

# 实验三 SEAL库的CKKS同态加密实践

2112852 密码科学与技术 胡佳佳

## 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。

## 理论知识

### 同态加密 (HE)

HE是一种特殊的加密方法，它允许直接对加密数据执行计算，如加法和乘法，而计算过程不会泄露原文的任何信息。计算的结果仍然是加密的，拥有密钥的用户对处理过的密文数据进行解密后，得到的正好是处理后原文的结果。

根据支持的计算类型和支持程度，同态加密可以分为以下三种类型：

- **半同态加密** (Partially Homomorphic Encryption, **PHE**)：只支持加法或乘法中的一种运算。其中，只支持加法运算的又叫加法同态加密 (Additive Homomorphic Encryption, AHE)；
- **部分同态加密** (Somewhat Homomorphic Encryption, **SWHE**)：可同时支持加法和乘法运算，但支持的计算次数有限；
- **全同态加密** (Fully Homomorphic Encryption, **FHE**)：支持任意次的加法和乘法运算。

### CKKS (第四代全同态加密方案)

CKKS (Cheon-Kim-Kim-Song) 方案支持针对实数或复数的浮点数加法和乘法同态运算，但是得到的计算结果是近似值。因此，它适用于不需要精确结果的场景。

支持浮点数运算这一功能在实际中有非常重要的作用，如实现机器学习模型训练等。这个方案的性能也非常优异，大多数算法库都实现了CKKS。

## 实验环境

由于在Ubuntu22虚拟机上多次尝试未能下载cmake SEAL文件所需的zstandard，故在本机window10上直接进行SEAL实验。需要的其它的环境还有visual studio2022。

## 实验步骤

### 安装SEAL库

#### 1.从github上克隆SEAL库下来

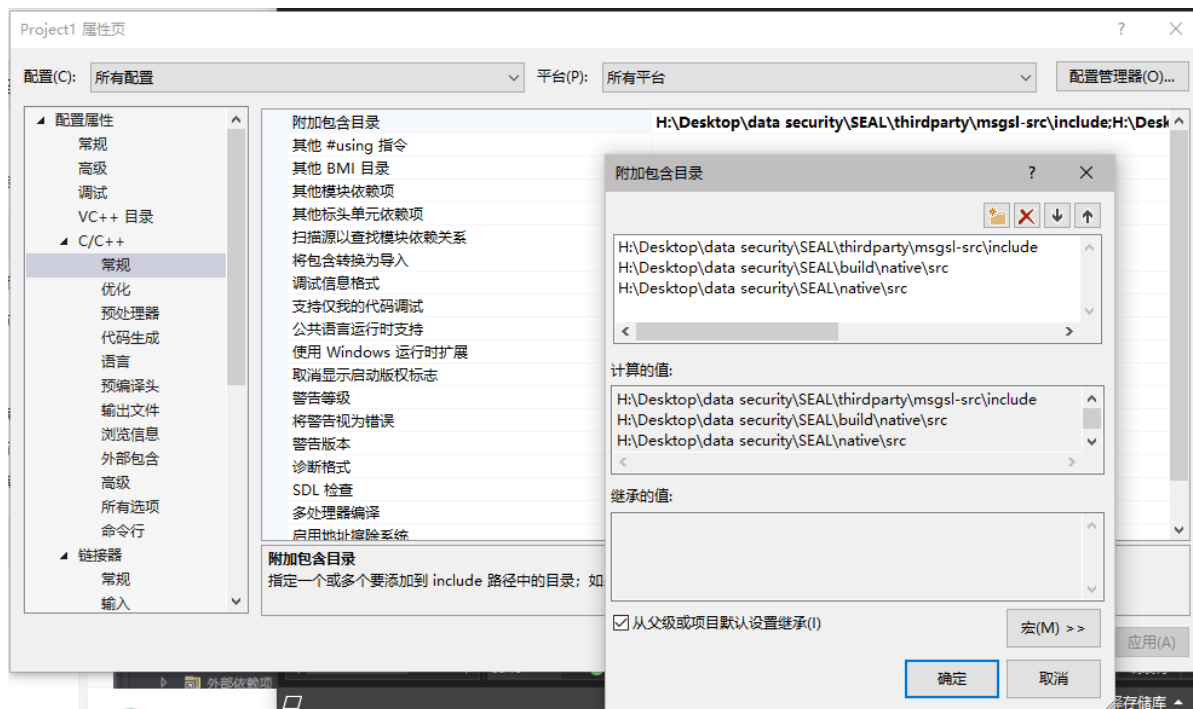
```
git clone https://github.com/microsoft/SEAL
```

