PRE week 4

날짜	@2023년 7월 28일

작성자 : 김주현

OpenFHE 설치

https://openfhe-development.readthedocs.io/en/latest/

macOS 환경에서 설치 및 build

https://openfhe-development.readthedocs.io/en/latest/sphinx_rsts/intro/installation/macos.html

코드 수정 후 build하는 방법

openfhe-development/build 디렉토리에서 `make` 후 실행파일 경로 입력

ex.

```
joohyun@gimjuhyeon-ui-MacBookAir build % bin/examples/pke/week4_task
```

📃 Task

OpenFHE 설치 후 (x+1)^2 * (x^2+1)을 계산 후 2 step 왼쪽으로 rotation하는 코드 작성

- src/pke/exmples/advanced-real-numbers.cpp 참고
- src/pke/exmples/simple-real-numbers.cpp 참고
- 초기 scale = 2^50

```
#define PROFILE

#include "openfhe.h"

using namespace lbcrypto;

void AutomaticRescaleDemo(ScalingTechnique scalTech);
void ManualRescaleDemo(ScalingTechnique scalTech);
void HybridKeySwitchingDemo1();
void HybridKeySwitchingDemo2();
void FastRotationsDemo1();
void FastRotationsDemo2();

int main(int argc, char* argv[]) {

    AutomaticRescaleDemo(FLEXIBLEAUTO);

    AutomaticRescaleDemo(FIXEDAUTO);

    ManualRescaleDemo(FIXEDMANUAL);

    return 0;
}

void AutomaticRescaleDemo(ScalingTechnique scalTech) {

    if (scalTech == FLEXIBLEAUTO) {
        std::cout << std::endl << std::endl << std::endl << " ===== FlexibleAutoDemo ============= " << std::endl;
    }
    else {
        std::cout << std::endl << std::endl << std::endl << " ===== FixedAutoDemo ============= " << std::endl;
```

```
    }

    uint32_t batchSize = 8;
    CCParams<CryptoContextCKKSRNS> parameters;
    parameters.SetMultiplicativeDepth(2);
    parameters.SetScalingModSize(50);
    parameters.SetScalingTechnique(scalTech);
    parameters.SetBatchSize(batchSize);

    CryptoContext<DCRTPoly> cc = GenCryptoContext(parameters);

    std::cout << "CKKS scheme is using ring dimension " << cc->GetRingDimension() << std::endl << std::endl;

    cc->Enable(PKE);
    cc->Enable(KEYSWITCH);
    cc->Enable(LEVELEDSHE);

    auto keys = cc->KeyGen();
    cc->EvalMultKeyGen(keys.secretKey);

    // Input
    std::vector<double> x = {1.0, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07};
    Plaintext ptxt       = cc->MakeCKKSPackedPlaintext(x);

    std::cout << "Input x: " << ptxt << std::endl;

    auto c = cc->Encrypt(ptxt, keys.publicKey);


    //Computing f(x) = (x+1)^2*(x^2+2)

    auto c1     = cc -> EvalAdd(c, 1.0);                    // (x+1)
    auto c1_2   = cc -> EvalMult(c1, c1);                 //(x+1)^2
    auto c2     = cc -> EvalMult(c, c);                   // x^2
    auto c2_pl  = cc -> EvalAdd(c2, 2.0);                  // (x^2+2)
    auto cRes   = cc -> EvalMult(c1_2, c2_pl);           // Final result

    Plaintext result;
    std::cout.precision(8);

    cc->Decrypt(cRes, keys.secretKey, &result);
    result->SetLength(batchSize);
    std::cout << "(x+1)^2*(x^2+2) = " << result << std::endl;


    /* --------- Rotation left 2 -------*/
    uint32_t dnum = 2;
    std::cout << "- Using HYBRID key switching with " << dnum << " digits" << std::endl << std::endl;

    cc->EvalRotateKeyGen(keys.secretKey, {2});

    TimeVar t;
    TIC(t);
    auto cRot2        = cc->EvalRotate(cRes, 2);
    double time2digits = TOC(t);

    Plaintext rotate_result;
    std::cout.precision(8);

    cc->Decrypt(keys.secretKey, cRot2, &rotate_result);
    result->SetLength(batchSize);
    std::cout << "x left rotate by 2 = " << rotate_result << std::endl;
    std::cout << " 2 rotations with HYBRID (2 digits) took " << time2digits << "ms" << std::endl;


}

void ManualRescaleDemo(ScalingTechnique scalTech) {

    std::cout << "\n\n\n ===== FixedManualDemo ============= " << std::endl;

    uint32_t batchSize = 8;
    CCParams<CryptoContextCKKSRNS> parameters;
    parameters.SetMultiplicativeDepth(2);
```

```
    parameters.SetScalingModSize(50);
