

## 2주차\_ 터미널에서 seal example 코드 컴파일 및 ckks, bfv 예제코드 분석

작성자 : 20200252 김주현

### 👉 터미널에서 example 코드 컴파일

1. 터미널에에서 cd SEAL -> cd native -> cd examples
2. examples 디렉토리 내의 cpp 파일들을 확인할 수 있음.

```
joohyun@gimjuhyeon-ui-MacBookAir examples % ls
1_bfv_basics.cpp          5_ckks_basics.cpp      CMakeLists.txt
2_encoders.cpp           6_rotation.cpp         build
3_levels.cpp             7_serialization.cpp    examples.cpp
4_bgv_basics.cpp         8_performance.cpp      examples.h
```

이 중 5번과 6번 소스코드를 집중적으로 분석

3. 터미널에 cd ../.. 입력해서 SEAL 폴더로 이동
4. cmake -S . -B build -DSEAL\_BUILD\_EXAMPLES=ON 입력
5. cmake --build build

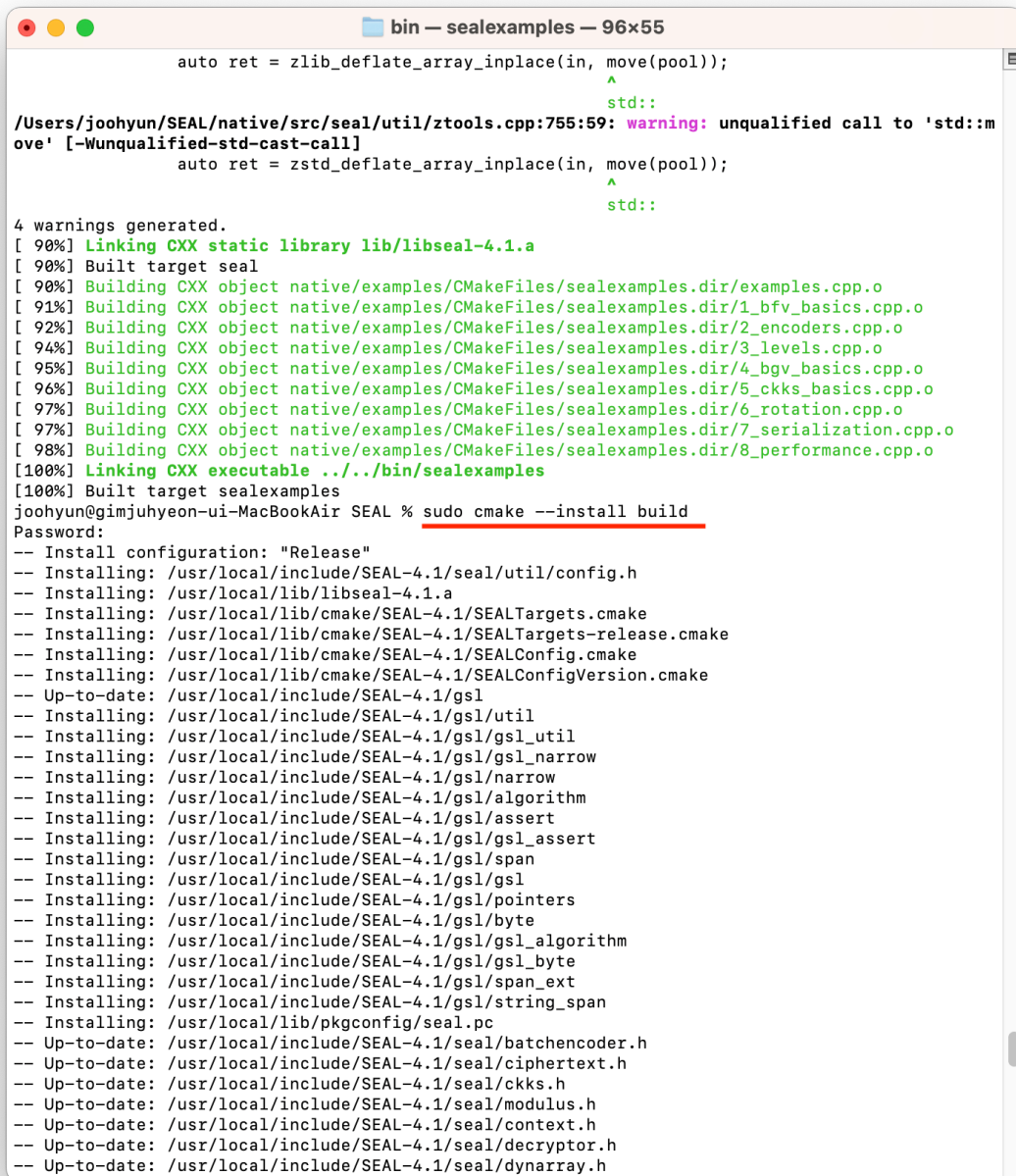


```
bin - sealexamples - 96x55
LICENSE
Makefile
cmake
cmake.install.cmake
joohyun@gimjuhyeon-ui-MacBookAir SEAL % cmake -S . -B build -DSEAL_BUILD_EXAMPLES=ON
-- Build type (CMAKE_BUILD_TYPE): Release
-- Microsoft SEAL debug mode: OFF
-- SEAL_USE_CXX17: ON
-- SEAL_BUILD_DEPS: ON
-- SEAL_USE_MSGSL: ON
-- Microsoft GSL: download ...
-- SEAL_USE_ZLIB: ON
-- ZLIB: download ...
CMake Deprecation Warning at build/thirdparty/zlib-src/CMakeLists.txt:1 (cmake_minimum_required)
:
Compatibility with CMake < 2.8.12 will be removed from a future version of
CMake.

Update the VERSION argument <min> value or use a ...<max> suffix to tell
CMake that the project does not need compatibility with older versions.

-- SEAL_USE_ZSTD: ON
-- Zstandard: download ...
-- ZSTD VERSION: 1.5.2
-- CMAKE_INSTALL_PREFIX: /usr/local
-- CMAKE_INSTALL_LIBDIR: lib
-- ZSTD_LEGACY_SUPPORT not defined!
-- ZSTD_MULTITHREAD_SUPPORT is disabled
-- SEAL_USE_INTEL_HEXL: OFF
-- BUILD_SHARED_LIBS: OFF
-- SEAL_THROW_ON_TRANSPARENT_CIPHERTEXT: ON
-- SEAL_USE_GAUSSIAN_NOISE: OFF
-- SEAL_DEFAULT_PRNG: Blake2xb
-- SEAL_AVOID_BRANCHING: OFF
-- arm_neon.h - found
-- SEAL_USE_INTRIN: ON
-- SEAL_USE_MEMSET_S: ON
-- SEAL_USE_EXPLICIT_BZERO: OFF
-- SEAL_USE_EXPLICIT_MEMSET: OFF
-- SEAL_BUILD_SEAL_C: OFF
-- SEAL_BUILD_EXAMPLES: ON
-- SEAL_BUILD_TESTS: OFF
-- SEAL_BUILD_BENCH: OFF
-- Configuring done (0.7s)
-- Generating done (0.0s)
-- Build files have been written to: /Users/joohyun/SEAL/build
joohyun@gimjuhyeon-ui-MacBookAir SEAL % cmake --build build
[ 1%] Building C object thirdparty/zstd-build/lib/CMakeFiles/libzstd_static.dir/__/__/lib/co
mmon/debug.c.o
[ 2%] Building C object thirdparty/zstd-build/lib/CMakeFiles/libzstd_static.dir/__/__/lib/co
mmon/entropy_common.c.o
[ 3%] Building C object thirdparty/zstd-build/lib/CMakeFiles/libzstd_static.dir/__/__/lib/co
mmon/error_private.c.o
[ 4%] Building C object thirdparty/zstd-build/lib/CMakeFiles/libzstd_static.dir/__/__/lib/co
mmon/fse_decompress.c.o
[ 4%] Building C object thirdparty/zstd-build/lib/CMakeFiles/libzstd_static.dir/__/__/lib/co
```

