数据安全 -- SEAL应用实践

学号：2013921
姓名：周延霖
专业：信息安全

一、实验名称

SEAL应用实践

二、实验环境

- os:Ubuntu
- tool:cmake、g++
- Programming language: C++
- package: SEAL

三、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 * z 的运算

1、实验背景（理论基础）

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

四、实验过程

1、SEAL 环境搭建

> git clone 加载库资源

新建 mySEAL 文件夹，在该文件夹终端内输入命令：

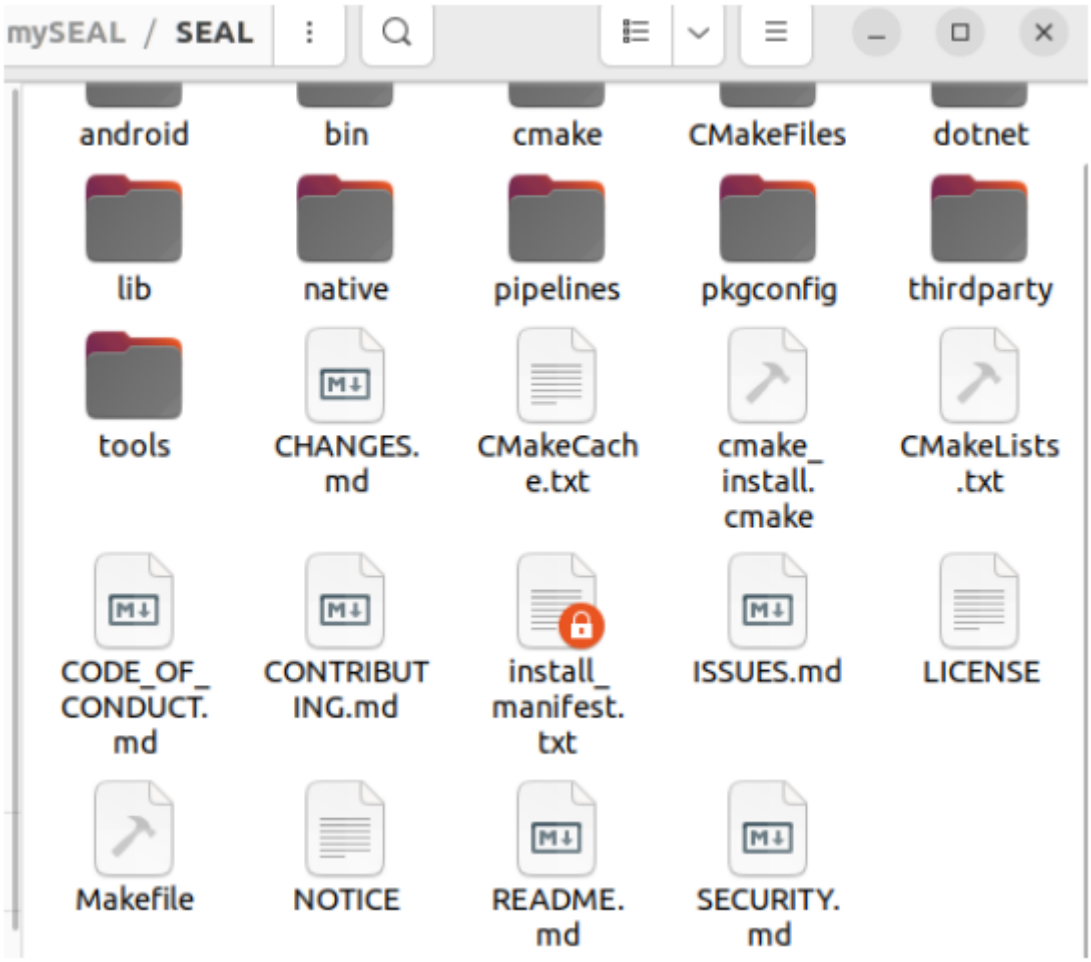
- git clone https://github.com/microsoft/SEAL

> 编译和安装

在刚才的文件夹中输入：

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

输入完如上命令后如下图所示：



2、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 * 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 * x
3. 计算 x^2 * x
4. 计算 y * z
5. 计算 x^3 + y * z

在最后一步时，x^3的 level 比 y * z 更低，因此要给 y * z 乘一个与 y * z 相同 level 的数据，同时不改变 y * 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 * 1) * (zc * 1)`

来降低 y * z 的 level，或者 (yc * zc) * (1 * 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 * 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 * x`

在计算 x^2 * 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 在相同的层

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x3在相同的层
    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 * z