## 2.在x64 Native Tools Command Prompt for VS 2022上，切换到SEAL当前目录并执行以下命令

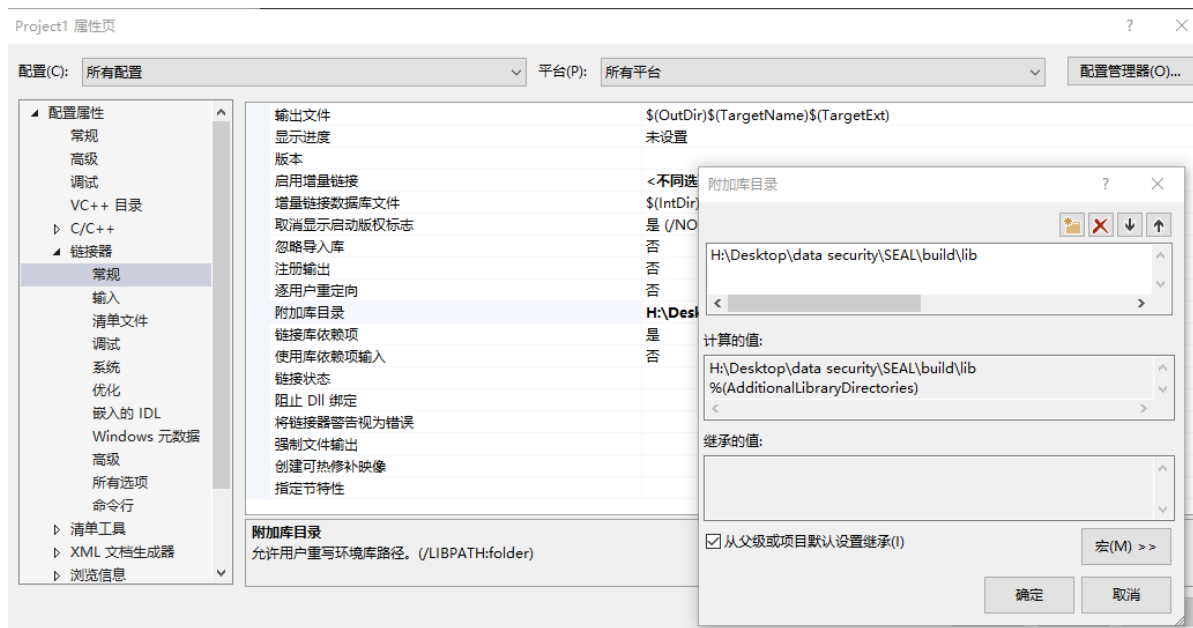
```
cmake -S . -B build -G Ninja
cmake --build build
cmake --install build
```