## 6. sudo make —install build 입력 후 비밀번호 입력



```
auto ret = zlib_deflate_array_inplace(in, move(pool));
std::
/Users/joohyun/SEAL/native/src/seal/util/ztools.cpp:755:59: warning: unqualified call to 'std::move' [-Wunqualified-std-cast-call]
    auto ret = zstd_deflate_array_inplace(in, move(pool));
std::

4 warnings generated.
[ 90%] Linking CXX static library lib/libseal-4.1.a
[ 90%] Built target seal
[ 90%] Building CXX object native/examples/CMakeFiles/sealexamples.dir/examples.cpp.o
[ 91%] Building CXX object native/examples/CMakeFiles/sealexamples.dir/1_bfv_basics.cpp.o
[ 92%] Building CXX object native/examples/CMakeFiles/sealexamples.dir/2_encoders.cpp.o
[ 94%] Building CXX object native/examples/CMakeFiles/sealexamples.dir/3_levels.cpp.o
[ 95%] Building CXX object native/examples/CMakeFiles/sealexamples.dir/4_bgv_basics.cpp.o
[ 96%] Building CXX object native/examples/CMakeFiles/sealexamples.dir/5_ckks_basics.cpp.o
[ 97%] Building CXX object native/examples/CMakeFiles/sealexamples.dir/6_rotation.cpp.o
[ 97%] Building CXX object native/examples/CMakeFiles/sealexamples.dir/7_serialization.cpp.o
[ 98%] Building CXX object native/examples/CMakeFiles/sealexamples.dir/8_performance.cpp.o
[100%] Linking CXX executable ../bin/sealexamples
[100%] Built target sealexamples
joohyun@gimjuhyeon-ui-MacBookAir SEAL % sudo cmake --install build
Password:
-- Install configuration: "Release"
-- Installing: /usr/local/include/SEAL-4.1/seal/util/config.h
-- Installing: /usr/local/lib/libseal-4.1.a
-- Installing: /usr/local/lib/cmake/SEAL-4.1/SEALTargets.cmake
-- Installing: /usr/local/lib/cmake/SEAL-4.1/SEALTargets-release.cmake
-- Installing: /usr/local/lib/cmake/SEAL-4.1/SEALConfig.cmake
-- Installing: /usr/local/lib/cmake/SEAL-4.1/SEALConfigVersion.cmake
-- Up-to-date: /usr/local/include/SEAL-4.1/gsl
-- Installing: /usr/local/include/SEAL-4.1/gsl/util
-- Installing: /usr/local/include/SEAL-4.1/gsl/gsl_util
-- Installing: /usr/local/include/SEAL-4.1/gsl/gsl_narrow
-- Installing: /usr/local/include/SEAL-4.1/gsl/narrow
-- Installing: /usr/local/include/SEAL-4.1/gsl/algorithm
-- Installing: /usr/local/include/SEAL-4.1/gsl/assert
-- Installing: /usr/local/include/SEAL-4.1/gsl/gsl_assert
-- Installing: /usr/local/include/SEAL-4.1/gsl/span
-- Installing: /usr/local/include/SEAL-4.1/gsl/gsl
-- Installing: /usr/local/include/SEAL-4.1/gsl/pointers
-- Installing: /usr/local/include/SEAL-4.1/gsl/byte
-- Installing: /usr/local/include/SEAL-4.1/gsl/gsl_algorithm
-- Installing: /usr/local/include/SEAL-4.1/gsl/gsl_byte
-- Installing: /usr/local/include/SEAL-4.1/gsl/span_ext
-- Installing: /usr/local/include/SEAL-4.1/gsl/string_span
-- Installing: /usr/local/lib/pkgconfig/seal.pc
-- Up-to-date: /usr/local/include/SEAL-4.1/seal/batchencoder.h
-- Up-to-date: /usr/local/include/SEAL-4.1/seal/ciphertext.h
-- Up-to-date: /usr/local/include/SEAL-4.1/seal/ckks.h
-- Up-to-date: /usr/local/include/SEAL-4.1/seal/modulus.h
-- Up-to-date: /usr/local/include/SEAL-4.1/seal/context.h
-- Up-to-date: /usr/local/include/SEAL-4.1/seal/decryptor.h
-- Up-to-date: /usr/local/include/SEAL-4.1/seal/dynarray.h
```

