数据安全 -- SEAL应用实践

学号：2212452
姓名：孟启轩
专业：计算机科学与技术

一、实验要求

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 $x^3 + y \times z$ 的运算。

二、实验原理

CKKS 是一个基于多项式环的全同态加密方案，支持同态加法和同态乘法。在进行同态乘法后密文的大小会扩增一倍，因此每次惩罚操作后 CKKS 都需要进行再线性化（relinearization）和再缩放（rescaling）操作

SEAL 是由微软开发的用于加密计算的 C++ 库，支持 CKKS 等同态方案，提供了相应函数实现 CKKS 等算法

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 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

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

三、实验过程

1、实验环境搭建

在终端内输入命令`git clone https://github.com/microsoft/SEAL`克隆加密库资源。

```
mqx@LAPTOP-2BCGD4JI:/mnt/f/study/datasecurity/lab3$  git clone https://github.com/microsoft/SEAL
Cloning into 'SEAL'...
remote: Enumerating objects: 17186, done.
remote: Counting objects: 100% (3041/3041), done.
remote: Compressing objects: 100% (220/220), done.
remote: Total 17186 (delta 2898), reused 2821 (delta 2821), pack-reused 14145 (from 2)
Receiving objects: 100% (17186/17186), 4.97 MiB | 2.71 MiB/s, done.
Resolving deltas: 100% (13068/13068), done.
Updating files: 100% (360/360), done.
mqx@LAPTOP-2BCGD4JI:/mnt/f/study/datasecurity/lab3$ cd SEAL
```

然后新建 SEAL 文件夹，依次执行：

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

```
-- Configuring done
-- Generating done
-- Build files have been written to: /mnt/f/study/datasecurity/lab3/SEAL
mqx@LAPTOP-2BCGD4JI:/mnt/f/study/datasecurity/lab3/SEAL$
```

```
[ 94%] Building CXX object CMakeFiles/seal.dir/native/src/seal/util/uintarithmod.cpp.o
[ 96%] Building CXX object CMakeFiles/seal.dir/native/src/seal/util/uintarithsmallmod.cpp.o
[ 97%] Building CXX object CMakeFiles/seal.dir/native/src/seal/util/uintcore.cpp.o
[ 98%] Building CXX object CMakeFiles/seal.dir/native/src/seal/util/ztools.cpp.o
[100%] Linking CXX static library lib/libseal-4.1.a
[100%] Built target seal
mqx@LAPTOP-2BCGD4JI:/mnt/f/study/datasecurity/lab3/SEAL$
```

```
-- 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
mqx@LAPTOP-2BCGD4JI:/mnt/f/study/datasecurity/lab3/SEAL$
```

接着创建`demo`文件夹，并写入`test.cpp`、`CMakeLists.txt`和`ckks_example.cpp`文件，以进行安装测试。编写完成后，在控制台依次运行`cmake .`、`make`、`./he`。

一开始提示没有`examples.h`，去`SEAL/native/examples/`文件夹下复制`examples.h`到`demo`文件夹。

```
-- Microsoft SEAL -> Version 4.1.2 detected
-- Microsoft SEAL -> Targets available: SEAL::seal
-- Configuring done
-- Generating done
-- Build files have been written to: /mnt/f/study/datasecurity/lab3/demo
mqx@LAPTOP-2BCGD4JI:/mnt/f/study/datasecurity/lab3/demo$
```

```
mqx@LAPTOP-2BCGD4JI:/mnt/f/study/datasecurity/lab3/demo$ make
[ 50%] Building CXX object CMakeFiles/he.dir/ckks_example.cpp.o
/mnt/f/study/datasecurity/lab3/demo/ckks_example.cpp:1:10: fatal error: examples.h: No such file or directory
    1 | #include "examples.h"
      |          ^~~~~~~~~~~~
compilation terminated.
make[2]: *** [CMakeFiles/he.dir/build.make:76: CMakeFiles/he.dir/ckks_example.cpp.o] Error 1
make[1]: *** [CMakeFiles/Makefile2:83: CMakeFiles/he.dir/all] Error 2
make: *** [Makefile:91: all] Error 2
mqx@LAPTOP-2BCGD4JI:/mnt/f/study/datasecurity/lab3/demo$ make
[ 50%] Building CXX object CMakeFiles/he.dir/ckks_example.cpp.o
[100%] Linking CXX executable he
[100%] Built target he
```

```
mqx@LAPTOP-2BCGD4JI:/mnt/f/study/datasecurity/lab3/demo$ make
Consolidate compiler generated dependencies of target he
[100%] Built target he
mqx@LAPTOP-2BCGD4JI:/mnt/f/study/datasecurity/lab3/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 ]
```