    parameters.SetBatchSize(batchSize);

    CryptoContext<DCRTPoly> cc = GenCryptoContext(parameters);

    std::cout << "CKKS scheme is using ring dimension " << cc->GetRingDimension() << std::endl << std::endl;

    cc->Enable(PKE);
    cc->Enable(KEYSWITCH);
    cc->Enable(LEVELEDSHE);

    auto keys = cc->KeyGen();
    cc->EvalMultKeyGen(keys.secretKey);

    // Input
    std::vector<double> x = {1.0, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07};
    Plaintext ptxt        = cc->MakeCKKSPackedPlaintext(x);

    std::cout << "Input x: " << ptxt << std::endl;

    auto c = cc->Encrypt(keys.publicKey, ptxt);


    //Computing f(x) = (x+1)^2*(x^2+2)

    // x+1
    auto c_pl_depth2 = cc->EvalAdd(c, 1.0);
    auto c_pl_depth1 = cc->Rescale(c_pl_depth2);
    // (x+1)^2
    auto c_pl_2_depth2 = cc->EvalMult(c_pl_depth1, c_pl_depth1);
    auto c_pl_2_depth1 = cc->Rescale(c_pl_2_depth2);
    // x^2
    auto c2_depth2 = cc->EvalMult(c, c);
    auto c2_depth1 = cc->Rescale(c2_depth2);
     // x^2+2
    auto c2_pl_depth2 = cc->EvalAdd(c2_depth1, 2.0);
    auto c2_pl_depth1 = cc->Rescale(c2_pl_depth2);
    // Final result
    auto cRes_depth2 = cc->EvalMult(c_pl_2_depth1, c2_pl_depth1);
    auto cRes_depth1 = cc->Rescale(cRes_depth2);

    Plaintext result;
    std::cout.precision(8);

    cc->Decrypt(keys.secretKey, cRes_depth1, &result);
    result->SetLength(batchSize);
    std::cout << "(x+1)^2 * (x^2+2) = " << result << std::endl;

    /* --------- Rotation left 2 -------*/
    uint32_t dnum = 2;
    std::cout << "- Using HYBRID key switching with " << dnum << " digits" << std::endl << std::endl;

    cc->EvalRotateKeyGen(keys.secretKey, {2});

    TimeVar t;
    TIC(t);
    auto cRot2         = cc->EvalRotate(cRes_depth1, 2);
    double time2digits = TOC(t);

    Plaintext rotate_result;
    std::cout.precision(8);

    cc->Decrypt(keys.secretKey, cRot2, &rotate_result);
    result->SetLength(batchSize);
    std::cout << "x left rotate by 2 = " << rotate_result << std::endl;
    std::cout << " rotations with HYBRID (2 digits) took " << time2digits << "ms" << std::endl;
}

void HybridKeySwitchingDemo1() {


    std::cout << "\n\n\n ===== HybridKeySwitchingDemo1 ============= " << std::endl;
    /*
   * dnum is the number of large digits in HYBRID decomposition
```

```
*
* If not supplied (or value 0 is supplied), the default value is
* set as follows:
* - If multiplicative depth is > 3, then dnum = 3 digits are used.
* - If multiplicative depth is 3, then dnum = 2 digits are used.
* - If multiplicative depth is < 3, then dnum is set to be equal to
* multDepth+1
*/
 uint32_t dnum = 2;
 /* To understand the effects of changing dnum, it is important to
* understand how the ciphertext modulus size changes during key
* switching.
*
* In our RNS implementation of CKKS, every ciphertext corresponds
* to a large number (which is represented as small integers in RNS)
* modulo a ciphertext modulus Q, which is defined as the product of
* (multDepth+1) prime numbers: Q = q0 * q1 * ... * qL. Each qi is
* selected to be close to the scaling factor D=2^p, hence the total
* size of Q is approximately:
*
* sizeof(Q) = (multDepth+1)*scaleModSize.
*
* HYBRID key switching takes a number d that's defined modulo Q,
* and performs 4 steps:
* 1 - Digit decomposition:
*     Split d into dnum digits - the size of each digit is roughly
*     ceil(sizeof(Q)/dnum)
* 2 - Extend ciphertext modulus from Q to Q*P
*     Here P is a product of special primes
* 3 - Multiply extended component with key switching key
* 4 - Decrease the ciphertext modulus back down to Q
*
* It's not necessary to understand how all these stages work, as
* long as it's clear that the size of the ciphertext modulus is
* increased from sizeof(Q) to sizeof(Q)+sizeof(P) in stage 2. P
