

**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 孙蕗

1. **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x3+y\*z的运算

1. **实验原理**
2. 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。
3. 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大工程的语言。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。
4. 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。

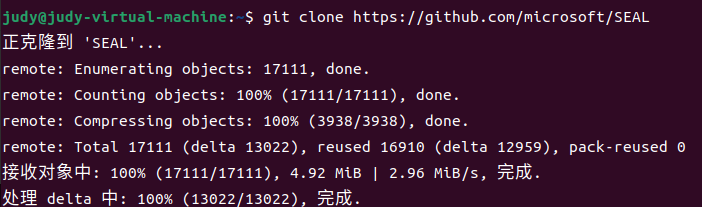
SEAL根据默认的参数创建了一个modulus switching chain ，在同一个链上的加密实例除了coefficient modulus 其他都相同。

“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。

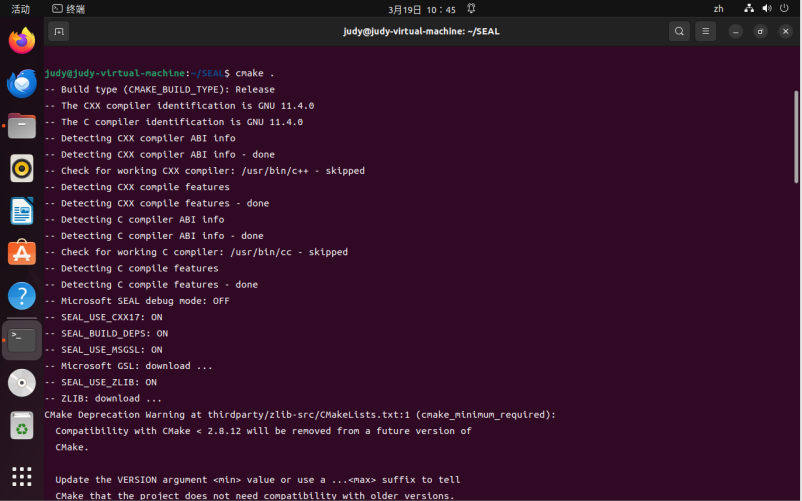
1. **实验过程**
2. **配置环境**

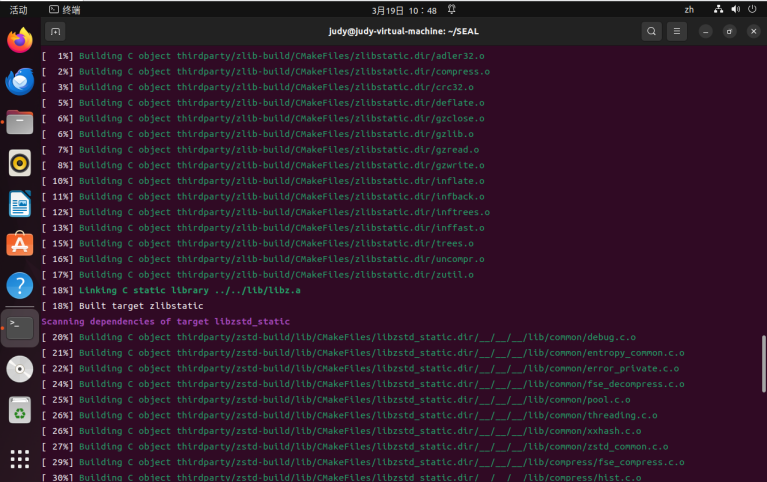
从github上拉取开源的SEAL库

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



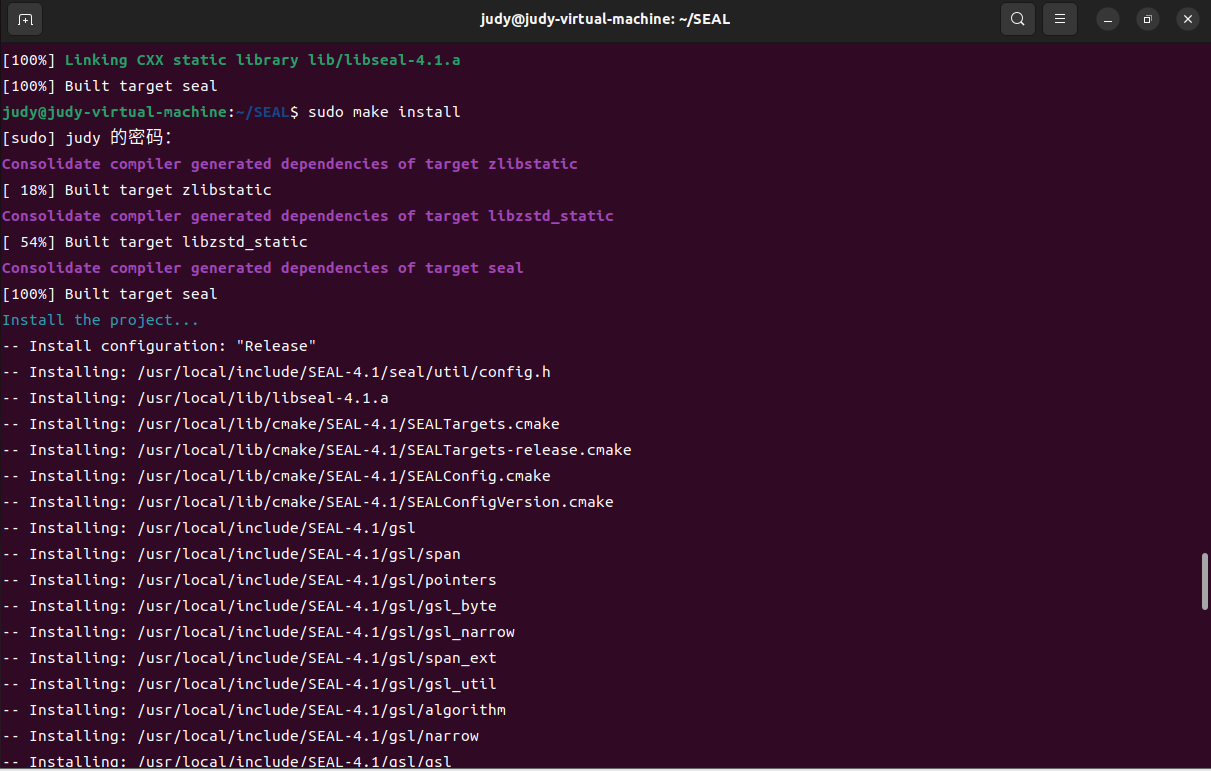
进入SEAL文件夹，输入cmake .和make





输入

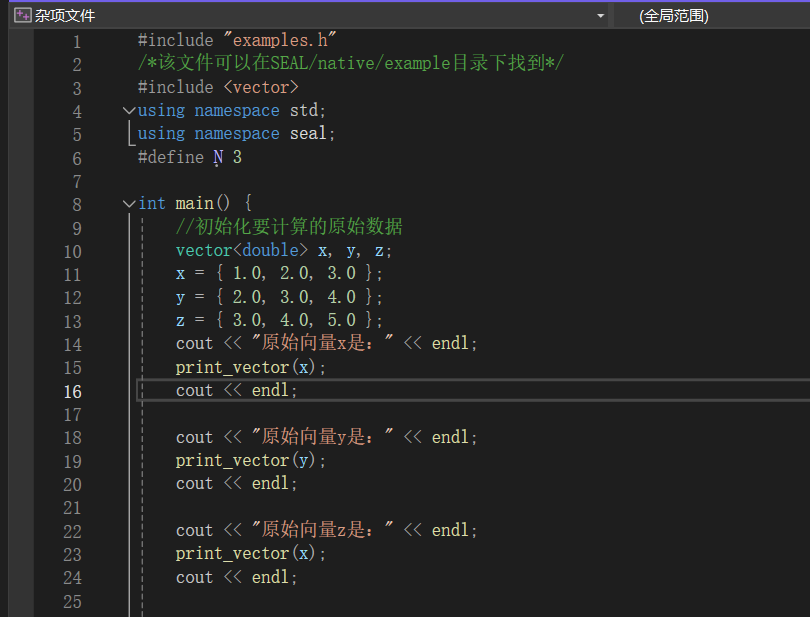
|  |
| --- |
| sudo make install |

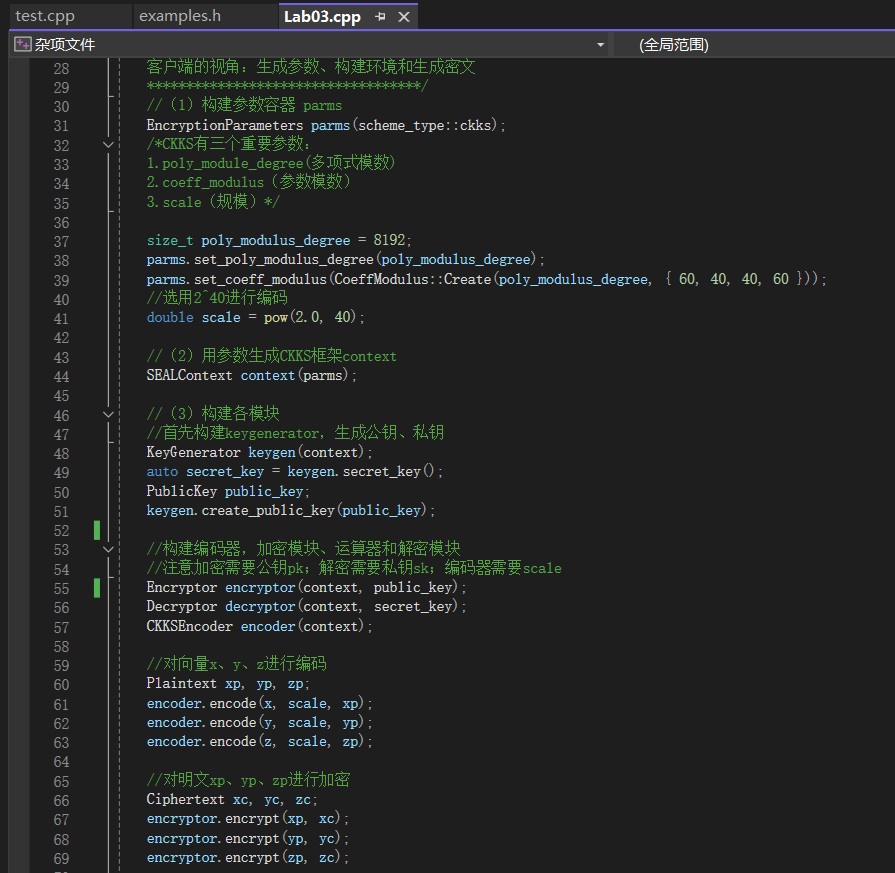


1. **代码实现**

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。





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