2、CKKS 改写基本思路

由于每次相乘后密文的 scale 都会翻倍，因此需要执行 rescaling 操作来约减一部分，每次执行完 rescaling 后数据的 level 都会改变。

进行加法、乘法运算前要保证参与运算的数据位于同一 level 上，level 不同的数据相互运算前要先构造计算来使 level 高的数据降低其 level，如给其乘 1。

3、流程

计算 $x^3 + y \times z$ 可以将其分为以下几步：

1. 计算 x、y、z 的密文 xc、yc、zc

2. 计算 $x \times x$
3. 计算 $x^2 \times x$
4. 计算 $y \times z$
5. 计算 $x^3 + y \times z$

在最后一步时，x^3的 level 比 $y \times z$ 更低，因此要给 $y \times z$ 乘一个与 $y \times z$ 相同 level 的数据，同时不改变 $y \times z$ 的值，因此可以乘 12，但在实验过程中发现不支持直接使用两个明文 1 相乘，因此可以通过:

> $(yc \times 1) \times (zc \times 1)$

来降低 $y \times z$ 的 level，或者 $(yc \times zc) \times (1 \times 1)$ 中的 1×1 使用一个明文 1 和一个密文 1 相乘。

> 计算 x、y、z 的密文 xc、yc、zc

```
//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);
```

> 计算 $x \times x$ 将密文保存在 tempx2 中。

```
//计算x*x，密文相乘，要进行relinearize和rescaling操作
        evaluator.multiply(xc,xc,tempx2);
        evaluator.relinearize_inplace(tempx2, relin_keys);
        evaluator.rescale_to_next_inplace(tempx2);
```

> 计算 $x^2 \times x$

在计算 $x^2 \times x$ 之前，x 没有进行过 rescaling 操作，所以需要对 x 进行一次乘法和 rescaling 操作，目的是使得 x^2 和 x 在相同的level。最后将密文保存在 tempx3 中。

```
//在计算x*x*x之前，x3没有进行过rescaling操作，所以需要对x3进行一次乘法和rescaling操作，目的是使得x*x 和x
        Plaintext wt;
        encoder.encode(1.0, scale, wt);

//执行乘法和rescaling操作：
        evaluator.multiply_plain_inplace(xc, wt);
        evaluator.rescale_to_next_inplace(xc);

//执行tempx2（x*x）* xc（x*1.0）
        evaluator.multiply_inplace(tempx2, xc);
        evaluator.relinearize_inplace(tempx2,relin_keys);
        evaluator.rescale_to_next(tempx2, tempx3);
```

> 计算 $y \times z$

采取 $(yc \times 1) \times (zc \times 1)$ 的方法，先给 yc 和 zc 分别乘 1.0，然后再进行 yc 和 zc 密文相乘，密文保存在 tempyz 中。

```
//对y z进行一次乘法和rescaling操作
        Plaintext wt1;
        encoder.encode(1.0, scale, wt1);

        evaluator.multiply_plain_inplace(yc, wt1);
        evaluator.rescale_to_next_inplace(yc);

        Plaintext wt2;
        encoder.encode(1.0, scale, wt1);

        evaluator.multiply_plain_inplace(zc, wt1);
        evaluator.rescale_to_next_inplace(zc);

//计算y*z，密文相乘，要进行relinearize和rescaling操作
        evaluator.multiply(yc,zc,tempyz);
        evaluator.relinearize_inplace(tempyz, relin_keys);
        evaluator.rescale_to_next_inplace(tempyz);
```

> 计算 $x^3 + y \times z$

将密文保存在 result_c 中

```
//x^3+y*z
        evaluator.add(tempx3,tempyz,result_c);
```

4、实验结果

```
mqx@LAPTOP-2BCGD4JI:/mnt/f/study/datasecurity/lab3/demo$ ./he
    + Modulus chain index for xc: 2
    + Modulus chain index for temp(x*x): 1
    + Modulus chain index for wt: 2
    + Modulus chain index for xc after xc*wt and rescaling: 1
    + Modulus chain index for tempx3 after rescaling: 0
    + Modulus chain index for yc and zc after yc*wt1 zc*wt2 and rescaling: 1
    + Modulus chain index for yc and zc after yc*zc and rescaling: 0
结果是:

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

一开始的x, y, z的选取如下：

```
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 };
```

因此最后计算 $x^3 + y \times z$ 得到的结果正确。同时可以看到 x^3 和 $y \times z$ 在乘 1 之后的 level 是 0，x^2和 $y \times z$ 在乘 1 之前的 level 是 1，原始密文 xc、yc、zc 的 level 是 2，进行直接二元计算的数据间的 level 相同。

四、心得体会

在本次实验中，我首先学习了C++ SEAL库函数的基本使用方法，掌握了如何利用CKKS算法进行算术密文的同态运算。此外，我还了解到参与运算的数据必须具有相同的level值，对于level值不同的数据，可以通过乘以一个level值相同的1.0来降低其level。最后，通过实际操作将学到的理论知识应用到了实验中，对SEAL加密库的使用也变得更加熟练。