* is always set to be as small as possible, as long as sizeof(P)
* is larger than the size of the largest digit, i.e., than
* ceil(sizeof(Q)/dnum). Therefore, the size of P is inversely
* related to the number of digits, so the more digits we have, the
* smaller P has to be.
*
* The tradeoff here is that more digits means that the digit
* decomposition stage becomes more expensive, but the maximum
* size of the ciphertext modulus Q*P becomes smaller. Since
* the size of Q*P determines the necessary ring dimension to
* achieve a certain security level, more digits can in some
* cases mean that we can use smaller ring dimension and get
* better performance overall.
*
* We show this effect with demos HybridKeySwitchingDemo1 and
* HybridKeySwitchingDemo2.
*
*/
 uint32_t batchSize = 8;
 CCParams<CryptoContextCKKSRNS> parameters;
 parameters.SetMultiplicativeDepth(5);
 parameters.SetScalingModSize(50);
 parameters.SetBatchSize(batchSize);
 parameters.SetScalingTechnique(FLEXIBLEAUTO);
 parameters.SetNumLargeDigits(dnum);

 CryptoContext<DCRTPoly> cc = GenCryptoContext(parameters);

 std::cout << "CKKS scheme is using ring dimension " << cc->GetRingDimension() << std::endl;

 std::cout << "- Using HYBRID key switching with " << dnum << " digits" << std::endl << std::endl;

 cc->Enable(PKE);
 cc->Enable(KEYSWITCH);
 cc->Enable(LEVELEDSHE);

 auto keys = cc->KeyGen();
 cc->EvalRotateKeyGen(keys.secretKey, {1, -2});
```

```
    // Input
    std::vector<double> x = {1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7};
    Plaintext ptxt        = cc->MakeCKKSPackedPlaintext(x);

    std::cout << "Input x: " << ptxt << std::endl;

    auto c = cc->Encrypt(keys.publicKey, ptxt);

    TimeVar t;
    TIC(t);
    auto cRot1         = cc->EvalRotate(c, 1);
    auto cRot2         = cc->EvalRotate(cRot1, -2);
    double time2digits = TOC(t);
    // Take note and compare the runtime to the runtime
    // of the same computation in the next demo.

    Plaintext result;
    std::cout.precision(8);

    cc->Decrypt(keys.secretKey, cRot2, &result);
    result->SetLength(batchSize);
    std::cout << "x rotate by -1 = " << result << std::endl;
    std::cout << " - 2 rotations with HYBRID (2 digits) took " << time2digits << "ms" << std::endl;


}

void HybridKeySwitchingDemo2() {
    /*
   * Please refer to comments in HybridKeySwitchingDemo1.
   *
   */

    std::cout << "\n\n\n ===== HybridKeySwitchingDemo2 ============= " << std::endl;

    /*
   * Here we use dnum = 3 digits. Even though 3 digits are
   * more than the two digits in the previous demo and the
   * cost of digit decomposition is higher, the increase in
   * digits means that individual digits are smaller, and we
   * can perform key switching by using only one special
   * prime in P (instead of two in the previous demo).
   *
   * This also means that the maximum size of ciphertext
   * modulus in key switching is smaller by 60 bits, and it
   * turns out that this decrease is adequate to warrant a
   * smaller ring dimension to achieve the same security
   * level (128-bits).
   *
   */
    uint32_t dnum = 3;

    uint32_t batchSize = 8;
    CCParams<CryptoContextCKKSRNS> parameters;
    parameters.SetMultiplicativeDepth(5);
    parameters.SetScalingModSize(50);
    parameters.SetBatchSize(batchSize);
    parameters.SetScalingTechnique(FLEXIBLEAUTO);
    parameters.SetNumLargeDigits(dnum);

    CryptoContext<DCRTPoly> cc = GenCryptoContext(parameters);

    // Compare the ring dimension in this demo to the one in
    // the previous.
    std::cout << "CKKS scheme is using ring dimension " << cc->GetRingDimension() << std::endl;

    std::cout << "- Using HYBRID key switching with " << dnum << " digits" << std::endl << std::endl;