7. `cd build -> cd bin -> ./sealexamples`

을 치면 컴파일할 수 있는 파일들이 나타나게 되고 실행하고자 하는 Examples의 번호를 입력하면 해당 파일을 컴파일할 수 있다.

```
joohyun@gimjuhyeon-ui-MacBookAir SEAL % ls
CHANGES.md      NOTICE          dotnet
CMakeCache.txt   README.md       install_manifest.txt
CMakeFiles       SEALDemo        lib
CMakeLists.txt   SECURITY.md      native
CODE_OF_CONDUCT.md  android        pipelines
CONTRIBUTING.md  bin             pkgconfig
ISSUES.md         build           thirdparty
LICENSE          cmake           tools
Makefile         cmake_install.cmake

joohyun@gimjuhyeon-ui-MacBookAir SEAL % cd build
joohyun@gimjuhyeon-ui-MacBookAir build % ls
CMakeCache.txt   cmake           lib
CMakeFiles       cmake_install.cmake  native
Makefile         dotnet          pkgconfig
bin              install_manifest.txt  thirdparty

joohyun@gimjuhyeon-ui-MacBookAir build % cd bin
joohyun@gimjuhyeon-ui-MacBookAir bin % ls
sealexamples
joohyun@gimjuhyeon-ui-MacBookAir bin % ./sealexamples
Microsoft SEAL version: 4.1.1
+-----+
| The following examples should be executed while reading |
| comments in associated files in native/examples/.      |
+-----+
| Examples          | Source Files    |
+-----+-----+
| 1. BFV Basics    | 1_bfv_basics.cpp |
| 2. Encoders       | 2_encoders.cpp   |
| 3. Levels         | 3_levels.cpp     |
| 4. BGV Basics    | 4_bgv_basics.cpp |
| 5. CKKS Basics   | 5_ckks_basics.cpp |
| 6. Rotation       | 6_rotation.cpp   |
| 7. Serialization  | 7_serialization.cpp |
| 8. Performance Test | 8_performance.cpp |
+-----+-----+
[      0 MB] Total allocation from the memory pool
```

