《数据安全》实验报告

姓名： 费泽锟 学号： 2014074 班级： 信安班

**实验名称：**

全同态加密SEAL应用实践

**实验要求：**

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。所要计算的向量在客户端初始化完成并加密，云服务器需要通过提供的加密后的向量进行计算。

在本次实验中，需要根据计算的实现代码改编为在云服务器实现求取的密文计算结果的过程。

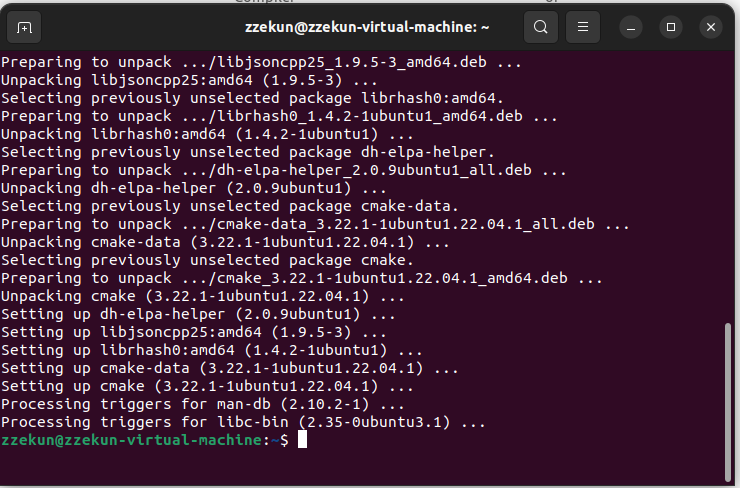
**实验过程：**

1. 配置实验环境

在进行实验之前，我们首先需要对实验环境进行配置，我们使用的是Ubuntu22.04版本，之前已经安装过了g++编译器，我们只需要继续安装好cmake工具即可。使用命令如下：

sudo apt install cmake

成功安装后结果如下图所示：



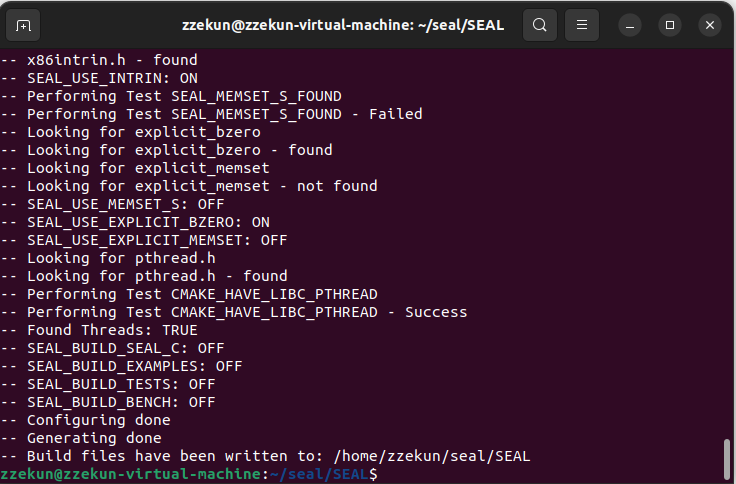
~~~~~~~

1. 编译安装SEAL库

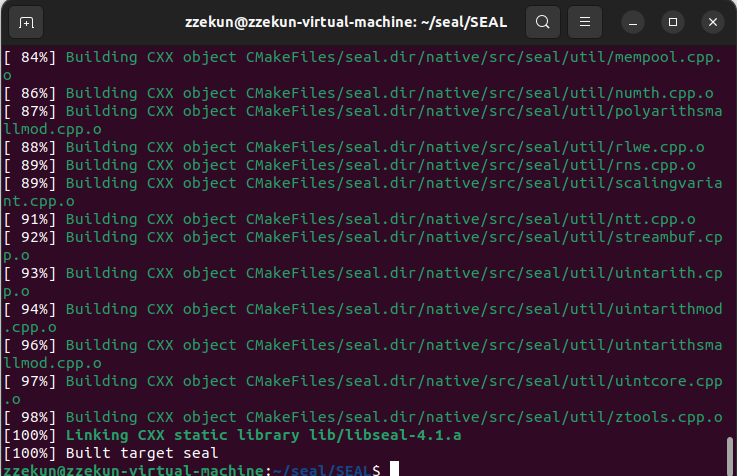
根据教材中的实验指导，在 Ubuntu 的 home 文件夹下建立文件夹 seal，进入该文件夹后，打开终端，输入命令：

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

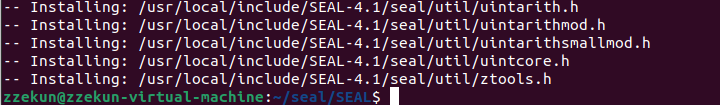
运行完毕，将在 seal 文件夹下自动建立 SEAL 这个新文件夹。接着通过cmake工具对SEAL库进行编译与路径安装。如果在虚拟机之中实在是通过不了NAT去访问GitHub，也可以在本地先clone仓库然后通过共享文件夹将其传输进虚拟机之中。



上图显示cmake编译链接已经成功，可以进行后续的make指令以及make install指令的安装过程，接着运行make指令与make install指令进行安装。



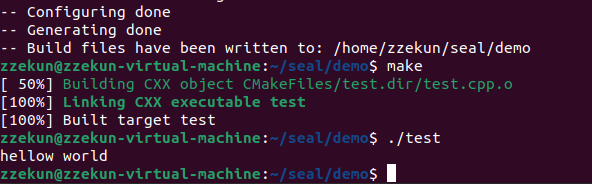
上图显示make命令该步骤成功，最后执行sudo make install命令。



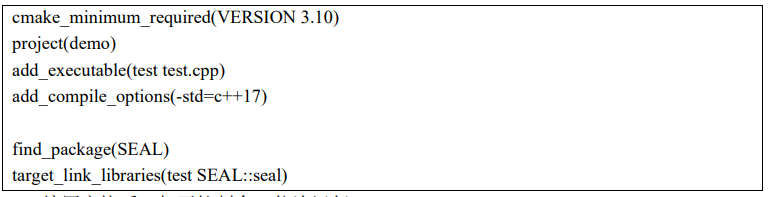
最后显示安装成功，那么至此SEAL库的编译与安装就已经完成了。

