DGIST 여름인턴 2주차 주간 보고서

3조 권태완, 조후연, 박준석

2주차: 7/14 – 7/18

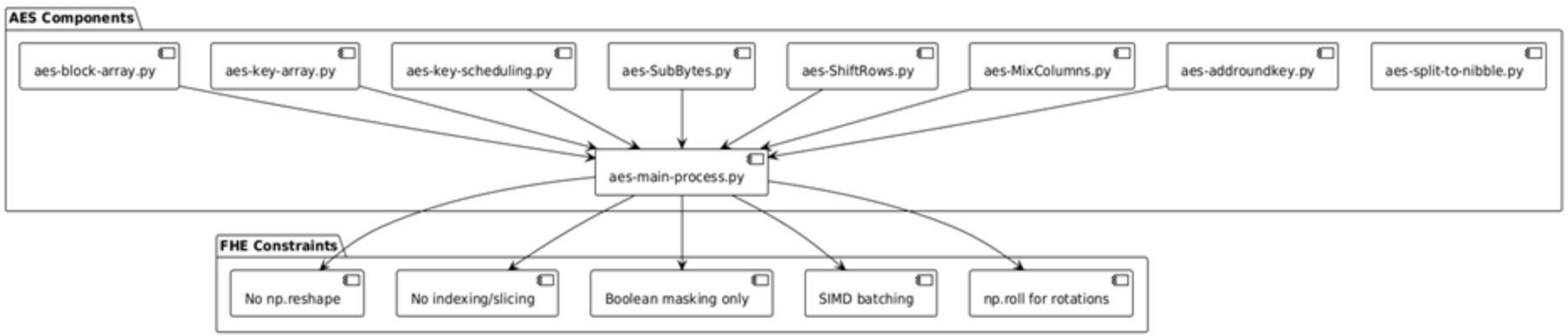
수행 리스트

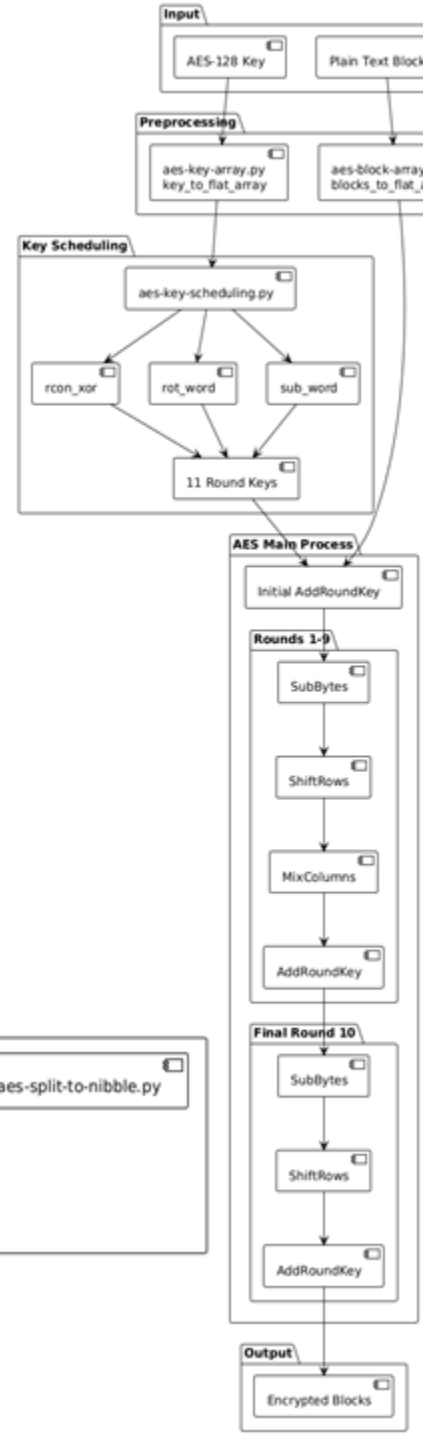
- AES 파이프라인 구성 (w.Numpy)
- Liberate 구현을 위한 모듈 조정
  - SubBytes 치환
  - xor

2주차: AES-pipeline

- AES의 전체 작동 과정의 구현을 위해 오른쪽과 같은 flow의 내부 모듈 설계

- aes-main-process.py를 중심으로 아래와 같은 파일들이 존재.