8. 7번까지 완료된 후에는 다음과 같이 SEAL -> build -> bin 디렉토리로 이동 후 `./sealexamples`만 치면 컴파일 할 수 있다.

```

bin — sealexamples — 80x24
LICENSE          cmake          tools
Makefile         cmake_install.cmake
[joohyun@gimjuhyeon-ui-MacBookAir SEAL % cd build
[joohyun@gimjuhyeon-ui-MacBookAir build % cd bin
[joohyun@gimjuhyeon-ui-MacBookAir bin % ./sealexamples
Microsoft SEAL version: 4.1.1

+-----+
| The following examples should be executed while reading |
| comments in associated files in native/examples/.      |
+-----+
| Examples          | Source Files      |
+-----+-----+
| 1. BFV Basics    | 1_bfv_basics.cpp  |
| 2. Encoders       | 2_encoders.cpp    |
| 3. Levels         | 3_levels.cpp       |
| 4. BGV Basics    | 4_bgv_basics.cpp  |
| 5. CKKS Basics   | 5_ckks_basics.cpp |
| 6. Rotation       | 6_rotation.cpp     |
| 7. Serialization  | 7_serialization.cpp |
| 8. Performance Test | 8_performance.cpp |
+-----+-----+

[      0 MB] Total allocation from the memory pool

> Run example (1 ~ 8) or exit (0):

```

아래는 5를 입력해 5\_ckks\_basics.cpp 를 컴파일한 결과의 일부이다.

```

| 7. Serialization    | 7_serialization.cpp |
| 8. Performance Test | 8_performance.cpp   |
+-----+-----+
[      0 MB] Total allocation from the memory pool

> Run example (1 ~ 8) or exit (0):
5

+-----+
|           Example: CKKS Basics           |
+-----+
/
| Encryption parameters :
|   scheme: CKKS
|   poly_modulus_degree: 8192
|   coeff_modulus size: 200 (60 + 40 + 40 + 60) bits
\

Number of slots: 4096
Input vector:

[ 0.0000000, 0.0002442, 0.0004884, ..., 0.9995116, 0.9997558, 1.0000000 ]

Evaluating polynomial  $PI \cdot x^3 + 0.4x + 1 \dots$ 
Line 129 --> Encode input vectors.
Line 140 --> Compute  $x^2$  and relinearize:
+ Scale of  $x^2$  before rescale: 80 bits
Line 152 --> Rescale  $x^2$ .
+ Scale of  $x^2$  after rescale: 40 bits
Line 165 --> Compute and rescale  $PI \cdot x$ .
+ Scale of  $PI \cdot x$  before rescale: 80 bits
+ Scale of  $PI \cdot x$  after rescale: 40 bits
Line 180 --> Compute, relinearize, and rescale  $(PI \cdot x) \cdot x^2$ .
+ Scale of  $PI \cdot x^3$  before rescale: 80 bits

```

## 5\_ckks\_basics.cpp 코드분석

### 👉 동형암호

동형암호란 암호화된 데이터를 복호화 없이 연산할 수 있는 암호 기술로, 동형은 정보를 암호화한 상태에서 각종 연산을 했을 때 그 결과가 암호화하지 않은 상태의 연산 결과와 동일하다는 의미에서 붙여진 것이다. 정보가 암호화돼 있는 상태이기 때문에 해커가 정보를 탈취해도 원천 정보가 노출되지 않아 빅데이터, 자율주행자동차, 사물인터넷 등의 분야에서 주목받고 있는 획기적인 암호 기술이다.

### 👉 CKKS 스키마

동형암호를 연산할 수 있는 알고리즘으로, 암호화된 데이터에 대한 계산을 효율적으로 수행할 수 있도록 설계되어 있다.

### 👉 5\_ckks\_basics.cpp 코드 요약

해당 코드는 암호화된 부동 소수점 데이터  $x$ 에 대해 다항 함수  $P(x) = x^3 + 0.4x + 1$ 를 계산한다.