~~~~~~~

1. 简单测试SEAL与CMakeLists的正确使用



我们分别对测试代码和CMakeLists.txt文件之中的信息进行简单的分析，上图为简单样例测试成功的显示。



在CMakeLists.txt文件之中规定了cmake工具的版本要求，要进行编译链接的项目名，确定编译的源文件以及生成的可执行文件名以及编译时需要的库文件。

~~~~~~~

1. 应用实例复现改编

CKKS 是一个公钥加密体系，具有公钥加密体系的一切特点，例如公钥加密、私钥解密等。因此，我们的代码中需要以下组件：

·密钥生成器 keygenerator

·加密模块 encryptor

·解密模块 decryptor

CKKS 是一个（level）全同态加密算法（level 表示其运算深度仍然存在限制），可以实现数据的“可算不可见”，因此我们还需要引入：

·密文计算模块 evaluator

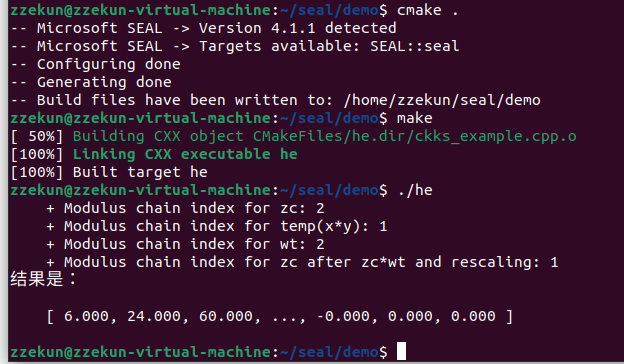
最后，加密体系都是基于某一数学困难问题构造的，CKKS 所基于的数学困难问题在一个“多项式环”上（环上的元素与实数并不相同)，因此我们需要引入编码器来实现数字和环上元素的相互转换：

·编码器

总结下来，整个构建过程为：

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

接着我们需要根据教材之中给出的的例子进行适当改编，将其修改为能够进行的云服务器的计算。我们首先先熟悉计算的流程，最终能够成功地得到结果：



接着我们需要根据实验的主体流程进行分析与改编：

本次实验实现的是同态加密算法最直观的应用——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。

接着我们通过对源代码的改进，对源代码和实验流程进行详细的分析与实现：

1. 对参数部分的修改

在SEAL框架之中，我们对乘法运算必须要保持运算的数据均在同一层才能够预算，所以我们需要对运算进行适当的修改，可以修改为的形式，如果修改为该种形式的情形，那么我们也就会进行两层的乘法运算，所以其实保持原有的参数即可：