## 3.在VS 2022的项目属性中，添加附加包含项，附加库目录和附加依赖项

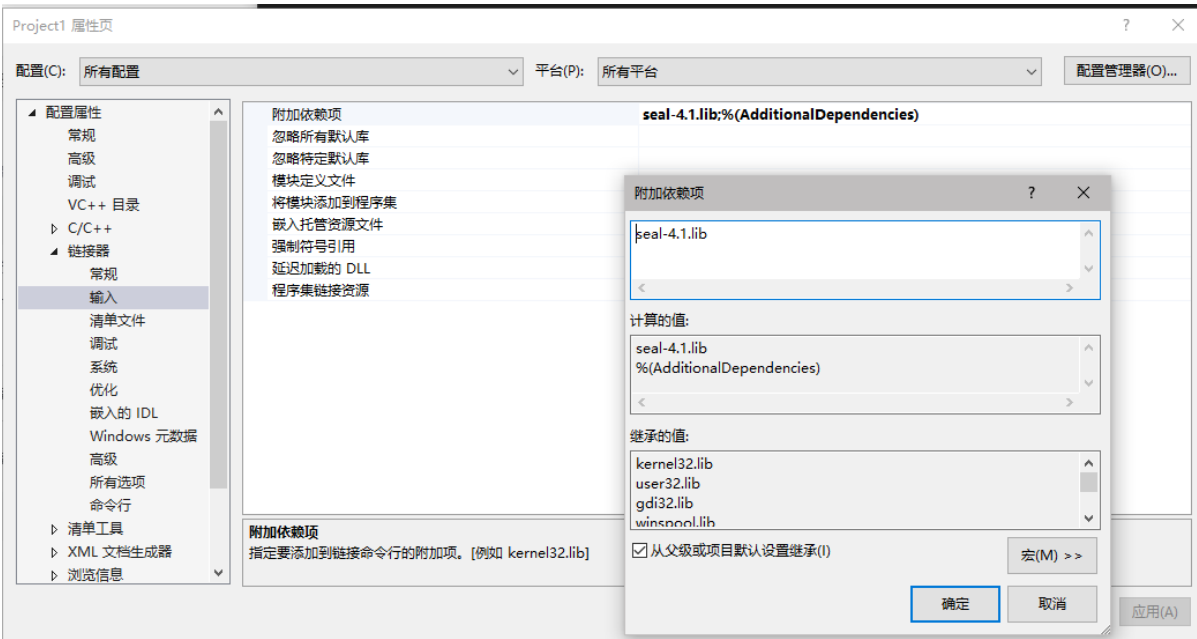
### 附加包含项



### 附加库目录

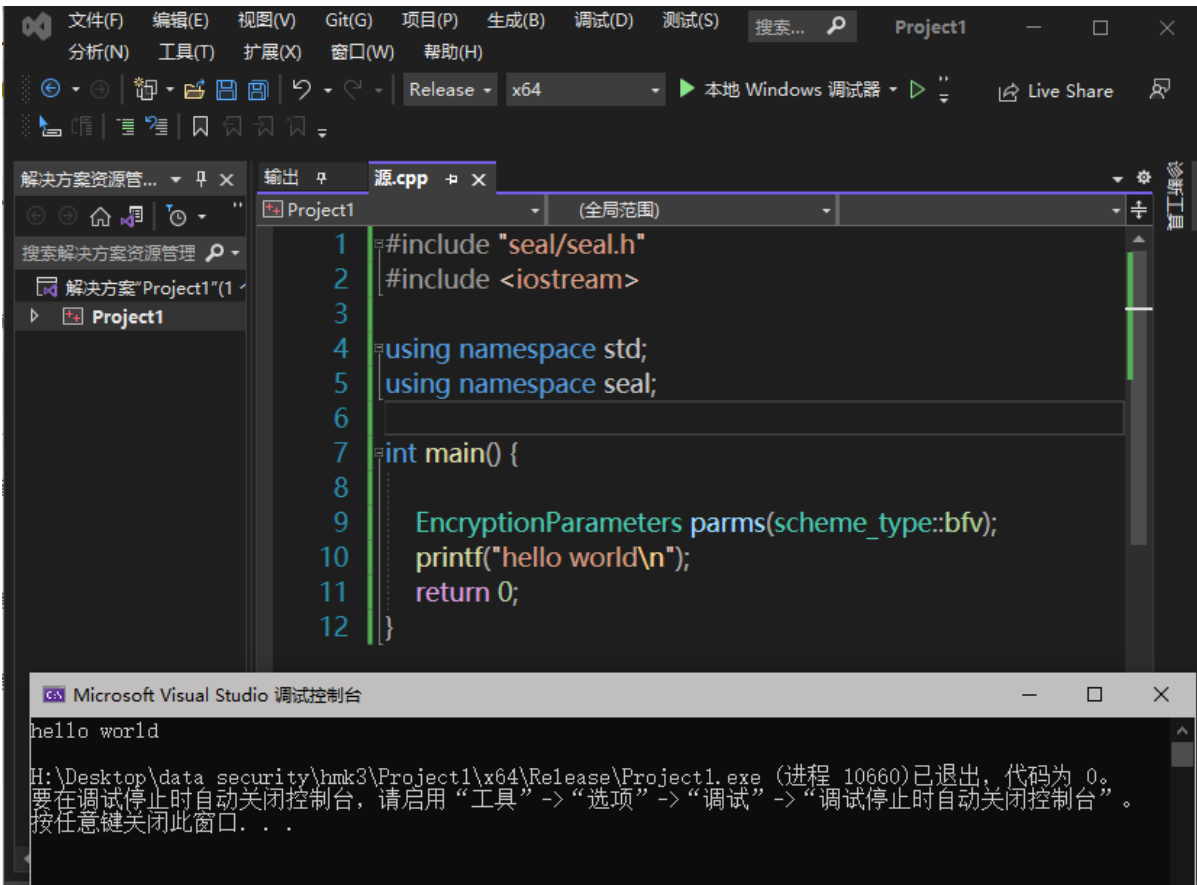


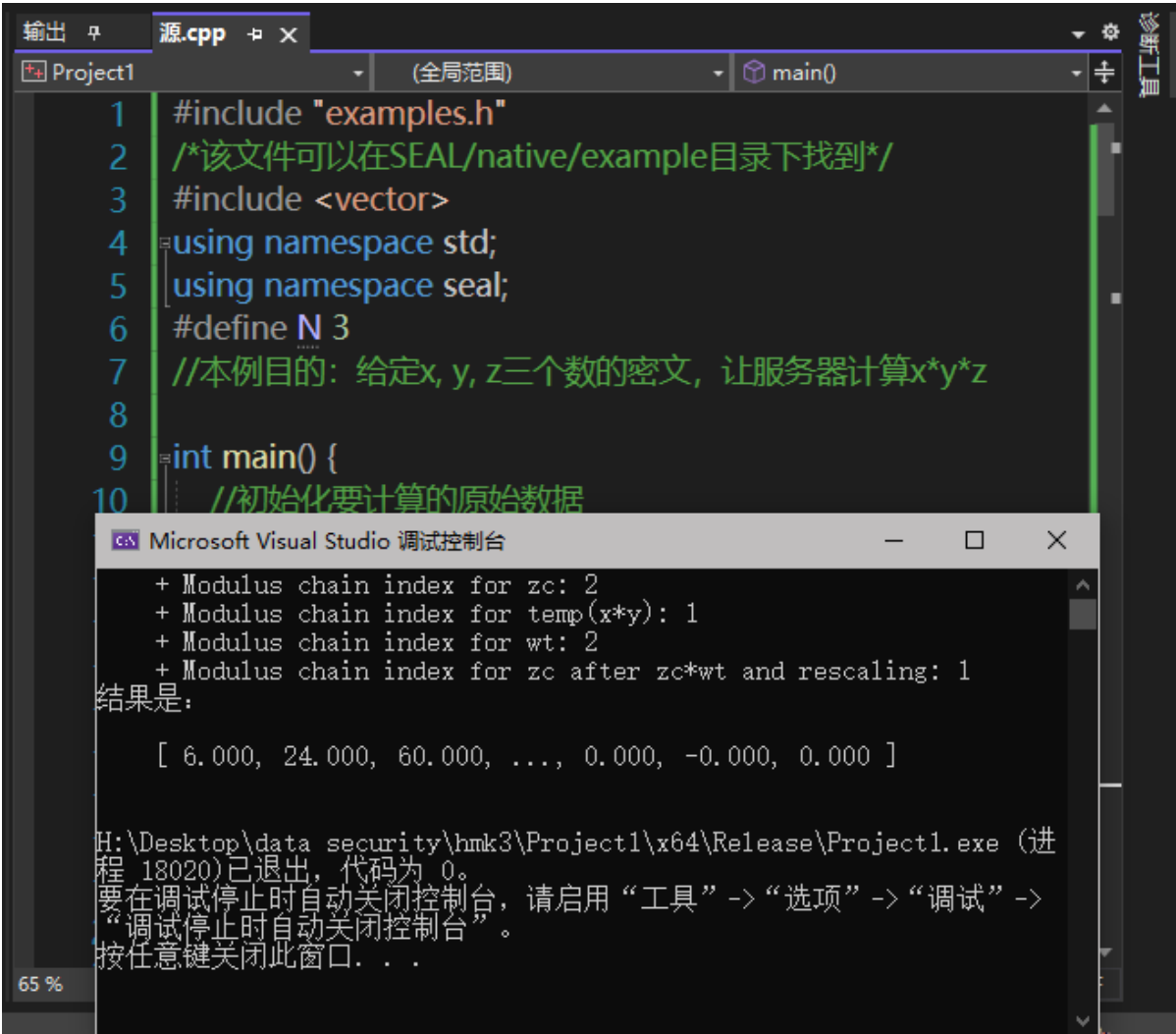
附加依赖项



测试demo

hello\_world test





The screenshot shows the Visual Studio IDE with a C++ source file named `源.cpp` (Source.cpp) open. The code defines a namespace `seal` and a constant `N` with a value of 3. It includes a header `examples.h` and a `main` function. The debug console window is open, showing the output of the program. The output includes several lines of text indicating the modulus chain index for `zc`, `temp(x*y)`, `wt`, and `zc` after `zc*wt` and rescaling. The final output is a vector of values: `[ 6.000, 24.000, 60.000, ..., 0.000, -0.000, 0.000 ]`. The console also shows a message indicating that the program has exited with code 0.

```
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 //本例目的: 给定x, y, z三个数的密文, 让服务器计算x*y*z
8
9 int main() {
10 //初始化要计算的原始数据
```

Microsoft Visual Studio 调试控制台

```
+ 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 ]
H:\Desktop\data security\hmk3\Project1\x64\Release\Project1.exe (进
程 18020)已退出, 代码为 0。
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->
“调试停止时自动关闭控制台”。
按任意键关闭此窗口...
```

## 计算 $x^3+y*z$

### CKKS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 $n$

## 客户端

### 生成参数

```