본 예제에서는 ckks 스키마를 사용하여 다항 함수를 암호화된 데이터에 대해 계산된 결과를 보여준다.

곱셈연산은 암호문의 scale을 증가시킨다.

암호문의 스케일이 coeff\_modulus의 총 크기와 너무 가까워지면 공간이 부족해진다.

따라서 ckks 스키마는 rescale 기능을 제공해 스케일을 줄여 안정화시킨다.

### 👉 코드 분석

예제 실행 전에 ckks 스키마 설정 단계부터 시작한다.

```
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 }));
//CoeffModulus::Create 함수로 적절한 크기의 소수를 생성.
//coeff_modulus의 선택은 암호 연산의 효율성과 결과의 정확성에 영향을 미칩니다.
```

Poly\_modulus\_degree는 8192로 설정되어 있으며, 암호화된 데이터에 대한 연산에서 다항식의 차수를 제한하는 역할을 한다.

예시에서 60비트, 40비트, 40비트, 60비트 크기의 소수를 생성하여 coeff\_modulus에 저장하고 있다. 이 소수들은 스케일 조절과 암호 연산에 사용된다.

```
double scale = pow(2.0, 40);
```

이 코드에서는 초기 스케일을 2의 40승으로 설정한다.

Scale : 연산에서 사용되는 암호화된 데이터의 정밀도를 의미

```
KeyGenerator keygen(context);
auto secret_key = keygen.secret_key();
PublicKey public_key;
keygen.create_public_key(public_key);
RelinKeys relin_keys;
keygen.create_relin_keys(relin_keys);
GaloisKeys gal_keys;
keygen.create_galois_keys(gal_keys);
Encryptor encryptor(context, public_key);
Evaluator evaluator(context);
Decryptor decryptor(context, secret_key);
```

- 👉 KeyGenerator 객체 : 암호화 및 복호화에 사용될 secret\_key와 public\_key 생성
- 👉 Encryptor : 암호화를 위한 객체
- 👉 Evaluator : 연산을 위한 객체
- 👉 Decryptor : 복호화를 위한 객체

```
CKKSEncoder encoder(context);
size_t slot_count = encoder.slot_count();
cout << "Number of slots: " << slot_count << endl;
```

- 👉 CKKSEncoder : 부동 소수점 데이터를 암호화하기 위한 인코딩을 수행하는 객체
- 👉 slot\_count : 인코딩할 수 있는 슬롯의 개수

👉 계산에 사용될 암호화된 데이터 준비

```
cout << "Evaluating polynomial  $PI \cdot x^3 + 0.4x + 1$  ..." << endl;

/*
We create plaintexts for PI, 0.4, and 1 using an overload of CKKSEncoder::encode
that encodes the given floating-point value to every slot in the vector.
*/
Plaintext plain_coeff3, plain_coeff1, plain_coeff0;
encoder.encode(3.14159265, scale, plain_coeff3);
encoder.encode(0.4, scale, plain_coeff1);
encoder.encode(1.0, scale, plain_coeff0);
```

위의 코드는  $PI \cdot x^3 + 0.4x + 1$  다항식 계산을 위해 PI와 0.4, 1에 대한 plaintext를 생성하는 부분이다.

👉 plaintext(평문)이란?

암호화되지 않은 원본 데이터

👉 Plaintext 타입 변수들

👉 plain\_coeff3 :  $x^3$ 의 계수인 PI(3.14159265)를 인코딩

👉 plain\_coeff2 :  $x^2$ 의 계수인 0.4를 인코딩

👉 plain\_coeff1 : 실수항 1을 인코딩

🦊 Encoder.encode 함수를 사용해 주어진 부동소수점 값을 각 슬롯에 인코딩하여 plaintext로 변환

🦊 scale은 인코딩된 값의 정밀도 조절을 위해 사용됨.

👉  $x^3$ 을 계산하기 위해  $x^2$ 을 먼저 계산하고 재선형화(relinearize)

```
/*
To compute  $x^3$  we first compute  $x^2$  and relinearize. However, the scale has
now grown to  $2^{80}$ .
*/
Ciphertext x3_encrypted;
print_line(__LINE__);
cout << "Compute  $x^2$  and relinearize:" << endl;
evaluator.square(x1_encrypted, x3_encrypted);
evaluator.relinearize_inplace(x3_encrypted, relin_keys);
cout << "    + Scale of  $x^2$  before rescale: " << log2(x3_encrypted.scale()) << " bits" << endl;
```

👉 X3\_encrypted : x1\_encrypted의 제곱을 저장 (Ciphertext 타입)

X3\_encrypted를 재선형화

\*재선형화 : 암호문의 크기를 최적화



위와 같은 방식으로  $PI \cdot x^3$ ,  $0.4 \cdot x$ 를 구하는 작업이 수행된다

👉 5\_ckks\_basics.cpp 컴파일 결과

```
+-----+
|               Example: CKKS Basics               |
+-----+
/
| Encryption parameters :
|   scheme: CKKS
|   poly_modulus_degree: 8192
|   coeff_modulus size: 200 (60 + 40 + 40 + 60) bits
\
```

Number of slots: 4096

👉 슬롯 개수 : 4096

암호문에 포함된 슬롯의 개수가 4096개라는 것.

