密明文 m 为密文 c
2. 发送方把密文 c 发送到服务器
3. 服务器执行密文运算，生成结果密文 c’
4. 服务器将结果密文 c’ 发送给接收方
5. 接收方利用私钥 sk 解密密文 c’ 为明文结果 m’

当发送方与接收方相同时，则该客户利用全同态加密算法完成了一次安全计算，即既 利用了云计算的算力，又保障了数据的安全性，这对云计算的安全应用有重要意义。

首先，我们要使用example.h，该头文件定义了使用 seal 的常见头文件，并定义了一些输出函数，在 SEAL/native/example目录下可以找到：

```
// Copyright (c) Microsoft Corporation. All rights reserved.
// Licensed under the MIT license.

#pragma once

#include "seal/seal.h"
#include <algorithm>
#include <chrono>
#include <cstddef>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <limits>
#include <memory>
#include <mutex>
#include <numeric>
#include <random>
#include <sstream>
```

```cpp
#include <string>
#include <thread>
#include <vector>

void example_bfv_basics();

void example_encoders();

void example_levels();

void example_bgv_basics();

void example_ckks_basics();

void example_rotation();

void example_serialization();

void example_performance_test();

/*
Helper function: Prints the name of the example in a fancy banner.
*/
inline void print_example_banner(std::string title)
{
    if (!title.empty())
    {
        std::size_t title_length = title.length();
        std::size_t banner_length = title_length + 2 * 10;
        std::string banner_top = "+" + std::string(banner_length - 2, '-') + "+";
        std::string banner_middle = "|" + std::string(9, ' ') + title + std::string(9, ' ') + "|";

        std::cout << std::endl << banner_top << std::endl << banner_middle << std::endl << banner_top << std::endl;
    }
}

/*
Helper function: Prints the parameters in a SEALContext.
*/
inline void print_parameters(const seal::SEALContext &context)
{
    auto &context_data = *context.key_context_data();

    /*
    Which scheme are we using?
    */
    std::string scheme_name;
    switch (context_data.parms().scheme())
    {
    case seal::scheme_type::bfv:
        scheme_name = "BFV";
        break;
    case seal::scheme_type::ckks:
        scheme_name = "CKKS";
        break;
    case seal::scheme_type::bgv:
        scheme_name = "BGV";
        break;
    default:
        throw std::invalid_argument("unsupported scheme");
    }
    std::cout << "/" << std::endl;
    std::cout << "| Encryption parameters :" << std::endl;
    std::cout << "|   scheme: " << scheme_name << std::endl;
    std::cout << "|   poly_modulus_degree: " << context_data.parms().poly_modulus_degree() << std::endl;

    /*
    Print the size of the true (product) coefficient modulus.
    */
    std::cout << "|   coeff_modulus size: ";
    std::cout << context_data.total_coeff_modulus_bit_count() << " (";
    auto coeff_modulus = context_data.parms().coeff_modulus();
    std::size_t coeff_modulus_size = coeff_modulus.size();
    for (std::size_t i = 0; i < coeff_modulus_size - 1; i++)
    {
        std::cout << coeff_modulus[i].bit_count() << " + ";
    }
    std::cout << coeff_modulus.back().bit_count();
    std::cout << ") bits" << std::endl;

    /*
    For the BFV scheme print the plain_modulus parameter.
    */
    if (context_data.parms().scheme() == seal::scheme_type::bfv)
    {
        std::cout << "|   plain_modulus: " << context_data.parms().plain_modulus().value() << std::endl;
    }

    std::cout << "\\" << std::endl;
```

```cpp
}

/*
Helper function: Prints the `parms_id' to std::ostream.
*/