采取 (yc * 1) * (zc * 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

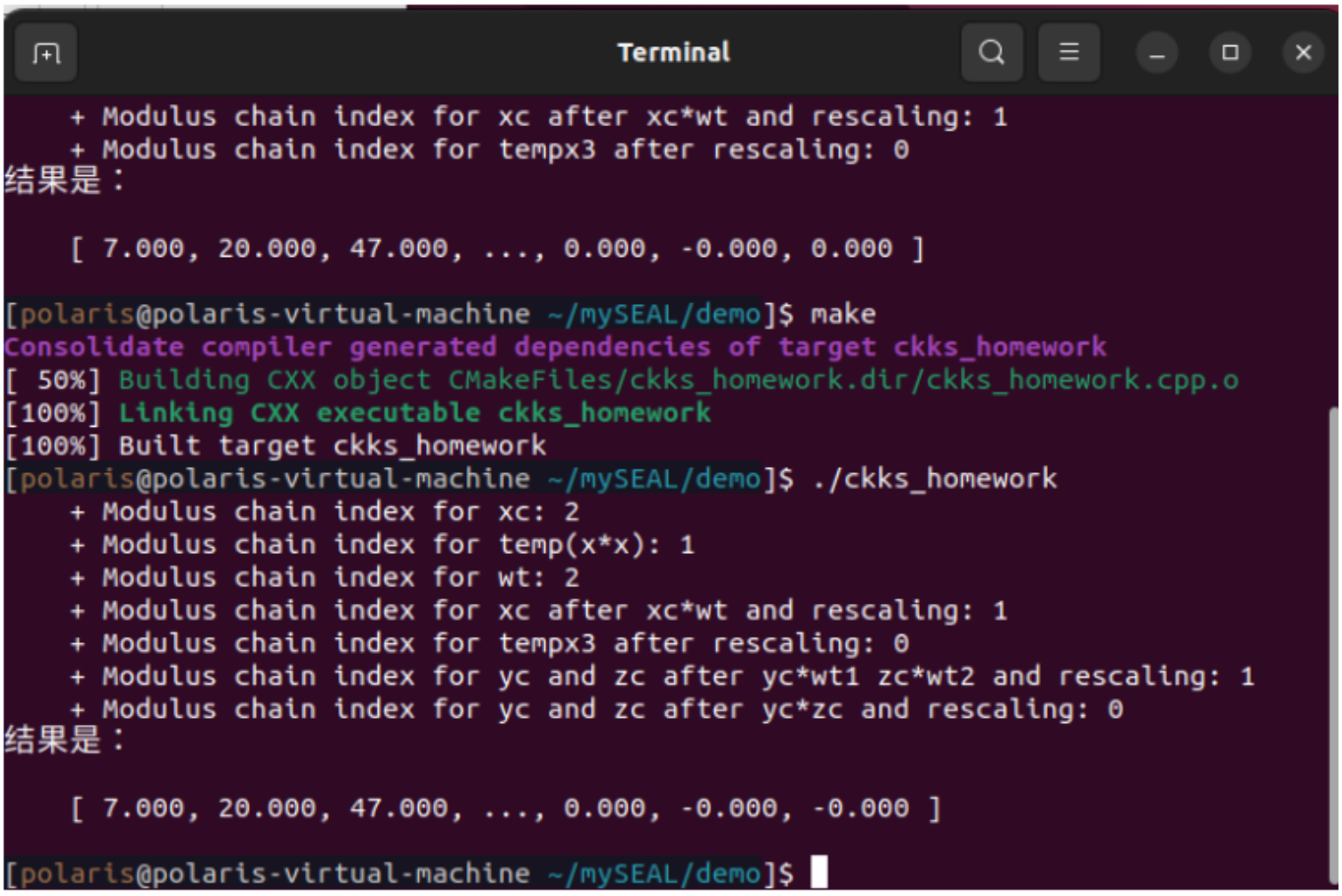
```
//对y z进行一次乘法和rescaling操作
    Plaintext wt1;
    encoder.encode(1.0, scale, wt1);

    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 * 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



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 * z` 得到的结果正确。同时可以看到 x^3 和 y * z 在乘 1 之后的 level 是 0，x^2和 y * 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

五、心得体会

在本次实验中，首先学习到了 C++SEAL 库函数的基本使用，可以运用 CKKS 算法进行算数密文的同态运算

还了解到了“参与运算的数据 level 必须相同”，对于 level 不同的数据可以通过对其乘 1.0 来降低 level，同时所乘的 1.0 的 level 也必须与该数据相同

最后通过实验对所学到的理论知识进行相应的应用，对SEAL加密库的应用也更加的熟练，期待自己未来更好的发展，心想事成、万事胜意、未来可期

六、附录 —— 完整代码

```
#include "examples.h"
#include <vector>
```

```
using namespace std;
using namespace seal;
#define N 3

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);

SEALContext context_server(parms);
    RelinKeys relin_keys;
    keygen.create_relin_keys(relin_keys);
    Evaluator evaluator(context_server);

    Ciphertext tempx2,tempx3,tempyz;
    Ciphertext result_c;
//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);

//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x3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 xc: "
<< context_server.get_context_data(xc.parms_id())->chain_index() << endl;
cout << "    + Modulus chain index for temp(x*x): "
<< context_server.get_context_data(tempx2.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(xc, wt);
    evaluator.rescale_to_next_inplace(xc);

//再次查看xc的层级，可以发现xc与tempx2层级变得相同
cout << "    + Modulus chain index for xc after xc*wt and rescaling: "
<< context_server.get_context_data(xc.parms_id())->chain_index() << endl;

//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);

cout << "    + Modulus chain index for tempx3 after rescaling: "
<< context_server.get_context_data(tempx3.parms_id())->chain_index() <<
endl;
//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);
cout << "    + Modulus chain index for yc and zc after yc*wt1 zc*wt2 and
rescaling: "
<< context_server.get_context_data(yc.parms_id())->chain_index() << endl;
//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);
cout << "    + Modulus chain index for yc and zc after yc*zc and
rescaling: "
<< context_server.get_context_data(tempyz.parms_id())->chain_index() <<
endl;
//x^3+y*z
    evaluator.add(tempx3,tempyz,result_c);


//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);

    cout << "结果是: " << endl;
    print_vector(result,3,3);
    return 0;
}
```