    cc->Enable(PKE);
    cc->Enable(KEYSWITCH);
    cc->Enable(LEVELEDSHE);

    auto keys = cc->KeyGen();
    cc->EvalRotateKeyGen(keys.secretKey, {1, -2});
```

```
    // Input
    std::vector<double> x = {1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7};
    Plaintext ptxt        = cc->MakeCKKSPackedPlaintext(x);

    std::cout << "Input x: " << ptxt << std::endl;

    auto c = cc->Encrypt(keys.publicKey, ptxt);

    TimeVar t;
    TIC(t);
    auto cRot1 = cc->EvalRotate(c, 1);
    auto cRot2 = cc->EvalRotate(cRot1, -2);
    // The runtime here is smaller than in the previous demo.
    double time3digits = TOC(t);

    Plaintext result;
    std::cout.precision(8);

    cc->Decrypt(keys.secretKey, cRot2, &result);
    result->SetLength(batchSize);
    std::cout << "x rotate by -1 = " << result << std::endl;
    std::cout << " - 2 rotations with HYBRID (3 digits) took " << time3digits << "ms" << std::endl;

}

void FastRotationsDemo1() {

    std::cout << "\n\n\n ===== FastRotationsDemo1 ============= " << std::endl;

    uint32_t batchSize = 8;
    CCParams<CryptoContextCKKSRNS> parameters;
    parameters.SetMultiplicativeDepth(1);
    parameters.SetScalingModSize(50);
    parameters.SetBatchSize(batchSize);

    CryptoContext<DCRTPoly> cc = GenCryptoContext(parameters);

    uint32_t N = cc->GetRingDimension();
    std::cout << "CKKS scheme is using ring dimension " << N << std::endl << std::endl;

    cc->Enable(PKE);
    cc->Enable(KEYSWITCH);
    cc->Enable(LEVELEDSHE);

    auto keys = cc->KeyGen();
    cc->EvalRotateKeyGen(keys.secretKey, {1, 2, 3, 4, 5, 6, 7});

    // Input
    std::vector<double> x = {0, 0, 0, 0, 0, 0, 0, 1};
    Plaintext ptxt        = cc->MakeCKKSPackedPlaintext(x);

    std::cout << "Input x: " << ptxt << std::endl;

    auto c = cc->Encrypt(keys.publicKey, ptxt);

    Ciphertext<DCRTPoly> cRot1, cRot2, cRot3, cRot4, cRot5, cRot6, cRot7;

    // First, we perform 7 regular (non-hoisted) rotations
    // and measure the runtime.
    TimeVar t;
    TIC(t);
    cRot1                 = cc->EvalRotate(c, 1);
    cRot2                 = cc->EvalRotate(c, 2);
    cRot3                 = cc->EvalRotate(c, 3);
    cRot4                 = cc->EvalRotate(c, 4);
    cRot5                 = cc->EvalRotate(c, 5);
    cRot6                 = cc->EvalRotate(c, 6);
    cRot7                 = cc->EvalRotate(c, 7);
    double timeNoHoisting = TOC(t);

    auto cResNoHoist = c + cRot1 + cRot2 + cRot3 + cRot4 + cRot5 + cRot6 + cRot7;

    // M is the cyclotomic order and we need it to call EvalFastRotation
```

```
    uint32_t M = 2 * N;

    // Then, we perform 7 rotations with hoisting.
    TIC(t);
    auto cPrecomp       = cc->EvalFastRotationPrecompute(c);
    cRot1               = cc->EvalFastRotation(c, 1, M, cPrecomp);
    cRot2               = cc->EvalFastRotation(c, 2, M, cPrecomp);
    cRot3               = cc->EvalFastRotation(c, 3, M, cPrecomp);
    cRot4               = cc->EvalFastRotation(c, 4, M, cPrecomp);
    cRot5               = cc->EvalFastRotation(c, 5, M, cPrecomp);
    cRot6               = cc->EvalFastRotation(c, 6, M, cPrecomp);
    cRot7               = cc->EvalFastRotation(c, 7, M, cPrecomp);
    double timeHoisting = TOC(t);
    // The time with hoisting should be faster than without hoisting.

    auto cResHoist = c + cRot1 + cRot2 + cRot3 + cRot4 + cRot5 + cRot6 + cRot7;

    Plaintext result;
    std::cout.precision(8);

    cc->Decrypt(keys.secretKey, cResNoHoist, &result);
    result->SetLength(batchSize);
    std::cout << "Result without hoisting = " << result << std::endl;
    std::cout << " - 7 rotations on x without hoisting took " << timeNoHoisting << "ms" << std::endl;

    cc->Decrypt(keys.secretKey, cResHoist, &result);