동형암호에서 처리 가능한 데이터의 크기를 나타냄.

Input vector:

```
[ 0.0000000, 0.0002442, 0.0004884, ..., 0.9995116, 0.9997558, 1.0000000 ]
```

Evaluating polynomial  $PI \cdot x^3 + 0.4x + 1 \dots$

Line 129 --> Encode input vectors.

Line 140 --> Compute  $x^2$  and relinearize:

+ Scale of  $x^2$  before rescale: 80 bits

Line 152 --> Rescale  $x^2$ .

+ Scale of  $x^2$  after rescale: 40 bits

Line 165 --> Compute and rescale  $PI \cdot x$ .

+ Scale of  $PI \cdot x$  before rescale: 80 bits

+ Scale of  $PI \cdot x$  after rescale: 40 bits

Line 180 --> Compute, relinearize, and rescale  $(PI \cdot x) \cdot x^2$ .

+ Scale of  $PI \cdot x^3$  before rescale: 80 bits

+ Scale of  $PI \cdot x^3$  after rescale: 40 bits

Line 192 --> Compute and rescale  $0.4 \cdot x$ .

+ Scale of  $0.4 \cdot x$  before rescale: 80 bits

+ Scale of  $0.4 \cdot x$  after rescale: 40 bits

👉 Input vector

입력 벡터는 0부터 1 사이의 값으로 이루어져 있으며, 4096개의 슬롯에 순서대로 할당되어 있음.

👉 입력된 벡터 [ 0.0000000, 0.0002442, 0.0004884, ..., 0.9995116, 0.9997558, 1.0000000 ]  
를 인코딩

👉  $x$ 로  $x^2$ 을 계산하고, 결과를 재선형화 ->  $x^2$ 의 스케일을 40비트로 조정(rescale)



👉  $Pi * X$

$Pi$ 와  $x$  를 곱해서  $pi*x$ 를 계산하고 계산 결과의 scale을 40bit로 조정

👉  $PI * X^3$

$PI * X^3$  즉,  $(PI * X) * X^2$ 을 계산 -> 재선형화 -> scale을 40bit로 조정

👉  $0.4X$

0.4와  $X$ 를 곱해서  $0.4 * X$ 를 계산 -> 재선형화 -> scale을 40bit로 조정

```
Line 209 --> Parameters used by all three terms are different.  
+ Modulus chain index for x3_encrypted: 0  
+ Modulus chain index for x1_encrypted: 1  
+ Modulus chain index for plain_coeff0: 2
```

```
Line 237 --> The exact scales of all three terms are different:  
+ Exact scale in  $PI * x^3$ : 1099512659965.7514648438  
+ Exact scale in  $0.4 * x$ : 1099511775231.0197753906  
+ Exact scale in  $1$ : 1099511627776.0000000000
```

👉 세 개의 항  $pi * x^3$ ,  $0.4 * x$ ,  $1$ 이 서로 다른 암호 파라미터를 사용하고 있다.

👉 세 개의 항  $pi * x^3$ ,  $0.4 * x$ ,  $1$ 의 정확한 스케일 값이 서로 다르다.

암호 파라미터와 스케일 값이 서로 다르면  $PI * x^3 + 0.4 * x + 1$  값을 계산하기 위한 더하기 연산을 수행할 수 없다. 따라서 스케일 값을 통일해줘야 한다.

```
Line 262 --> Normalize scales to  $2^{40}$ .  
Line 273 --> Normalize encryption parameters to the lowest level.  
Line 282 --> Compute  $PI * x^3 + 0.4 * x + 1$ .  
Line 292 --> Decrypt and decode  $PI * x^3 + 0.4 * x + 1$ .
```

👉 스케일 값을  $2^{40}$ 으로 정규화

👉 암호화 파라미터를 가장 낮은 레벨로 정규화

👉 세 항을 더해서  $PI * x^3 + 0.4 * x + 1$  을 계산

👉 계산 결과를 복호화 -> 디코딩해서 원래 값으로 변환

```

/*
First print the true result.
*/
Plaintext plain_result;
print_line(__LINE__);
cout << "Decrypt and decode  $\text{PI} \cdot x^3 + 0.4x + 1.$ " << endl;
cout << "      + Expected result:" << endl;
vector<double> true_result;
for (size_t i = 0; i < input.size(); i++)
{
    double x = input[i];
    true_result.push_back((3.14159265 * x * x + 0.4) * x + 1);
}
print_vector(true_result, 3, 7);