Poly\_module\_degree = 8196; coeff\_modulus={60,40,40,60};scale = 2^40

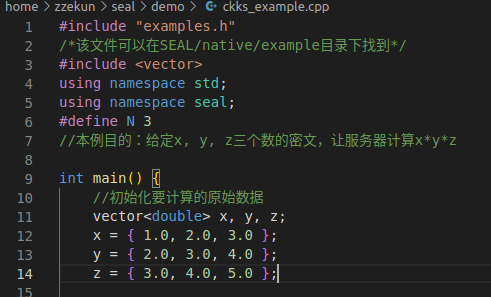
原有的参数就可以保持进行两层的乘法运算。但是我们也可以使用参数形式如下：

Poly\_module\_degree = 8196; coeff\_modulus={50,30,30,30.50};scale = 2^30

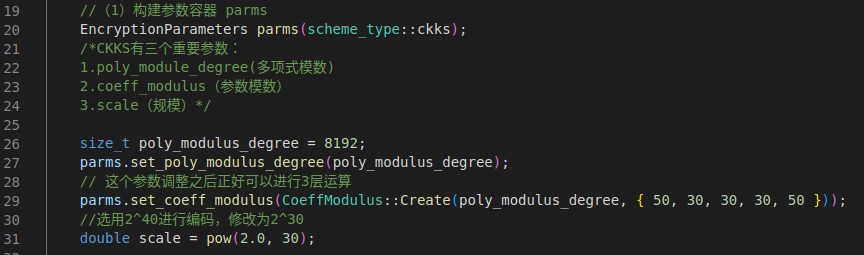
上述的参数形式能够维持三层的乘法运算，实验之中我们采用的是第二种参数形式，虽然这样会降低密文规模。

1. 源代码分析与修改

我们接下来对源代码进行分析与修改：



首先我们需要使用seal的命名空间，并且因为计算过程之中只有3个数的密文，所以需要定义好N为3，在seal框架之中数据的运算可以是vector容器类的运算。



接下来就是对CKKS算法的参数进行设置，其中：

1. poly\_modulus\_degree（polynomial modulus）

该参数必须是 2 的幂，如 1024, 2048, 4096, 8192, 16384, 32768，当然再大点也没问题。更大的 poly\_modulus\_degree 会增加密文的尺寸，这会让计算变慢，但也能让你执行更复杂的计算。

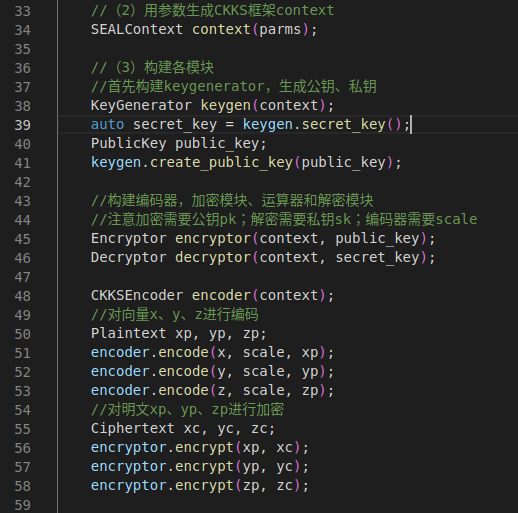
1. coefficient modulus

这是一组重要参数，因为 rescaling 操作依赖于 coeff\_modules。简单来说，coeff\_modules 的个数决定了你能进行 rescaling 的次数，进而决定了你能执行的乘法操作的次数

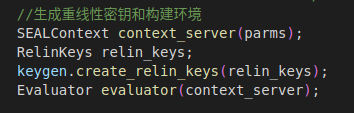
1. Scale

Encoder 利用该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。

接下来我们需要生成CKKS框架context，然后去生成基于格的公钥与私钥，并对vector的浮点数进行编码与加密的操作，其中编码的目的在于将明文编码进向量空间。