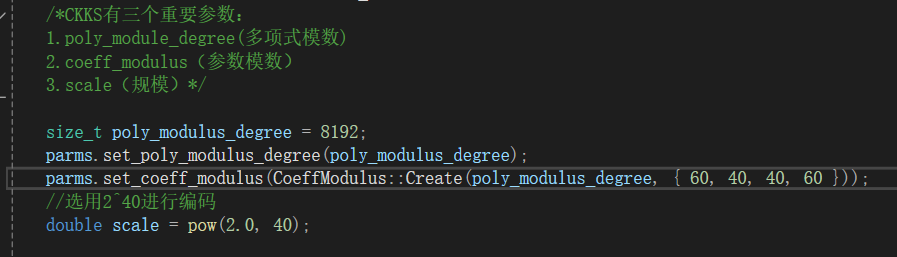
⑥使用evaluator对密文c运算为密文c

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

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。



[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。

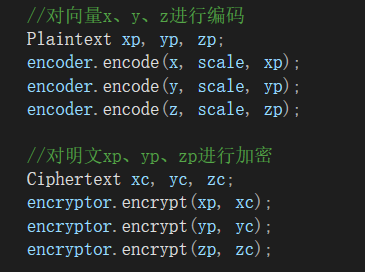
(60,40,40,60}有以下含义：①coeff\_modules总位长200(60+40+40+60)位，在poly\_modulus\_degrees上限之下；②最多进行两次(两层)乘法操作。该系列数字的选择不是随意的，总位长不能超过上表限制；最后一个参数为特殊模数，其值应该与中间模数的最大值相等；中间模数与scale尽量相近。如果将模数变大，则可以支持更多层级的乘法运算，比如poly\_modulus为16384则可以支持coeff\_modules=(60,40,40,40,40,40,40,60},也就是6层的运算。

