

南开大学

数据安全课程实验报告

实验三：SEAL 应用实践



学 院_____网络空间安全学院_____
专 业_____信息安全_____
学 号_____2112060_____
姓 名_____孙露_____

一、实验要求

参考教材实验 2.3，实现将 3 个数的密文发送到服务器完成 x^3+y*z 的运算

二、实验原理

1. SEAL(Simple Encrypted Arithmetic Library)是微软开源的基于 C++的同态加密库，支持 CKKS 方案等多种全同态加密方案，支持基于整数的精确同态运算和基于浮点数的近似同态运算。该项目采用商业友好的 MIT 许可证在 GitHub 上 (<https://github.com/microsoft/SEAL>) 开源。SEAL 基于 C++实现，不需要其他依赖库。

2. cmake 是一种高级编译配置工具，它可以将多个 cpp、hpp 文件组合构建为一个工程的语言。它能够输出各种各样的 makefile 或者 project 文件，所有操作都是通过编译 CMakeLists.txt 来完成。

3. SEAL 中的层级

当从给定的 EncryptionParameters 实例创建 SEALContext 时，微软 SEAL 会自动创建一个所谓的“模数切换链”，它是从原始集合衍生出来的其他加密参数链。模量交换链中的参数除系数模量沿链向下逐渐减小外，其余参数均与原参数相同。更准确地说，链中的每个参数集都试图从前一个集合中移除最后一个系数模素数；这会一直持续到参数集不再有效为止。（例如：plain_modulus 大于剩余的 coeff_modulus）。很容易遍历整个链并访问所有参数集。此外，链中设置的每个参数都有一个“链索引”，指示其在链中的位置，因此最后一组参数的索引为 0。一组加密参数，或者一个携带这些加密参数的对象，在链中的级别高于另一组参数，如果它的链索引较大，也就是说，它在链中较早。

SEAL 根据默认的参数创建了一个 modulus switching chain，在同一个链上的加密实例除了 coefficient modulus 其他都相同。

“Modulus switching”是一种降低密文参数的技术在链中 Evaluator::mod_switch_to_next 总是切换到下一级链，而 Evaluator::mod_switch_to 切换到在对应于给定 parms_id 的链上设置的参数。但他是不能在链条向上切换的。

三、实验过程

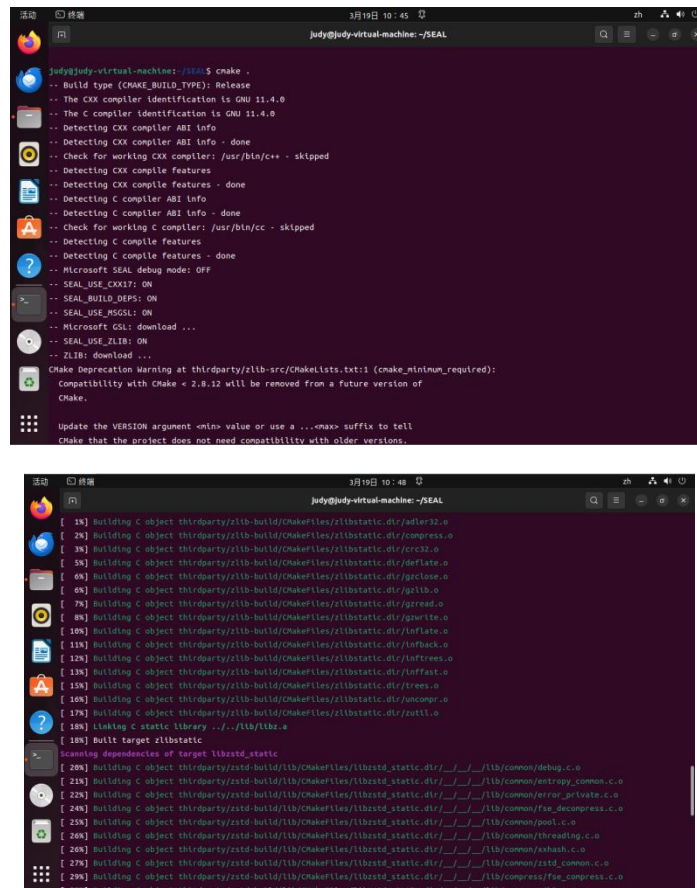
(一) 配置环境

从 github 上拉取开源的 SEAL 库

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

```
judy@judy-virtual-machine:~$ git clone https://github.com/microsoft/SEAL
正克隆到 'SEAL'...
remote: Enumerating objects: 17111, done.
remote: Counting objects: 100% (17111/17111), done.
remote: Compressing objects: 100% (3938/3938), done.
remote: Total 17111 (delta 13022), reused 16910 (delta 12959), pack-reused 0
接收对象中: 100% (17111/17111), 4.92 MiB | 2.96 MiB/s, 完成。
处理 delta 中: 100% (13022/13022), 完成。
```