inline std::ostream &operator<<(std::ostream &stream, seal::parms_id_type parms_id)
{
    /*
    Save the formatting information for std::cout.
    */
    std::ios old_fmt(nullptr);
    old_fmt.copyfmt(std::cout);

    stream << std::hex << std::setfill('0') << std::setw(16) << parms_id[0] << " " << std::setw(16) << parms_id[1]
           << " " << std::setw(16) << parms_id[2] << " " << std::setw(16) << parms_id[3] << " ";

    /*
    Restore the old std::cout formatting.
    */
    std::cout.copyfmt(old_fmt);

    return stream;
}

/*
Helper function: Prints a vector of floating-point values.
*/
template <typename T>
inline void print_vector(std::vector<T> vec, std::size_t print_size = 4, int prec = 3)
{
    /*
    Save the formatting information for std::cout.
    */
    std::ios old_fmt(nullptr);
    old_fmt.copyfmt(std::cout);

    std::size_t slot_count = vec.size();

    std::cout << std::fixed << std::setprecision(prec);
    std::cout << std::endl;
    if (slot_count <= 2 * print_size)
    {
        std::cout << "    [";
        for (std::size_t i = 0; i < slot_count; i++)
        {
            std::cout << " " << vec[i] << ((i != slot_count - 1) ? "," : " ]\n");
        }
    }
    else
    {
        vec.resize(std::max(vec.size(), 2 * print_size));
        std::cout << "    [";
        for (std::size_t i = 0; i < print_size; i++)
        {
            std::cout << " " << vec[i] << ",";
        }
        if (vec.size() > 2 * print_size)
        {
            std::cout << " ...,";
        }
        for (std::size_t i = slot_count - print_size; i < slot_count; i++)
        {
            std::cout << " " << vec[i] << ((i != slot_count - 1) ? "," : " ]\n");
        }
    }
    std::cout << std::endl;

    /*
    Restore the old std::cout formatting.
    */
    std::cout.copyfmt(old_fmt);
}

/*
Helper function: Prints a matrix of values.
*/
template <typename T>
inline void print_matrix(std::vector<T> matrix, std::size_t row_size)
{
    /*
    We're not going to print every column of the matrix (there are 2048). Instead
    print this many slots from beginning and end of the matrix.
    */
    std::size_t print_size = 5;

    std::cout << std::endl;
    std::cout << "    [";
    for (std::size_t i = 0; i < print_size; i++)
    {
        std::cout << std::setw(3) << std::right << matrix[i] << ",";
    }
    std::cout << std::setw(3) << " ...,";
    for (std::size_t i = row_size - print_size; i < row_size; i++)
    {
        std::cout << std::setw(3) << matrix[i] << ((i != row_size - 1) ? "," : " ]\n");
    }
    std::cout << "    [";
    for (std::size_t i = row_size; i < row_size + print_size; i++)
    {
        std::cout << std::setw(3) << matrix[i] << ",";
    }
    std::cout << std::setw(3) << " ...,";
    for (std::size_t i = 2 * row_size - print_size; i < 2 * row_size; i++)
    {
        std::cout << std::setw(3) << matrix[i] << ((i != 2 * row_size - 1) ? "," : " ]\n");
    }
    std::cout << std::endl;
}

/*
Helper function: Print line number.
*/
inline void print_line(int line_number)
{
    std::cout << "Line " << std::setw(3) << line_number << " --> ";
}

/*
Helper function: Convert a value into a hexadecimal string, e.g., uint64_t(17) --> "11".
*/
inline std::string uint64_to_hex_string(std::uint64_t value)
{
    return seal::util::uint_to_hex_string(&value, std::size_t(1));
}