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。

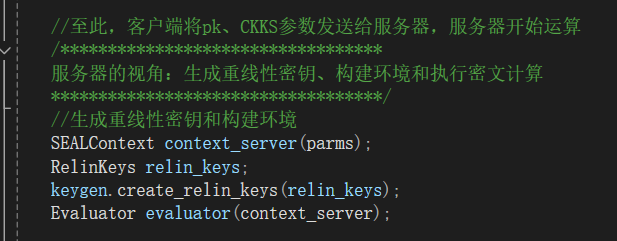


用参数生成CKKS框架context，构建keygenerator模块，分别使用keygen.secret\_key()和keygen.create\_public\_key生成公钥和私钥。构建编码器加密模块运算器和解密模块，需要注意的是加密需要公钥pk，解密需要私钥sk，编码器需要scale。

然后对三个向量进行编码和加密

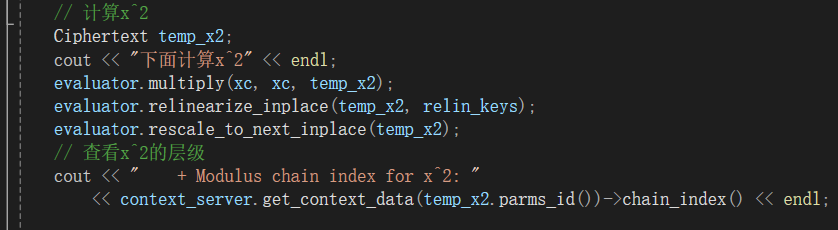


至此，客户端将pk、CKKS参数发送给服务器，服务器开始运算。服务器的视角：生成重线性密钥、构建环境和执行密文计算。生成重线性密钥与构建环境与样例代码保持一致。

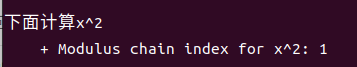


1. 执行密文计算
2. 计算x2,即x\*x,将结果保存在temp\_x2中。

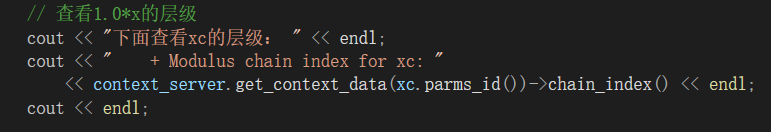
计算x\*x，密文相乘，要进行relinearize和rescaling操作



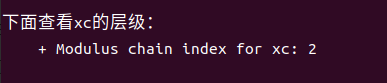
计算完成后查看x2的层级



1. 准备计算x\*x2,先查看xc的层级

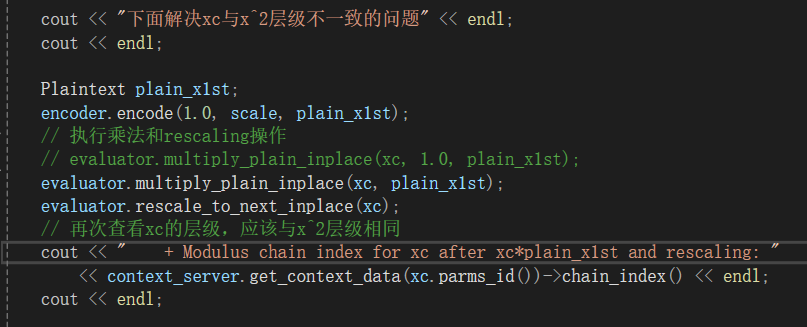


查看xc的层级

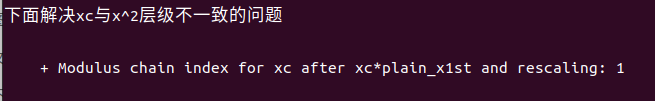


可以看到，xc与x2层级是不一样的，下面解决两者层级不一致的问题。