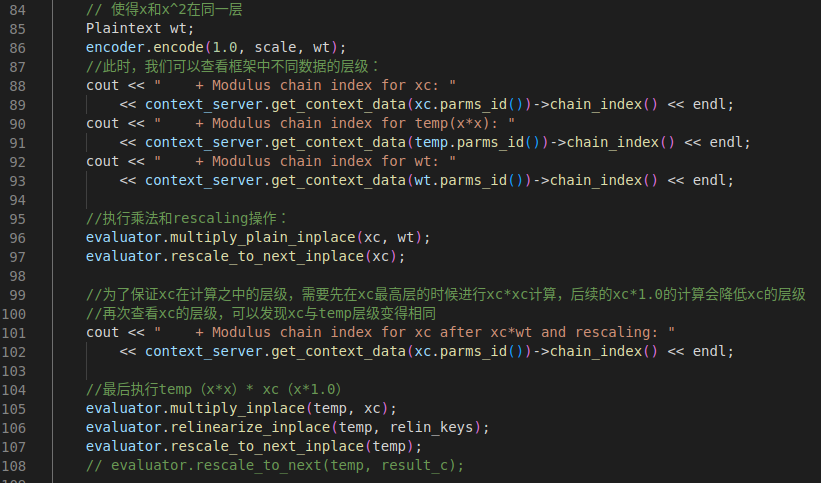


我们还需要生成重线性密钥并且进一步构建环境和运算器evaluator，其中重线性过程需要重线性密钥来进行配合运算。

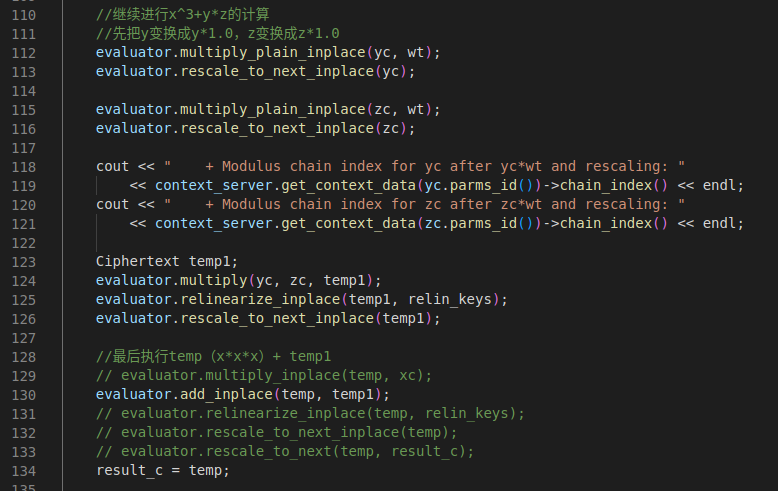


我们已经构建好了参数、密钥以及进行密文计算的环境，那么接下来我们就只需要设计好合理的计算流程就能够成功地进行同态计算了。我们需要获得的计算流程为，我们首先可以进行的计算，这一步计算为同层计算可以直接正常的进行，但是进而进行的计算时，此时的的计算结果与x不处于同一层之中，需要进行再线性化与再缩放，此时我们只需要将x转化为，即可将这两部分数据转化为同层次的数据进行计算。

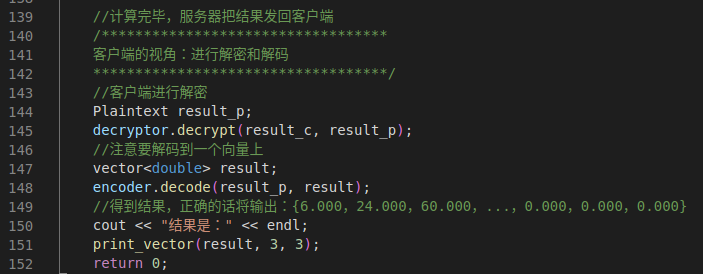
在这一部分之中，我们需要注意的一点是：我们需要先进行的计算，因为如果我们先进行的计算并且不使用temp数据对其结果进行存储的话，可能会导致x原本处于的运算层数的降低，具体的计算过程如下：



我们在第一步之中已经完成了第一部分的运算，接下来需要设计的就是第二部分也就是部分的计算，这里我们的变换策略为对y和z数据均去乘1.0，降低层数，这时运算的结果与计算结果的层数就相同了，接着就可以直接进行加法运算。

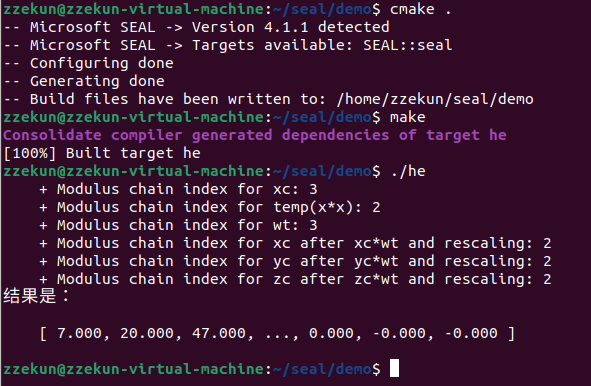


在得到了密文的计算结果之后，在客户端只需要进行解密已经从明文域的解码操作就能够成功地获得最终的明文运算结果了。



最终我们能够成功地得到最后的结果，也就是的计算结果，得到的最后结果为：

[7.000, 20.000, 47.000]，但是对于vector的最后的部分出现了-0.000的原因仍需继续研究。



**心得体会**

通过本次实验，我们学习了全同态加密算法的原理，实验中给出了一个非常贴近现实的使用CKKS的场景，我们完成了最终密文计算，明文解密的过程。

经过学习与实践，将在密码学课上学习的知识和数据安全学习的内容结合在了一起，更好的理解了CKKS算法和SEAL框架的特性及其适合的应用场景。