**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.

텍스트, 폰트, 영수증, 화이트이(가) 표시된 사진

자동 생성된 설명

이 중 5번과 6번 소스코드를 집중적으로 분석

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

텍스트, 전자제품, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. sudo make —install build 입력 후 비밀번호 입력

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. cd build -> cd bin -> ./sealexamples

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

텍스트, 스크린샷, 메뉴, 폰트이(가) 표시된 사진

자동 생성된 설명

1. 7번까지 완료된 후에는 다음과 같이 SEAL -> build -> bin 디렉토리로 이동 후 ./sealexamples만 치면 컴파일 할 수 있다.

텍스트, 스크린샷, 소프트웨어, 번호이(가) 표시된 사진

자동 생성된 설명

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

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명

**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 x에 대해 다항 함수 PI\*x^3 + 0.4\*x + 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.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

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.



이 코드에서는 초기 스케일을 2의 40승으로 설정한다.

Scale : 연산에서 사용되는 암호화된 데이터의 정밀도를 의미

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

👉 KeyGenerator 객체 : 암호화 및 복호화에 사용될 secret\_key와 public\_key 생성

👉 Encryptor : 암호화를 위한 객체

👉 Evaluator : 연산을 위한 객체

👉 Decryptor : 복호화를 위한 객체

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

👉 CKKSEncoder : 부동 소수점 데이터를 암호화하기 위한 인코딩을 수행하는 객체

👉 slot\_count : 인코딩할 수 있는 슬롯의 개수

👉 계산에 사용될 암호화된 데이터 준비

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

위의 코드는 PI\*x^3 + 0.4\*x + 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^3의 계수인 PI(3.14159265)를 인코딩

👉 plain\_coeff2 : x^2의 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^3을 계산하기 위해 x^2을 먼저 계산하고 재선형화(relinearize)

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

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

X3\_encrypted를 재선형화

\*재선형화 : 암호문의 크기를 최적화

위와 같은 방식으로 PI\*x^3, 0.4\*x를 구하는 작업이 수행된다

.

✍️ 5\_ckks\_basics.cpp 컴파일 결과

텍스트, 폰트, 스크린샷, 영수증이(가) 표시된 사진

자동 생성된 설명

👉 슬롯 개수 : 4096

암호문에 포함된 슬롯의 개수가 4096개라는 것.

동형암호에서 처리 가능한 데이터의 크기를 나타냄.

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명

👉 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

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

👉 Pi \* X

Pi와 x 를 곱해서 pi\*x를 계산하고 계산 결과의 scale을 40bit로 조정

👉 PI\*X^3

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

👉 0.4X

0.4와 X를 곱해서 0.4\*X를 계산 -> 재선형화 -> scale을 40bit로 조정

텍스트, 폰트, 스크린샷, 화이트이(가) 표시된 사진

자동 생성된 설명

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

👉 세 개의 항 pI\* x^3, 0.4\*x, 1의 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.

텍스트, 폰트, 화이트, 대수학이(가) 표시된 사진

자동 생성된 설명

👉 스케일 값을 2^40으로 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

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

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

텍스트, 폰트, 영수증, 화이트이(가) 표시된 사진

자동 생성된 설명

👉 계산된 결과가 예상 결과와 일치함

👉 다항식 PI\*x^3 + 0.4\*x + 1을 입력된 벡터에 대해 동형암호를 사용해서 연산한 후 복호화한 값이 예상 결과와 일치하는 것을 확인할 수 있다.

👉 6\_rotation.cpp 코드 분석

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

✍️ example\_ratation\_bfv 함수

bfv 암호화 스키마를 사용해 회전 연산을 수행

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

✍️ example\_rotation\_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)