1. 解决xc与x2层级不一致的问题

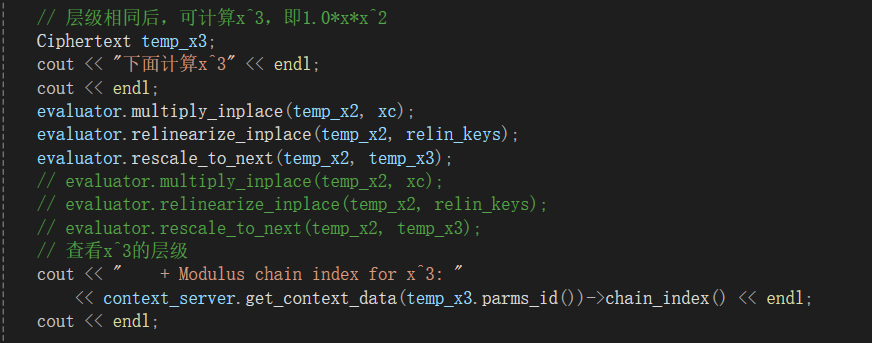


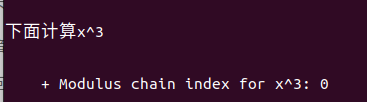
通过将xc与乘法单位元1相乘，将xc的层级从2变成1，执行乘法和rescaling操作。



1. 将前面两步操作的结果相乘，计算xc\*x2,得到x3，存在temp\_x3中。此步需要进行relinearize和rescaling 操作。

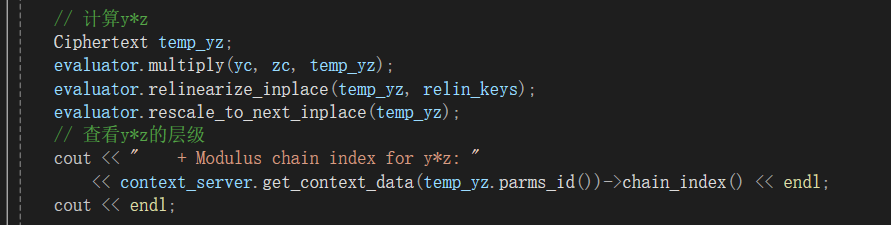
计算完成后查看x3的层级。

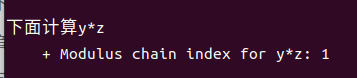




1. 计算y\*z，结果存在temp\_yz中。与前面的步骤相同，同样需要进行relinearize和rescaling 操作。

计算完成后查看y\*z的层级

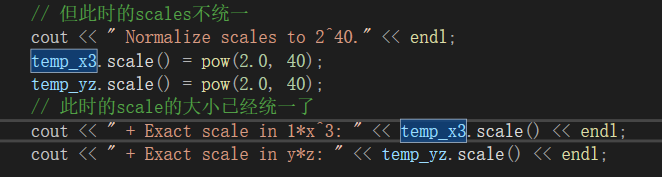


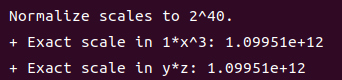


发现y\*z的层级和scale（执行rescaling次数不同）都与x3不一致，需要统一scale大小后再进行层级统一。

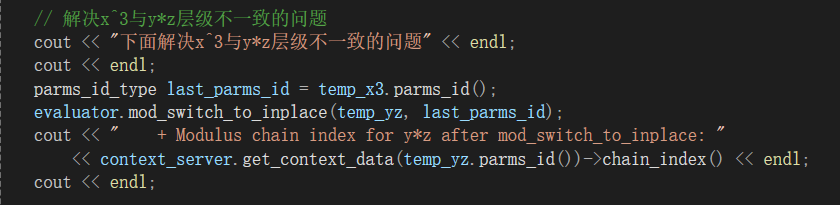
1. 统一y\*z与x^3的scale

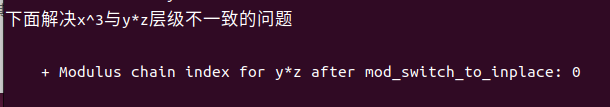
P2和P1非常接近于240，可以简单地“欺骗”SEAL 并将比例尺设置为相同。将1.0 ∗x ∗x^2的比例尺更改为 240 仅意味着通过 2120/(P22 ∗ P1) 缩放1.0∗x ∗x2的值，非常接近于 1。