// (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
```

### 构建环境

```
// (2) 用参数生成CKKS框架context
SEALContext context(parms);
```

### 生成密文

```
// (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);
```

## 服务器

### 生成重线性密钥并构建环境

```
//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);
```

### 执行密文计算

```
/*对密文进行计算，要说明的原则是：
-加法可以连续运算，但乘法不能连续运算
-密文乘法后要进行relinearize操作
-执行乘法后要进行rescaling操作
-进行运算的密文必需执行过相同次数的rescaling（位于相同level）*/
Ciphertext temp;
Ciphertext result_c;

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

//在x^2*x计算之前，x没有进行过rescaling操作，所以需要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
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(temp.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.relinearize_inplace(xc, relin_keys);
evaluator.rescale_to_next_inplace(xc);
cout << "    + Modulus chain index for xc: "
      << context_server.get_context_data(xc.parms_id())->chain_index() << endl;

//最后执行temp (x*x) * xc (x*1.0)
evaluator.multiply_inplace(temp, xc);
evaluator.relinearize_inplace(temp, relin_keys);
evaluator.rescale_to_next_inplace(temp);

cout << "    + Modulus chain index for temp: "
      << context_server.get_context_data(temp.parms_id())->chain_index() <<
endl;

//计算y*z=yc(y*1.0)*zc(z*1.0)也是类似
```

```