```

☞ 입력된 벡터 [ 0.0000000, 0.0002442, 0.0004884, ..., 0.9995116, 0.9997558, 1.0000000 ]  
를  $\text{PI} \cdot x^3 + 0.4x + 1$ 에 넣어서 실제로 계산했을때 출력되는 값들. 즉 기댓값(Expected  
result). 암호연산 후 복호화한 값과 비교해보기 위한 값

+ Expected result:

[ 1.0000000, 1.0000977, 1.0001954, ..., 4.5367965, 4.5391940, 4.5415926 ]

+ Computed result ..... Correct.

[ 1.0000000, 1.0000977, 1.0001954, ..., 4.5367995, 4.5391970, 4.5415956 ]

☞ 계산된 결과가 예상 결과와 일치함

☞ 다항식  $\text{PI} \cdot x^3 + 0.4x + 1$ 을 입력된 벡터에 대해 동형암호를 사용해서 연산한 후 복  
호화한 값이 예상 결과와 일치하는 것을 확인할 수 있다.

☞ 6\_rotation.cpp 코드 분석

```

void example_rotation()
{
    print_example_banner("Example: Rotation");

    /*
    Run all rotation examples.
    */
    example_rotation_bfv();
    example_rotation_ckks();
}

```

#### 👉 example\_rotatation\_bfv 함수

bfv 암호화 스키마를 사용해 회전 연산을 수행

\*bfv에서의 회전 : 행을 왼쪽 또는 오른쪽으로 이동하거나 열을 교환

#### 👉 example\_rotatation\_ckks 함수

ckks 암호화 스키마를 사용해 회전연산 수행

주어진 벡터를 ckks 스키마에 인코딩해서 암호화한 뒤, 벡터를 회전시킨 결과를 출력

\*ckks 에서의 회전 : 벡터의 원소를 왼쪽 또는 오른쪽으로 이동

#### 👉 BFV (Brakerski-Fan-Vercauteren)

ckks와 마찬가지로, 암호화된 상태에서도 계산을 수행할 수 있는 기능 제공.

ckks가 실수 계산에 적절하다면, bfv는 주로 정수 계산에 사용됨.

회전 기능 제공

회전 : 암호문의 데이터를 주기적으로 이동시키는 연산

#### 👉 bfv 예제

```
void example_rotatation_bfv()
{
    print_example_banner("Example: Rotation / Rotation in BFV");

    EncryptionParameters parms(scheme_type::bfv);

    size_t poly_modulus_degree = 8192;
    parms.set_poly_modulus_degree(poly_modulus_degree);
    parms.set_coeff_modulus(CoeffModulus::BFVDefault(poly_modulus_degree));
    parms.set_plain_modulus(PlainModulus::Batching(poly_modulus_degree, 20));

    SEALContext context(parms);
    print_parameters(context);
    cout << endl;
```

> Run example (1 ~ 8) or exit (0): 6

```
+-----+
|           Example: Rotation           |
+-----+
```

```
+-----+
|           Example: Rotation / Rotation in BFV           |
+-----+
```

```
/
| Encryption parameters :
|   scheme: BFV
|   poly_modulus_degree: 8192
|   coeff_modulus size: 218 (43 + 43 + 44 + 44 + 44) bits
|   plain_modulus: 1032193
\
```

👉 암호화 파라미터 설정

```
/*
First we use BatchEncoder to encode the matrix into a plaintext. We encrypt
the plaintext as usual.
*/
Plaintext plain_matrix;
print_line(__LINE__);
cout << "Encode and encrypt." << endl;
batch_encoder.encode(pod_matrix, plain_matrix);
Ciphertext encrypted_matrix;
encryptor.encrypt(plain_matrix, encrypted_matrix);
cout << "    + Noise budget in fresh encryption: " << decryptor.invariant_noise_budget(encrypted_matrix) << " bits"
    << endl;
cout << endl;
```

Plaintext matrix row size: 4096

Input plaintext matrix:

```
[ 0, 1, 2, 3, 0, ..., 0, 0, 0, 0, 0 ]
[ 4, 5, 6, 7, 0, ..., 0, 0, 0, 0, 0 ]
```

Line 65 --> Encode and encrypt.