进入 SEAL 文件夹，输入 cmake . 和 make



```
judy@judy-virtual-machine:~/SEAL$ cmake .
-- Build type (CMAKE_BUILD_TYPE): Release
-- The CXX compiler identification is GNU 11.4.0
-- The C compiler identification is GNU 11.4.0
-- 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
-- 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
-- Microsoft SEAL debug mode: OFF
-- SEAL_USE_CXX17: ON
-- SEAL_BUILD_DEPS: ON
-- SEAL_USE_MSGL: ON
-- Microsoft GSL: download ...
-- SEAL_USE_ZLIB: ON
-- ZLIB: download ...
CMake Deprecation Warning at thirdparty/zlib-src/CMakeLists.txt:1 (cmake_minimum_required):
Compatibility with CMake < 2.8.12 will be removed from a future version of
CMake.

Update the VERSION argument <min> value or use a ...>max> suffix to tell
CMake that the project does not need compatibility with older versions.

judy@judy-virtual-machine:~/SEAL$ make
[ 1%] Building C object thirdparty/zlib-build/CMakeFiles/zlibstatic.dir/adler32.o
[ 2%] Building C object thirdparty/zlib-build/CMakeFiles/zlibstatic.dir/compress.o
[ 3%] Building C object thirdparty/zlib-build/CMakeFiles/zlibstatic.dir/crc32.o
[ 5%] Building C object thirdparty/zlib-build/CMakeFiles/zlibstatic.dir/deflate.o
[ 6%] Building C object thirdparty/zlib-build/CMakeFiles/zlibstatic.dir/gzclose.o
[ 8%] Building C object thirdparty/zlib-build/CMakeFiles/zlibstatic.dir/gzlib.o
[ 7%] Building C object thirdparty/zlib-build/CMakeFiles/zlibstatic.dir/gzread.o
[ 8%] Building C object thirdparty/zlib-build/CMakeFiles/zlibstatic.dir/gzwrite.o
[ 10%] Building C object thirdparty/zlib-build/CMakeFiles/zlibstatic.dir/inflate.o
[ 11%] Building C object thirdparty/zlib-build/CMakeFiles/zlibstatic.dir/inflate.o
[ 12%] Building C object thirdparty/zlib-build/CMakeFiles/zlibstatic.dir/inftrees.o
[ 13%] Building C object thirdparty/zlib-build/CMakeFiles/zlibstatic.dir/inftree.o
[ 15%] Building C object thirdparty/zlib-build/CMakeFiles/zlibstatic.dir/trees.o
[ 16%] Building C object thirdparty/zlib-build/CMakeFiles/zlibstatic.dir/uncompr.o
[ 17%] Building C object thirdparty/zlib-build/CMakeFiles/zlibstatic.dir/zutil.o
[ 18%] Linking C static library ../lib/zlib.a
[ 18%] Built target zlibstatic
Scanning dependencies of target libzstd_static
[ 20%] Building C object thirdparty/zstd-build/lib/CMakeFiles/libzstd_static.dir/__/__/lib/common/debug.c.o
[ 21%] Building C object thirdparty/zstd-build/lib/CMakeFiles/libzstd_static.dir/__/__/lib/common/entropy_common.c.o
[ 22%] Building C object thirdparty/zstd-build/lib/CMakeFiles/libzstd_static.dir/__/__/lib/common/error_private.c.o
[ 24%] Building C object thirdparty/zstd-build/lib/CMakeFiles/libzstd_static.dir/__/__/lib/common/fse_decompress.c.o
[ 25%] Building C object thirdparty/zstd-build/lib/CMakeFiles/libzstd_static.dir/__/__/lib/common/pool.c.o
[ 26%] Building C object thirdparty/zstd-build/lib/CMakeFiles/libzstd_static.dir/__/__/lib/common/threading.c.o
[ 26%] Building C object thirdparty/zstd-build/lib/CMakeFiles/libzstd_static.dir/__/__/lib/common/txt_common.c.o
[ 27%] Building C object thirdparty/zstd-build/lib/CMakeFiles/libzstd_static.dir/__/__/lib/common/zstd_common.c.o
[ 28%] Building C object thirdparty/zstd-build/lib/CMakeFiles/libzstd_static.dir/__/__/lib/compress/fse_compress.c.o
[ 30%] Building C object thirdparty/zstd-build/lib/CMakeFiles/libzstd_static.dir/__/__/lib/compress/hist.c.o
```