2주차: AES-pipeline

Tasks Constraints

- 모든 입력 데이터 및 key 의 shape는 (32768,)임을 상정
- [B00_all_blocks, B01_all_blocks,...,B15_all_blocks] 형태로, SIMD 최적화
  - High nibble, Low nibble로 분리하여 구현
- Numpy 사용. Indexing, slicing, reshape, bitwise operation 사용 배제
  - 단, 현재 상황에서 s-box기반 인덱싱 연산 및 xor은 사용이 불가피
- Shiftrow는 단독 연산만 구현. Shiftrow & mixcolumn 혼합 연산의 경우 단순히 설계하는 것이 어려워 liberate를 활용하여 바로 계산하는 것을 계획

2주차: AES-pipeline

Tasks

- SubBytes는 일반적인 구현 후 Liberate 구현 시 다시 한 번 구현
- 그 외에 다른 연산은 numpy로도 일단 전체적인 liberate 구현이 가능하기 때문에 구조에만 잘 fit하게 만들었음

2주차: Encode?

- Encode의 경우 $\ell = 4$ 인 복소수 평면 $\zeta = \exp(-2 \cdot \pi \cdot i/N)$로 인코딩된다.
- Liberate(v0.9.0 기준)의 Encoding함수는 내부에서 m2poly를 호출하고, m2poly 안에서 FFT 결과에 twister(복소수 unit phasor 배열)를 곱해 위상을 인코딩한다.
- twister 자체는 $\exp(-k \cdot \pi \cdot i/N)$ 형태로 만들어지며, 이 값이 곱해지면서 각 주파수 성분에 필요한 위상 변이가 진행된다.
- 결론적으로 단위원 위 N차 근 집합으로의 변환이 같으므로 engine.encode() 사용가능.

2주차: RotWord()

- RotWord는 engine.multiply가 element-wise라는 점을 생각했을 때 Masking 행렬을 사용하는 것이 간단하다고 판단.
- Ciphertext * row_masking + rotation = rotated masked ciphertext

2주차: Rotation- Example

```
engine = Engine(log_coeff_count = 16, special_prime_count=1,
mode="cpu")

# Create test input: (16*2048,) array where each byte is
repeated 2048 times
# Pattern: [0,0,...,0, 1,1,...,1, 2,2,...,2, ..., 15,15,...,
15]
message = np.arange(16, dtype=np.uint8)
message = np.repeat(message, 2048)

# --- Clone a ciphertext ---
secret_key = engine.create_secret_key()
public_key = engine.create_public_key(secret_key)
rotation_key = engine.create_rotation_key(secret_key)
relinearization_key = engine.create_relinearization_key
(secret_key)
```

```
row0_mask = np.concatenate([
    np.ones(max_blocks),    # byte 0
    np.ones(max_blocks),    # byte 1
    np.ones(max_blocks),    # byte 2
    np.ones(max_blocks),    # byte 3
    np.zeros(12 * max_blocks)  # bytes 4-15
])
encode_row0_mask = engine.encode(row0_mask)

ciphertext = engine.encrypt(message, public_key)
print(ciphertext.level)

masked_ciphertext = engine.multiply(ciphertext, encode_row0_mask)
rot_masked_ciphertext = engine.rotate(masked_ciphertext, rotation_key,
4*max_blocks)

print(rot_masked_ciphertext.level)
masked_dec_ciphertext = engine.decrypt(rot_masked_ciphertext, secret_key)
```