    result->SetLength(batchSize);
    std::cout << "Result with hoisting = " << result << std::endl;
    std::cout << " - 7 rotations on x with hoisting took " << timeHoisting << "ms" << std::endl;
}

void FastRotationsDemo2() {


    std::cout << "\n\n\n ===== FastRotationsDemo2 ============= " << std::endl;


    uint32_t digitSize = 10;
    uint32_t batchSize = 8;

    CCParams<CryptoContextCKKSRNS> parameters;
    parameters.SetMultiplicativeDepth(1);
    parameters.SetScalingModSize(50);
    parameters.SetBatchSize(batchSize);
    parameters.SetScalingTechnique(FLEXIBLEAUTO);
    parameters.SetKeySwitchTechnique(BV);
    parameters.SetFirstModSize(60);
    parameters.SetDigitSize(digitSize);

    CryptoContext<DCRTPoly> cc = GenCryptoContext(parameters);

    uint32_t N = cc->GetRingDimension();
    std::cout << "CKKS scheme is using ring dimension " << N << std::endl << std::endl;

    cc->Enable(PKE);
    cc->Enable(KEYSWITCH);
    cc->Enable(LEVELEDSHE);

    auto keys = cc->KeyGen();
    cc->EvalRotateKeyGen(keys.secretKey, {1, 2, 3, 4, 5, 6, 7});

    // Input
    std::vector<double> x = {0, 0, 0, 0, 0, 0, 0, 1};
    Plaintext ptxt        = cc->MakeCKKSPackedPlaintext(x);

    std::cout << "Input x: " << ptxt << std::endl;

    auto c = cc->Encrypt(keys.publicKey, ptxt);

    Ciphertext<DCRTPoly> cRot1, cRot2, cRot3, cRot4, cRot5, cRot6, cRot7;

    // First, we perform 7 regular (non-hoisted) rotations
    // and measure the runtime.
    TimeVar t;
```

```
    TIC(t);
    cRot1                 = cc->EvalRotate(c, 1);
    cRot2                 = cc->EvalRotate(c, 2);
    cRot3                 = cc->EvalRotate(c, 3);
    cRot4                 = cc->EvalRotate(c, 4);
    cRot5                 = cc->EvalRotate(c, 5);
    cRot6                 = cc->EvalRotate(c, 6);
    cRot7                 = cc->EvalRotate(c, 7);
    double timeNoHoisting = TOC(t);

    auto cResNoHoist = c + cRot1 + cRot2 + cRot3 + cRot4 + cRot5 + cRot6 + cRot7;

    // M is the cyclotomic order and we need it to call EvalFastRotation
    uint32_t M = 2 * N;

    // Then, we perform 7 rotations with hoisting.
    TIC(t);
    auto cPrecomp       = cc->EvalFastRotationPrecompute(c);
    cRot1               = cc->EvalFastRotation(c, 1, M, cPrecomp);
    cRot2               = cc->EvalFastRotation(c, 2, M, cPrecomp);
    cRot3               = cc->EvalFastRotation(c, 3, M, cPrecomp);
    cRot4               = cc->EvalFastRotation(c, 4, M, cPrecomp);
    cRot5               = cc->EvalFastRotation(c, 5, M, cPrecomp);
    cRot6               = cc->EvalFastRotation(c, 6, M, cPrecomp);
    cRot7               = cc->EvalFastRotation(c, 7, M, cPrecomp);
    double timeHoisting = TOC(t);
    /* The time with hoisting should be faster than without hoisting.
   * Also, the benefits from hoisting should be more pronounced in this
   * case because we're using BV. Of course, we also observe less
   * accurate results than when using HYBRID, because of using
   * digitSize = 10 (Users can decrease digitSize to see the accuracy
   * increase, and performance decrease).
   */

    auto cResHoist = c + cRot1 + cRot2 + cRot3 + cRot4 + cRot5 + cRot6 + cRot7;

    Plaintext result;
    std::cout.precision(8);

    cc->Decrypt(keys.secretKey, cResNoHoist, &result);
    result->SetLength(batchSize);
    std::cout << "Result without hoisting = " << result << std::endl;
    std::cout << " - 7 rotations on x without hoisting took " << timeNoHoisting << "ms" << std::endl;

    cc->Decrypt(keys.secretKey, cResHoist, &result);
    result->SetLength(batchSize);
    std::cout << "Result with hoisting = " << result << std::endl;
    std::cout << " - 7 rotations on x with hoisting took " << timeHoisting << "ms" << std::endl;
}
```