输入

```
sudo make install
```

```
Judy@Judy-virtual-machine: ~/SEAL
[100%] Linking CXX static library lib/libseal-4.1.a
[100%] Built target seal
Judy@Judy-virtual-machine: ~/SEAL$ sudo make install
[sudo] judy 的密码:
Consolidate compiler generated dependencies of target zlibstatic
[ 18%] Built target zlibstatic
Consolidate compiler generated dependencies of target libzstd_static
[ 54%] Built target libzstd_static
Consolidate compiler generated dependencies of target seal
[100%] Built target seal
Install the project...
-- Install configuration: "Release"
-- Installing: /usr/local/include/SEAL-4.1/seal/util/config.h
-- Installing: /usr/local/lib/libseal-4.1.a
-- Installing: /usr/local/lib/cmake/SEAL-4.1/SEALTargets.cmake
-- Installing: /usr/local/lib/cmake/SEAL-4.1/SEALTargets-release.cmake
-- Installing: /usr/local/lib/cmake/SEAL-4.1/SEALConfig.cmake
-- Installing: /usr/local/lib/cmake/SEAL-4.1/SEALConfigVersion.cmake
-- Installing: /usr/local/include/SEAL-4.1/gsl
-- Installing: /usr/local/include/SEAL-4.1/gsl/span
-- Installing: /usr/local/include/SEAL-4.1/gsl/pointers
-- Installing: /usr/local/include/SEAL-4.1/gsl/gsl_byte
-- Installing: /usr/local/include/SEAL-4.1/gsl/gsl_narrow
-- Installing: /usr/local/include/SEAL-4.1/gsl/span_ext
-- Installing: /usr/local/include/SEAL-4.1/gsl/gsl_util
-- Installing: /usr/local/include/SEAL-4.1/gsl/algorithm
-- Installing: /usr/local/include/SEAL-4.1/gsl/narrow
-- Installing: /usr/local/include/SEAL-4.1/gsl/gsl
```

(二) 代码实现

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：1) 用同一组参数进行加密；2) 位于 (chain) 上的同一 level；3) scale 相同。

1. 仿照样例，进行参数和初始化的设置。

```
1 #include "examples.h"
2 /*该文件可以在SEAL/native/example目录下找到*/
3 #include <vector>
4 using namespace std;
5 using namespace seal;
6 #define N 3
7
8 int main() {
9     //初始化要计算的原始数据
10    vector<double> x, y, z;
11    x = { 1.0, 2.0, 3.0 };
12    y = { 2.0, 3.0, 4.0 };
13    z = { 3.0, 4.0, 5.0 };
14    cout << "原始向量x是: " << endl;
15    print_vector(x);
16    cout << endl;
17
18    cout << "原始向量y是: " << endl;
19    print_vector(y);
20    cout << endl;
21
22    cout << "原始向量z是: " << endl;
23    print_vector(z);
24    cout << endl;
25 }
```

```

test.cpp  examples.h  Lab03.cpp  X
杂项文件  (全局范围)
28  客户端的视角：生成参数、构建环境和生成密文
29  *****
30  // (1) 构建参数容器 params
31  EncryptionParameters params(scheme_type::ckks);
32  /*CKKS有三个重要参数：
33  1.poly_module_degree(多项式模数)
34  2.coeff_modulus(参数模数)
35  3.scale(规模)*/
36
37  size_t poly_module_degree = 8192;
38  params.set_poly_module_degree(poly_module_degree);
39  params.set_coeff_modulus(CoeffModulus::Create(poly_module_degree, { 60, 40, 40, 60 }));
40  //选用2^40进行编码
41  double scale = pow(2.0, 40);
42
43  // (2) 用参数生成CKKS框架context
44  SEALContext context(params);
45
46  // (3) 构建各模块
47  //首先构建Keygenerator, 生成公钥、私钥
48  KeyGenerator keygen(context);
49  auto secret_key = keygen.secret_key();
50  PublicKey public_key;
51  keygen.create_public_key(public_key);
52
53  //构建编码器, 加密模块、运算器和解密模块
54  //注意加密需要公钥pk, 解密需要私钥sk, 编码器需要scale
55  Encryptor encryptor(context, public_key);
56  Decryptor decryptor(context, secret_key);
57  CKKSEncoder encoder(context);
58
59  //对向量x、y、z进行编码
60  Plaintext xp, yp, zp;
61  encoder.encode(x, scale, xp);
62  encoder.encode(y, scale, yp);
63  encoder.encode(z, scale, zp);
64
65  //对明文xp、yp、zp进行加密
66  Ciphertext xc, yc, zc;
67  encryptor.encrypt(xp, xc);
68  encryptor.encrypt(yp, yc);
69  encryptor.encrypt(zp, zc);