```

使用的代码如下：

```
#include "examples.h"
```

```cpp
/*该文件可以在SEAL/native/example目录下找到*/
#include <vector>
using namespace std;
using namespace seal;
#define N 3
//本例目的：给定x, y, z三个数的密文，让服务器计算x*y*z

int main(){
//初始化要计算的原始数据
vector<double> x, y, z;
    x = { 1.0, 2.0, 3.0 };
    y = { 2.0, 3.0, 4.0 };
    z = { 3.0, 4.0, 5.0 };

/**********************************
客户端的视角：生成参数、构建环境和生成密文
***********************************/
//（1）构建参数容器 parms
EncryptionParameters parms(scheme_type::ckks);
/*CKKS有三个重要参数：
1.poly_module_degree(多项式模数)
2.coeff_modulus（参数模数）
3.scale（规模）*/

size_t poly_modulus_degree = 8192;
parms.set_poly_modulus_degree(poly_modulus_degree);
parms.set_coeff_modulus(CoeffModulus::Create(poly_modulus_degree, { 60, 40,
40, 60 }));
//选用2^40进行编码
double scale = pow(2.0, 40);

//（2）用参数生成CKKS框架context
SEALContext context(parms);

//（3）构建各模块
//首先构建keygenerator，生成公钥、私钥
KeyGenerator keygen(context);
auto secret_key = keygen.secret_key();
PublicKey public_key;
    keygen.create_public_key(public_key);

//构建编码器，加密模块、运算器和解密模块
//注意加密需要公钥pk；解密需要私钥sk；编码器需要scale
    Encryptor encryptor(context, public_key);
    Decryptor decryptor(context, secret_key);

    CKKSEncoder encoder(context);
```

```
//对向量x、y、z进行编码
    Plaintext xp, yp, zp;
    encoder.encode(x, scale, xp);
    encoder.encode(y, scale, yp);
    encoder.encode(z, scale, zp);
//对明文xp、yp、zp进行加密
    Ciphertext xc, yc, zc;
    encryptor.encrypt(xp, xc);
    encryptor.encrypt(yp, yc);
    encryptor.encrypt(zp, zc);


//至此，客户端将pk、CKKS参数发送给服务器，服务器开始运算
/***********************************
服务器的视角：生成重线性密钥、构建环境和执行密文计算
************************************/
//生成重线性密钥和构建环境
SEALContext context_server(parms);
    RelinKeys relin_keys;
    keygen.create_relin_keys(relin_keys);
    Evaluator evaluator(context_server);

/*对密文进行计算，要说明的原则是：
-加法可以连续运算，但乘法不能连续运算
-密文乘法后要进行relinearize操作
-执行乘法后要进行rescaling操作
-进行运算的密文必需执行过相同次数的rescaling（位于相同level）*/
    Ciphertext temp;
    Ciphertext result_c;
//计算x*y，密文相乘，要进行relinearize和rescaling操作
    evaluator.multiply(xc,yc,temp);
    evaluator.relinearize_inplace(temp, relin_keys);
    evaluator.rescale_to_next_inplace(temp);

//在计算x*y * z之前，z没有进行过rescaling操作，所以需要对z进行一次乘法和rescaling操作，目的是使得x*y 和z在相同的层
    Plaintext wt;
    encoder.encode(1.0, scale, wt);
//此时，我们可以查看框架中不同数据的层级：
cout << "    + Modulus chain index for zc: "
<< context_server.get_context_data(zc.parms_id())->chain_index() << endl;
cout << "    + Modulus chain index for temp(x*y): "
<< context_server.get_context_data(temp.parms_id())->chain_index() << endl;
cout << "    + Modulus chain index for wt: "
<< context_server.get_context_data(wt.parms_id())->chain_index() << endl;

//执行乘法和rescaling操作：
```

```
    evaluator.multiply_plain_inplace(zc, wt);
    evaluator.rescale_to_next_inplace(zc);

//再次查看zc的层级，可以发现zc与temp层级变得相同
cout << "    + Modulus chain index for zc after zc*wt and rescaling: "
<< context_server.get_context_data(zc.parms_id())->chain_index() << endl;

//最后执行temp（x*y）* zc（z*1.0）
    evaluator.multiply_inplace(temp, zc);
    evaluator.relinearize_inplace(temp,relin_keys);
    evaluator.rescale_to_next(temp, result_c);

//计算完毕，服务器把结果发回客户端
/**********************************
客户端的视角：进行解密和解码
***********************************/
//客户端进行解密
    Plaintext result_p;
    decryptor.decrypt(result_c, result_p);
//注意要解码到一个向量上
    vector<double> result;
    encoder.decode(result_p, result);
//得到结果，正确的话将输出：{6.000, 24.000, 60.000, ..., 0.000, 0.000, 0.000}
    cout << "结果是：" << endl;
    print_vector(result,3,3);
return 0;
}

```

每次进行运算前，要保证参与运算的数据位于同一“level”上。 加法不需要进行 rescaling 操作，因此不会改变数据的 level。数据的 level 只能降低无法升 高，所以要小心设计计算的先后顺序。 可以通过输出 `p.scale()`、`p.parms_id()` 以及 `context->get_context_data( p.parms_id() ) ->chain_index()` 来确认即将进行操作的数据满足如下计算条件：

- 用同一组参数进行加密；
- 位于（chain）上的同一 level；
- scale 相同。

要想把不同 level 的数据拉到同一 level，可以利用乘法单位元 1 把层数较高的操作数 拉到较低的 level（如本例），也可以通过内置函数进行直接转换。