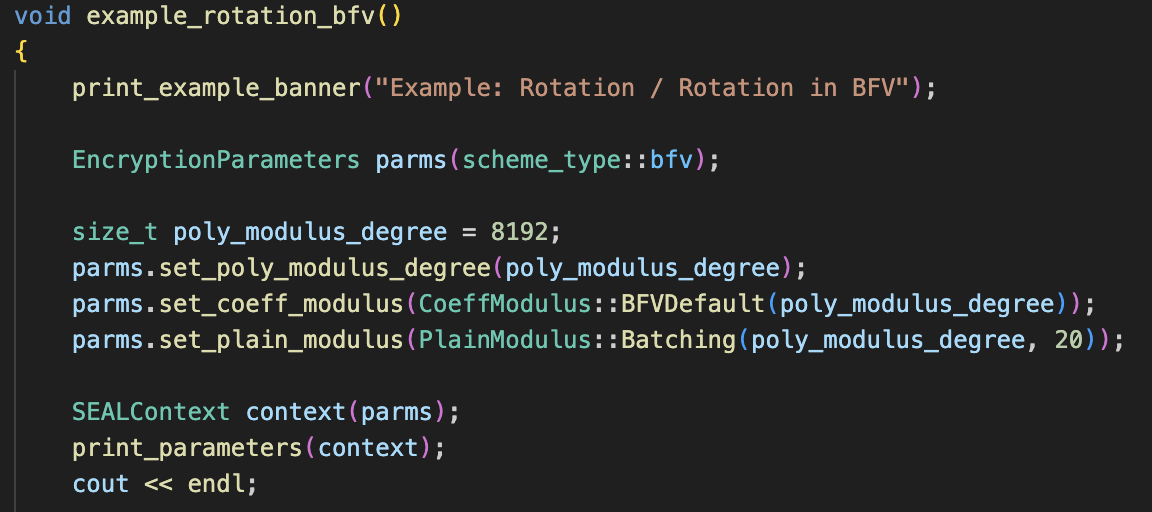
ckks와 마찬가지로, 암호화된 상태에서도 계산을 수행할 수 있는 기능 제공.

ckks가 실수 계산에 적절하다면, bfv는 주로 정수 계산에 사용됨.

회전 기능 제공

회전 : 암호문의 데이터를 주기적으로 이동시키는 연산

✍️ bfv 예제



텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명

👉 암호화 파라미터 설정

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

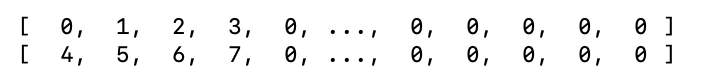
텍스트, 폰트, 스크린샷, 화이트이(가) 표시된 사진

자동 생성된 설명

👉 BatchEncoder를 사용해 평문 행렬을 인코딩한 후, 평문을 암호화

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

 회전 전

텍스트, 폰트, 화이트, 스크린샷이(가) 표시된 사진

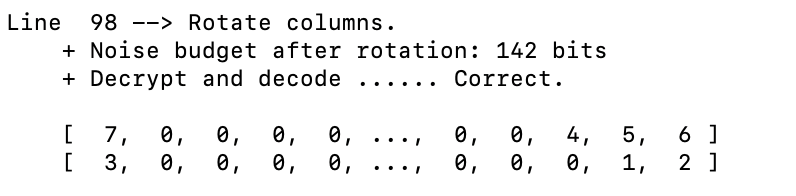
자동 생성된 설명회전 후

👉 암호화된 상태로 회전 연산을 수행하고 결과를 해독하여 회전 결과를 출력.

위 예시에서는 행을 3칸씩 왼쪽으로 회전.

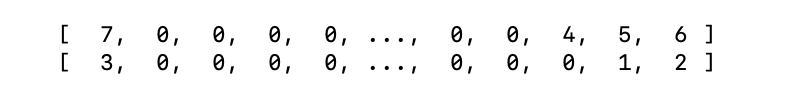
텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명회전 전

회전 후

👉 마찬가지로 암호화된 상태의 행렬에서 회전한 후 결과를 해독하여 출력한다.

위 예시에서는 열을 회전시키는 연산을 진행. (1행과 2행이 swap됨)

회전 전

텍스트, 폰트, 화이트, 스크린샷이(가) 표시된 사진

자동 생성된 설명회전 후

👉 마지막 행을 4단계 오른쪽으로 회전.

✍️ ckks 예제

입력 벡터를 2단계 왼쪽으로 회전

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명

👉 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.