```

前半部分并未进行改动，与样例保持一致。

CKKS 算法由五个模块组成：密钥生成器 keygenerator、加密模块 encryptor、解密模块 decryptor、密文计算模块 evaluator 和编码器 encoder, 其中编码器实现数据和环上元素的相互转换。依据这五个模块，构建同态加密应用的过程为：

- ①选择 CKKS 参数 params
- 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

参数容器 params 有三个参数：poly_module_degree(多项式参数)、coeff_modulus(参数模数)、scale(规模)。

`poly_modulus_degree`(polynomial modulus)的参数必须是 2 的幂，如 1024, 2048, 4096, 8192, 16384, 32768。更大的参数会增加密文的尺寸，这会让计算变慢，但支持的计算也更多。在这里与样例保持一致选择 8192 作为参数。

```
/*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);
```

`[ciphertext]`coefficient modulus coeff_modules` 的个数决定了能进行 rescaling 的次数，进而决定了能执行的乘法操作的次数。`coeff_modules` 的最大位数与 `poly_modules` 有直接的关系。在 `poly_modulus_degrees` 为 8192 时，根据关系，`max coeff_modules bit length` 应该为 218。

`coeff_modulus` 与样例保持一致，选择 {60, 40, 40, 60}。将 60 位质数作为 `coeff_modulus` 中的第一个质数，在解密时可以获得最高精度；选择另一个 60 位质数作为 `coeff_modulus` 的最后一个元素，因为它将用作特殊质数，并且应该与其他质数中最大的那个一样大；中间质数彼此接近。

{60, 40, 40, 60} 有以下含义：①`coeff_modules` 总位长 200 (60+40+40+60) 位，在 `poly_modulus_degrees` 上限之下；②最多进行两次(两层)乘法操作。该系列数字的选择不是随意的，总位长不能超过上表限制；最后一个参数为特殊模数，其值应该与中间模数的最大值相等；中间模数与 `scale` 尽量相近。如果将模数变大，则可以支持更多层级的乘法运算，比如 `poly_modulus` 为 16384 则可以支持 `coeff_modules`={60, 40, 40, 40, 40, 40, 40, 40, 60}，也就是 6 层的运算。

encoder 利用 `scale` 参数对浮点数进行缩放，每次相乘后密文的 `scale` 都会翻倍，因此需要执行 rescaling 操作约减一部分，约模的大素数位长由 `coeff_modules` 中的参数决定。与样例保持一致，选择了 `scale=240`。


```

// (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);

```

用参数生成 CKKS 框架 context，构建 keygenerator 模块，分别使用 keygen.secret_key() 和 keygen.create_public_key 生成公钥和私钥。构建编码器加密模块运算器和解密模块，需要注意的是加密需要公钥 pk，解密需要私钥 sk，编码器需要 scale。

然后对三个向量进行编码和加密

```

// 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。生成重线性密钥与构建环境与样例代码保持一致。

```

// 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);

```

2. 执行密文计算

(1) 计算 x^2 , 即 $x*x$, 将结果保存在 `temp_x2` 中。

计算 $x*x$, 密文相乘, 要进行 `relinearize` 和 `rescaling` 操作