📃 컴파일 결과

```
[ 94%] Built target lib-benchmark
[ 95%] Built target mult-vs-square
[ 96%] Built target poly-benchmark-16k
[ 97%] Built target poly-benchmark-1k
[ 97%] Built target poly-benchmark-4k
[ 98%] Built target poly-benchmark-64k
[100%] Built target serialize-ckks
joohyun@gimjuhyeon-ui-MacBookAir build % bin/examples/pke/week4_task



 ===== FlexibleAutoDemo =============
CKKS scheme is using ring dimension 16384

Input x: (1, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07,  ... ); Estimated precision: 50 b
its

(x+1)^2*(x^2+2) = (12, 12.201506, 12.406048, 12.613663, 12.824387, 13.038256, 13.255309,
 13.475582,  ... ); Estimated precision: 35 bits

- Using HYBRID key switching with 2 digits

x left rotate by 2 = (12.406048, 12.613663, 12.824387, 13.038256, 13.255309, 13.475582,
12, 12.201506,  ... ); Estimated precision: 35 bits

 2 rotations with HYBRID (2 digits) took 3ms



 ===== FixedAutoDemo =============
CKKS scheme is using ring dimension 16384

Input x: (1, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07,  ... ); Estimated precision: 50 b
its

(x+1)^2*(x^2+2) = (12, 12.201506, 12.406048, 12.613663, 12.824387, 13.038256, 13.255309,
 13.475582,  ... ); Estimated precision: 35 bits

- Using HYBRID key switching with 2 digits

x left rotate by 2 = (12.406048, 12.613663, 12.824387, 13.038256, 13.255309, 13.475582,
12, 12.201506,  ... ); Estimated precision: 35 bits

 2 rotations with HYBRID (2 digits) took 3ms



 ===== FixedManualDemo =============
CKKS scheme is using ring dimension 16384

Input x: (1, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07,  ... ); Estimated precision: 50 b
its

(x+1)^2 * (x^2+2) = (12, 12.201506, 12.406048, 12.613663, 12.824387, 13.038256, 13.25530
9, 13.475582,  ... ); Estimated precision: 40 bits

- Using HYBRID key switching with 2 digits

x left rotate by 2 = (12.406048, 12.613663, 12.824387, 13.038256, 13.255309, 13.475582,
12, 12.201506,  ... ); Estimated precision: 39 bits

 rotations with HYBRID (2 digits) took 3ms
joohyun@gimjuhyeon-ui-MacBookAir build %
```