Test Code

```
#include <iostream>
#include <libscapi/include/mid_layer/DamgardJurikEnc.hpp>

void test_enc_and_dec() {

    std::cout<<"****** Test Encryption and Decryption ******"<<std::endl;
    auto random = get_seeded_prg();
    // create a DamgardJurik encryption object
    DamgardJurikEnc encryptor;

    // generate a key pair using the object
    DJKeyGenParameterSpec spec;
    auto pair = encryptor.generateKey(&spec);
    encryptor.setKey(pair.first, pair.second);
    biginteger r = getRandomInRange(0, dynamic_pointer_cast<DamgardJurikPu
    // create a plaintext
    BigIntegerPlainText plaintext("123");
    std::cout<<"Original plaintext is: "<<plaintext.getX()<<std::endl;
    shared_ptr<Plaintext> sharePlaintext = make_shared<BigIntegerPlainText
    shared_ptr<AsymmetricCiphertext> cipher = encryptor.encrypt(sharePlain
    shared_ptr<Plaintext> plain = encryptor.decrypt(cipher.get());
    auto decryptedPlaintext = *(dynamic_cast<BigIntegerPlainText*>(plain.g
    //std::cout<<"Decrypted plaintext is: "<<plain.get()<<std::endl;
    std::cout<<"Decrypted plaintext integer is: "<<decryptedPlaintext.getX
}

void test_add_and_mult() {

    std::cout<<"****** Test Homomorphic Addition and Multiplication ******
    auto random = get_seeded_prg();
    // create a DamgardJurik encryption object
    DamgardJurikEnc encryptor;

    // generate a key pair using the object
```

```
    DJKeyGenParameterSpec spec;
    auto pair = encryptor.generateKey(&spec);
    encryptor.setKey(pair.first, pair.second);
    biginteger r1 = getRandomInRange(0, dynamic_pointer_cast<DamgardJurikE
    biginteger r2 = getRandomInRange(0, dynamic_pointer_cast<DamgardJurikE

    // create a plaintext
    BigIntegerPlainText plaintext("123");
    auto sharePlaintext = make_shared<BigIntegerPlainText>(plaintext);
    auto ciphertext = encryptor.encrypt(sharePlaintext, r1);
    BigIntegerPlainText anotherPlaintext("123");
    auto anotherSharePlaintext = make_shared<BigIntegerPlainText>(anotherP
    auto anotherCiphertext = encryptor.encrypt(anotherSharePlaintext, r2);

    // homomorphic addition
    auto addCiphertext = encryptor.add(ciphertext.get(), anotherCiphertext
    shared_ptr<Plaintext> plain = encryptor.decrypt(addCiphertext.get());
    auto decryptedPlaintext = *(dynamic_cast<BigIntegerPlainText*>(plain.g
    std::cout<<"Plaintext addition is: "<<plaintext.getX() + anotherPlaint
    std::cout<<"Decrypted plaintext addition integer is: "<<decryptedPlain

    // homomorphic multiplication
    // const BigIntegerPlainText constant("321");
    // biginteger* constant = 100;
    biginteger a = 100;
    auto multiCiphertext = encryptor.multByConst(ciphertext.get(), a);
    shared_ptr<Plaintext> multiPlain = encryptor.decrypt(multiCiphertext.g
    auto decryptedMultiPlaintext = *(dynamic_cast<BigIntegerPlainText*>(mu
    std::cout<<"Plaintext multiplication is: "<<plaintext.getX() * a<<std:
    std::cout<<"Decrypted plaintext multiplication integer is: "<<decrypte
}

void test_fixed_precision_add_and_mult() {

    std::cout<<"****** Test Fixed Precision Addition and Multiplication **
    auto random = get_seeded_prg();
    // create a DamgardJurik encryption object
    DamgardJurikEnc encryptor;

    // generate a key pair using the object
```

```
    DJKeyGenParameterSpec spec;
    auto pair = encryptor.generateKey(&spec);
    encryptor.setKey(pair.first, pair.second);
    biginteger r = getRandomInRange(0, dynamic_pointer_cast<DamgardJurikPu

    // create two float plaintext
    float v1 = 0.0123456, v2 = 98.7654321;
    long fv1 = v1 * 100000000, fv2 = v2 * 100000000;
    //std::cout<<"1e8 = "<<1e8<<std::endl;
    std::cout<<"fv1 = "<<fv1<<", fv2 = "<<fv2<<std::endl;
    std::string sfv1 = std::to_string(fv1);
    BigIntegerPlainText plaintext1(sfv1);
    std::cout<<"Plaintext float multiplication result is: "<<v1 * v2<<std:
    auto plain1 = make_shared<BigIntegerPlainText>(plaintext1);
    auto cipher1 = encryptor.encrypt(plain1, r);
    biginteger plain2 = fv2;
    auto multiCipher = encryptor.multByConst(cipher1.get(), plain2);
    shared_ptr<Plaintext> multiPlain = encryptor.decrypt(multiCipher.get()
    auto decryptedPlaintext = *(dynamic_cast<BigIntegerPlainText*>(multiPl
    std::cout<<"Decrypted plaintext integer is: "<<decryptedPlaintext.getX
    biginteger decryptedInteger = decryptedPlaintext.getX();
    long mult = decryptedInteger.convert_to<long>();
    float multFloat = mult * 0.00000001 * 0.00000001;
    std::cout<<"Decrypted plaintext multiplication is: "<<multFloat<<std::
}

int main(int argc, char* argv[]) {
    test_enc_and_dec();
    test_add_and_mult();
    test_fixed_precision_add_and_mult();
}
```