```
// 计算x^2
Ciphertext temp_x2;
cout << "下面计算x^2" << endl;
evaluator.multiply(xc, xc, temp_x2);
evaluator.relinearize_inplace(temp_x2, relin_keys);
evaluator.rescale_to_next_inplace(temp_x2);
// 查看x^2的层级
cout << "    + Modulus chain index for x^2: "
    << context_server.get_context_data(temp_x2.parms_id())->chain_index() << endl;
```

计算完成后查看 x^2 的层级

```
下面计算x^2
    + Modulus chain index for x^2: 1
```

(2) 准备计算 $x*x^2$, 先查看 `xc` 的层级

```
// 查看1.0*x的层级
cout << "下面查看xc的层级: " << endl;
cout << "    + Modulus chain index for xc: "
    << context_server.get_context_data(xc.parms_id())->chain_index() << endl;
cout << endl;
```

查看 `xc` 的层级

```
下面查看xc的层级:
    + Modulus chain index for xc: 2
```

可以看到, `xc` 与 x^2 层级是不一样的, 下面解决两者层级不一致的问题。

(3) 解决 `xc` 与 x^2 层级不一致的问题

```
cout << "下面解决xc与x^2层级不一致的问题" << endl;
cout << endl;

Plaintext plain_x1st;
encoder.encode(1.0, scale, plain_x1st);
// 执行乘法和rescaling操作
// evaluator.multiply_plain_inplace(xc, 1.0, plain_x1st);
evaluator.multiply_plain_inplace(xc, plain_x1st);
evaluator.rescale_to_next_inplace(xc);
// 再次查看xc的层级, 应该与x^2层级相同
cout << "    + Modulus chain index for xc after xc*plain_x1st and rescaling: "
    << context_server.get_context_data(xc.parms_id())->chain_index() << endl;
cout << endl;
```

通过将 `xc` 与乘法单位元 1 相乘, 将 `xc` 的层级从 2 变成 1, 执行乘法和 `rescaling` 操作。

下面解决xc与x^2层级不一致的问题

```
+ Modulus chain index for xc after xc*plain_x1st and rescaling: 1
```

(4) 将前面两步操作的结果相乘, 计算 $xc \cdot x^2$, 得到 x^3 , 存在 temp_x3 中。此步需要进行 relinearize 和 rescaling 操作。

计算完成后查看 x^3 的层级。

```
// 层级相同后, 可计算x^3, 即1.0*x*x^2
Ciphertext temp_x3;
cout << "下面计算x^3" << endl;
cout << endl;
evaluator.multiply_inplace(temp_x2, xc);
evaluator.relinearize_inplace(temp_x2, relin_keys);
evaluator.rescale_to_next(temp_x2, temp_x3);
// evaluator.multiply_inplace(temp_x2, xc);
// evaluator.relinearize_inplace(temp_x2, relin_keys);
// evaluator.rescale_to_next(temp_x2, temp_x3);
// 查看x^3的层级
cout << "    + Modulus chain index for x^3: "
    << context_server.get_context_data(temp_x3.parms_id())->chain_index() << endl;
cout << endl;
```

下面计算x^3

```
+ Modulus chain index for x^3: 0
```

(5) 计算 $y \cdot z$, 结果存在 temp_yz 中。与前面的步骤相同, 同样需要进行 relinearize 和 rescaling 操作。

计算完成后查看 $y \cdot z$ 的层级

```
// 计算y*z
Ciphertext temp_yz;
evaluator.multiply(yz, zc, temp_yz);
evaluator.relinearize_inplace(temp_yz, relin_keys);
evaluator.rescale_to_next_inplace(temp_yz);
// 查看y*z的层级
cout << "    + Modulus chain index for y*z: "
    << context_server.get_context_data(temp_yz.parms_id())->chain_index() << endl;
cout << endl;
```

下面计算y*z

```
+ Modulus chain index for y*z: 1
```

发现 $y \cdot z$ 的层级和 scale (执行 rescaling 次数不同) 都与 x^3 不一致, 需要统一 scale 大小后再进行层级统一。

(6) 统一 $y \cdot z$ 与 x^3 的 scale

P2 和 P1 非常接近于 2^{40} ，可以简单地“欺骗” SEAL 并将比例尺设置为相同。将 $1.0 * x * x^2$ 的比例尺更改为 2^{40} 仅意味着通过 $2^{120}/(P2^2 * P1)$ 缩放 $1.0 * x * x^2$ 的值，非常接近于 1。