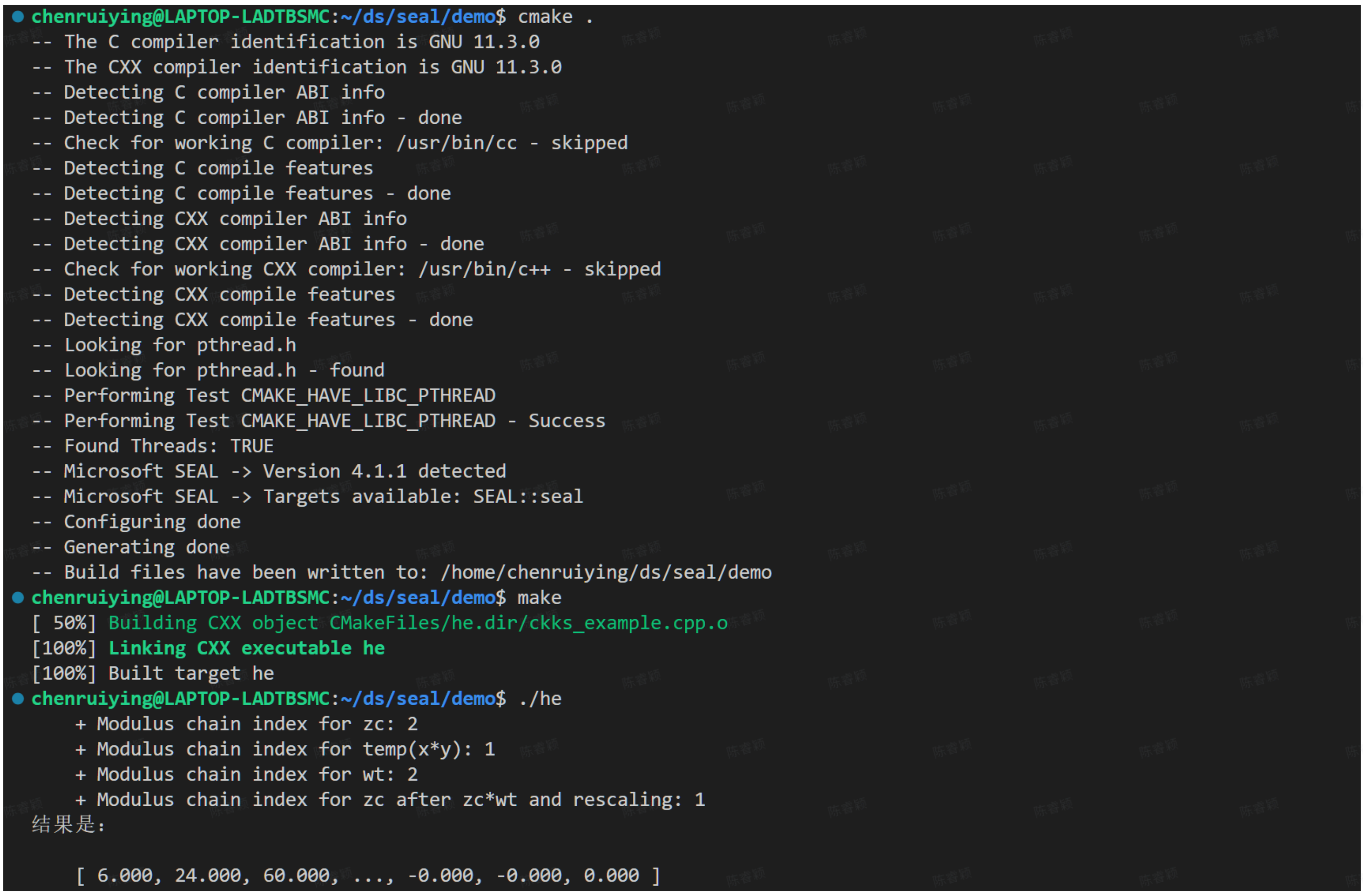
使用的CMakeList.txt如下：

```
cmake_minimum_required(VERSION 3.10)
project(demo)
```

```
add_executable(he ckks_example.cpp)
add_compile_options(-std=c++17)

find_package(SEAL)
target_link_libraries(he SEAL::seal)
```

编译运行结果如下



```
chenruiying@LAPTOP-LADTBSMC:~/ds/seal/demo$ cmake .
-- The C compiler identification is GNU 11.3.0
-- The CXX compiler identification is GNU 11.3.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Looking for pthread.h
-- Looking for pthread.h - found
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD - Success
-- Found Threads: TRUE
-- Microsoft SEAL -> Version 4.1.1 detected
-- Microsoft SEAL -> Targets available: SEAL::seal
-- Configuring done
-- Generating done
-- Build files have been written to: /home/chenruiying/ds/seal/demo
chenruiying@LAPTOP-LADTBSMC:~/ds/seal/demo$ make
[ 50%] Building CXX object CMakeFiles/he.dir/ckks_example.cpp.o
[100%] Linking CXX executable he
[100%] Built target he
chenruiying@LAPTOP-LADTBSMC:~/ds/seal/demo$ ./he
    + Modulus chain index for zc: 2
    + Modulus chain index for temp(x*y): 1
    + Modulus chain index for wt: 2
    + Modulus chain index for zc after zc*wt and rescaling: 1
结果是:

    [ 6.000, 24.000, 60.000, ..., -0.000, -0.000, 0.000 ]
```

运行结果正确!

4. 心得体会

- 学习了SEAL库在Ubuntu环境下的安装
- 学习了基于 CKKS 方案构建一个基于云服务器的算力协助完成客户端的某种运算的标准化构建流程
- 学习到了利用C++语言实现将三个数的密文发送到服务器完成 x^3+yz 的运算