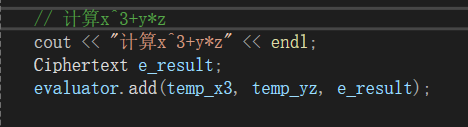


1. 统一y\*z与x3层级，Evaluator::mod\_switch\_to\_inplace方法会将明文/密文的模切换到下一位，结果保存在原来的密文里，沿着chain向下减一个level。

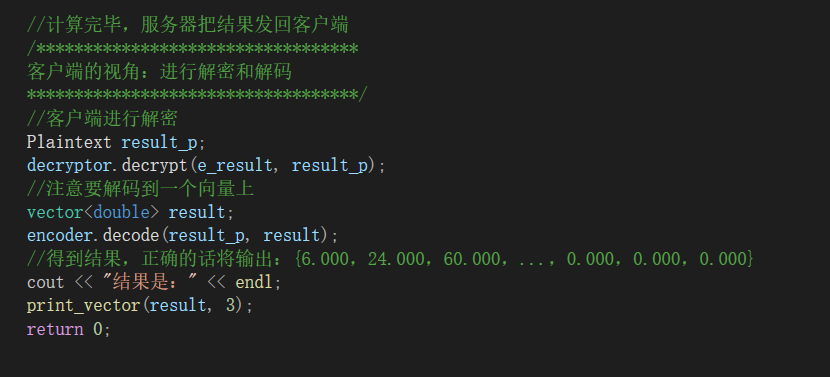




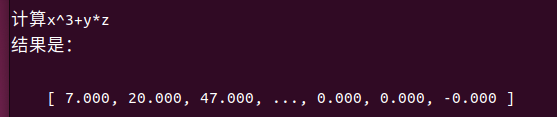
1. 计算x^3+y\*z，使用evaluator.add实现加法运算，结果保存在e\_result中



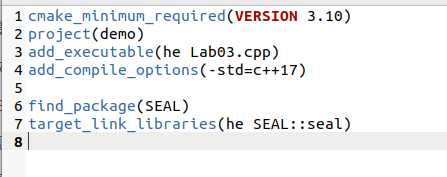
1. 客户端进行解密和解码



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

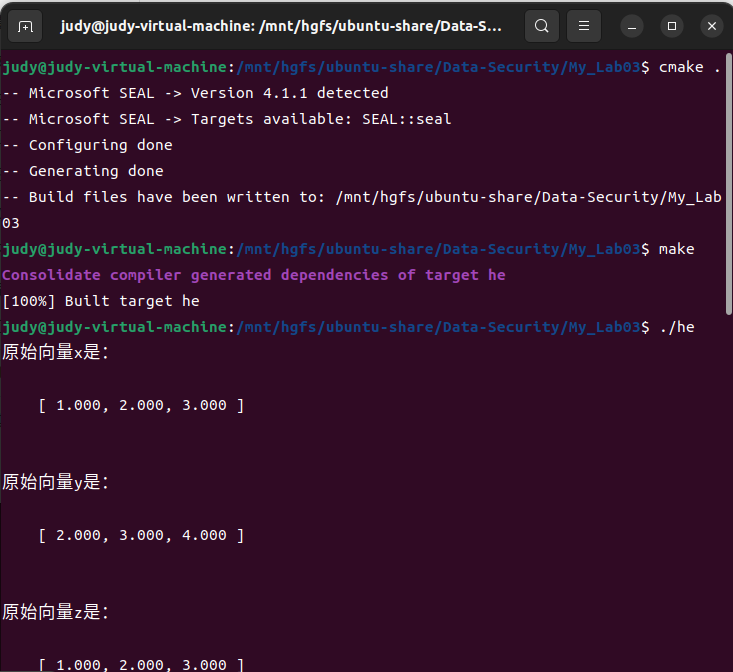


1. 仿照样例修改CmakeList.txt

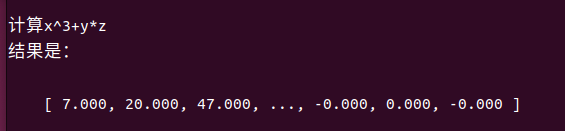


1. 运行如下命令完成本次实验

|  |
| --- |
| cmake .  make  ./he |



计算结果如下：



1. **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。