```
// 但此时的scales不统一
cout << " Normalize scales to 2^40." << endl;
temp_x3.scale() = pow(2.0, 40);
temp_yz.scale() = pow(2.0, 40);
// 此时的scale的大小已经统一了
cout << " + Exact scale in 1*x^3: " << temp_x3.scale() << endl;
cout << " + Exact scale in y*z: " << temp_yz.scale() << endl;
```

```
Normalize scales to 2^40.
+ Exact scale in 1*x^3: 1.09951e+12
+ Exact scale in y*z: 1.09951e+12
```

(7) 统一 $y*z$ 与 x^3 层级，Evaluator::mod_switch_to_inplace 方法会将明文/密文的模切换到下一位，结果保存在原来的密文里，沿着 chain 向下减一个 level。

```
// 解决x^3与y*z层级不一致的问题
cout << "下面解决x^3与y*z层级不一致的问题" << endl;
cout << endl;
parms_id_type last_parms_id = temp_x3.parms_id();
evaluator.mod_switch_to_inplace(temp_yz, last_parms_id);
cout << " + Modulus chain index for y*z after mod_switch_to_inplace: "
    << context_server.get_context_data(temp_yz.parms_id())->chain_index() << endl;
cout << endl;
```

```
下面解决x^3与y*z层级不一致的问题

+ Modulus chain index for y*z after mod_switch_to_inplace: 0
```

(8) 计算 x^3+y*z ，使用 evaluator.add 实现加法运算，结果保存在 e_result 中

```
// 计算x^3+y*z
cout << "计算x^3+y*z" << endl;
Ciphertext e_result;
evaluator.add(temp_x3, temp_yz, e_result);
```

(9) 客户端进行解密和解码

```

//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(e_result, 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);
return 0;

```

计算完毕，服务器把结果发回客户端，客户端进行解密和解码，注意解码到一个向量上。

```

计算 $x^3+y*z$ 
结果是：

[ 7.000, 20.000, 47.000, ..., 0.000, 0.000, -0.000 ]

```

3. 仿照样例修改 CmakeList.txt

```

1 cmake_minimum_required(VERSION 3.10)
2 project(demo)
3 add_executable(he Lab03.cpp)
4 add_compile_options(-std=c++17)
5
6 find_package(SEAL)
7 target_link_libraries(he SEAL::seal)
8

```

4. 运行如下命令完成本次实验

```

cmake .
make
./he

```

```
judy@judy-virtual-machine: /mnt/hgfs/ubuntu-share/Data-S...
judy@judy-virtual-machine:/mnt/hgfs/ubuntu-share/Data-Security/My_Lab03$ cmake .
-- Microsoft SEAL -> Version 4.1.1 detected
-- Microsoft SEAL -> Targets available: SEAL::seal
-- Configuring done
-- Generating done
-- Build files have been written to: /mnt/hgfs/ubuntu-share/Data-Security/My_Lab03
judy@judy-virtual-machine:/mnt/hgfs/ubuntu-share/Data-Security/My_Lab03$ make
Consolidate compiler generated dependencies of target he
[100%] Built target he
judy@judy-virtual-machine:/mnt/hgfs/ubuntu-share/Data-Security/My_Lab03$ ./he
原始向量x是:

[ 1.000, 2.000, 3.000 ]

原始向量y是:

[ 2.000, 3.000, 4.000 ]

原始向量z是:

[ 1.000, 2.000, 3.000 ]
```

计算结果如下:

```
计算 $x^3+y*z$ 
结果是:

[ 7.000, 20.000, 47.000, ..., -0.000, 0.000, -0.000 ]
```

四、实验结论及心得体会

本次实验练习了 SEAL 库的使用, 完成了 x^3+y*z 的运算。为完成本次实验, 研究了 example 的代码和注释及官方文档。为保持层级一致, 在运算顺序的设计上花费了一定的时间, 在最后的加法运算时, 两者的 scale 实际上是不一致的, 选择直接设置两者的 scale 相同, 然后再进行加法运算。

本次实验不仅锻炼了代码能力, 还加深了对理论课及相关密码学知识的理解, 加强了对先前课程的知识融合。