evaluator.multiply_plain_inplace(yc, wt);
evaluator.relinearize_inplace(yc, relin_keys);
evaluator.rescale_to_next_inplace(yc);

evaluator.multiply_plain_inplace(zc, wt);
evaluator.relinearize_inplace(zc, relin_keys);
evaluator.rescale_to_next_inplace(zc);

evaluator.multiply(yc, zc, temp1);
evaluator.relinearize_inplace(temp1, relin_keys);
evaluator.rescale_to_next_inplace(temp1);

cout << "    + Modulus chain index for y*z: "
      << context_server.get_context_data(temp1.parms_id())->chain_index() <<
endl;

//相加
evaluator.add(temp, temp1, result_c);
cout << "    + Modulus chain index for result_c: "
      << context_server.get_context_data(result_c.parms_id())->chain_index() <<
endl;

```

## 客户端

### 解密

```

//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);

```

## 运行结果

可以看到运行中各个变量的chain index变化，简单验证结果前三列是正确的。

```

Microsoft Visual Studio 调试控制台
+ 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: 1
+ Modulus chain index for temp: 0
+ Modulus chain index for y*z: 0
+ Modulus chain index for result_c: 0
结果是:
[ 7.000, 20.000, 47.000, ..., 0.000, 0.000, -0.000 ]
H:\Desktop\data security\hmk3\Project1\x64\Release\Project1.exe (进程 20748)已退出, 代码为 0。
要在调试停止时自动关闭控制台, 请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口。...

```

## 参考文献

- 【1】 [同态加密库Seal库的安装 \(win11+VS2022\) 同态解密seal安装-CSDN博客](#)