+ Noise budget in fresh encryption: 146 bits

👉 BatchEncoder를 사용해 평문 행렬을 인코딩한 후, 평문을 암호화

```

/*
Now rotate both matrix rows 3 steps to the left, decrypt, decode, and print.
*/
print_line(__LINE__);
cout << "Rotate rows 3 steps left." << endl;
evaluator.rotate_rows_inplace(encrypted_matrix, 3, galois_keys);
Plaintext plain_result;
cout << "    + Noise budget after rotation: " << decryptor.invariant_noise_budget(encrypted_matrix) << " bits"
    << endl;
cout << "    + Decrypt and decode ..... Correct." << endl;
decryptor.decrypt(encrypted_matrix, plain_result);
batch_encoder.decode(plain_result, pod_matrix);
print_matrix(pod_matrix, row_size);

```

```

[ 0, 1, 2, 3, 0, ..., 0, 0, 0, 0, 0 ]
[ 4, 5, 6, 7, 0, ..., 0, 0, 0, 0, 0 ]

```

회전 전

```

Line 84 --> Rotate rows 3 steps left.
    + Noise budget after rotation: 142 bits
    + Decrypt and decode ..... Correct.

```

```

[ 3, 0, 0, 0, 0, ..., 0, 0, 0, 1, 2 ]
[ 7, 0, 0, 0, 0, ..., 0, 0, 4, 5, 6 ]

```

회전 후

👉 암호화된 상태로 회전 연산을 수행하고 결과를 해독하여 회전 결과를 출력.  
위 예시에서는 행을 3칸씩 왼쪽으로 회전.

```

Line 84 --> Rotate rows 3 steps left.
    + Noise budget after rotation: 142 bits
    + Decrypt and decode ..... Correct.

```

```

[ 3, 0, 0, 0, 0, ..., 0, 0, 0, 1, 2 ]
[ 7, 0, 0, 0, 0, ..., 0, 0, 4, 5, 6 ]

```

회전 전

```

Line 98 --> Rotate columns.
    + Noise budget after rotation: 142 bits
    + Decrypt and decode ..... Correct.

```

```

[ 7, 0, 0, 0, 0, ..., 0, 0, 4, 5, 6 ]
[ 3, 0, 0, 0, 0, ..., 0, 0, 0, 1, 2 ]

```

회전 후

- 👉 마찬가지로 암호화된 상태의 행렬에서 회전한 후 결과를 해독하여 출력한다.  
위 예시에서는 열을 회전시키는 연산을 진행. (1행과 2행이 swap됨)

```
[ 7, 0, 0, 0, 0, ..., 0, 0, 4, 5, 6 ]  
[ 3, 0, 0, 0, 0, ..., 0, 0, 0, 1, 2 ]
```

회전 전

```
Line 111 --> Rotate rows 4 steps right.  
+ Noise budget after rotation: 142 bits  
+ Decrypt and decode ..... Correct.
```

```
[ 0, 4, 5, 6, 7, ..., 0, 0, 0, 0, 0 ]  
[ 0, 0, 1, 2, 3, ..., 0, 0, 0, 0, 0 ]
```

회전 후

- 👉 마지막 행을 4단계 오른쪽으로 회전.

👉 ckks 예제

입력 벡터를 2단계 왼쪽으로 회전

```

+-----+
|           Example: Rotation / Rotation in CKKS           |
+-----+
/
| Encryption parameters :
|   scheme: CKKS
|   poly_modulus_degree: 8192
|   coeff_modulus size: 200 (40 + 40 + 40 + 40 + 40) bits
\

```

Number of slots: 4096

Input vector:

```
[ 0.0000000, 0.0002442, 0.0004884, ..., 0.9995116, 0.9997558, 1.0000000 ]
```

Line 176 --> Encode and encrypt.

Line 184 --> Rotate 2 steps left.

+ Decrypt and decode ..... Correct.

```
[ 0.0004884, 0.0007326, 0.0009768, ..., 1.0000000, -0.0000000, 0.0002442 ]
```

- 👉 ckks 스키마에 대한 암호화 파라미터를 설정
- 👉 ckksEncoder를 사용하여 입력 벡터를 인코딩
- 👉 인코딩 된 벡터를 암호화
- 👉 회전 연산 수행 (왼쪽으로 2칸씩 회전)
- 👉 회전된 벡터를 출력

👉 이해가 잘 되지 않았던 점

- 👉 Slot과 slot의 개수는 어떤 것을 의미하는지 이해가 잘 가지 않는다.
- 👉 5\_ckks\_basics.cpp 에서 Poly\_modulus\_degree 를 8192로 설정하는 부분이 있는데, Poly\_modulus\_degree가 무엇인지 와닿지 않는다.