2주차: Rotation() - Example

```
print(masked_dec_ciphertext[0*max_blocks:(1*max_blocks)])
print(masked_dec_ciphertext[1*max_blocks:(2*max_blocks)])
print(masked_dec_ciphertext[2*max_blocks:(3*max_blocks)])
print(masked_dec_ciphertext[3*max_blocks:(4*max_blocks)])
print(masked_dec_ciphertext[4*max_blocks:(5*max_blocks)])
print(masked_dec_ciphertext[5*max_blocks:(6*max_blocks)])
print(masked_dec_ciphertext[6*max_blocks:(7*max_blocks)])
print(masked_dec_ciphertext[7*max_blocks:(8*max_blocks)])
print(masked_dec_ciphertext[8*max_blocks:(9*max_blocks)])
print(masked_dec_ciphertext[9*max_blocks:(10*max_blocks)])
print(masked_dec_ciphertext[10*max_blocks:(11*max_blocks)])
print(masked_dec_ciphertext[11*max_blocks:(12*max_blocks)])
print(masked_dec_ciphertext[12*max_blocks:(13*max_blocks)])
print(masked_dec_ciphertext[13*max_blocks:(14*max_blocks)])
print(masked_dec_ciphertext[14*max_blocks:(15*max_blocks)])
print(masked_dec_ciphertext[15*max_blocks:(16*max_blocks)])
```

level: 40 -> 39

```
Byte 0: [-7.18741010e-10 -8.70488133e-10 -9.27824103e-10 ...  2.32667873e-10 -9.43472243e-10  4.11663840e-10]
Byte 1: [ 3.78648517e-10  9.93901794e-10  1.70631752e-10 ... -4.08140282e-10 -8.07685302e-10 -2.88442830e-10]
Byte 2: [-1.05738148e-09  5.28808410e-10 -3.20729793e-10 ... -3.08659038e-10  6.05140102e-10 -6.76228441e-10]
Byte 3: [-2.33072209e-10 -1.87205111e-10 -1.22432500e-09 ...  4.97176109e-10 -1.05713555e-09 -1.69745377e-09]
Byte 4: [ 1.13687398e-08 -7.50084481e-09  1.07607422e-08 ...  3.51572014e-09 -7.54943206e-09 -5.96235538e-10]
Byte 5: [0.99999999 1.         1.00000004 ... 1.         1.00000001 1.        ]
Byte 6: [2.         2.         2.         ... 2.00000001 2.         1.99999999]
Byte 7: [3.00000002 3.         3.         ... 3.         2.99999999 3.00000001]
Byte 8: [-3.13944948e-10  1.59467406e-10  2.78230867e-10 ... -1.46095728e-10  3.50947818e-10 -5.03082482e-11]
Byte 9: [1.54628143e-10 4.35958083e-11 5.21660424e-11 ... 2.57944641e-10 8.17179250e-10 2.82920255e-10]
Byte 10: [ 3.61339475e-10 -2.74429367e-10  3.10232308e-12 ... -6.30881742e-11 -5.59990061e-10 -1.79579092e-10]
Byte 11: [1.13041063e-10 5.34574004e-11 9.76620723e-11 ... 5.60290269e-10 6.84434653e-10 2.84137445e-10]
Byte 12: [ 8.58778791e-10  4.65797955e-10  5.00664349e-10 ...  1.02793311e-10 -5.68016013e-10  8.46982338e-12]
Byte 13: [-5.45759212e-10 -2.03670789e-10 -4.83016380e-10 ... -2.81438692e-10 -9.87719159e-10 -1.23345602e-11]
Byte 14: [ 2.03533343e-10  4.91035842e-10  4.20063827e-10 ...  3.03938217e-10 -1.72603906e-11 -1.10941986e-10]
Byte 15: [-5.57769032e-10 -9.67277762e-10  9.74771979e-10 ...  4.81123341e-10 -8.47808832e-10  1.13294412e-09]
```

2주차:SubBytes 전략 수립

문제 분석: S-Box의 비선형성과 FHE 제약	최종 채택 전략: 다변수 다항식 평가
한계점 1: 단일 다항식 근사 S-Box를 단일 고차 다항식으로 근사하는 방식은 100% 정확성 보장이 어렵고, 곱셈 깊이가 비효율적임을 확인.	핵심 아이디어 S-Box를 '8비트 입력 → 4비트 출력' 두 개의 2변수 다항식으로 분해하여, desilofhe 라이브러리의 최적화된 다항식 연산 기능을 직접 활용 동형 연산 흐름 1. 동형 니블 추출: engine.evaluate_polynomial() 을 통해 Encrypted(Byte) 에서 Encrypted(Byte_high) + Encrypted(Byte_low) 로 분해하여 추출 2. 다변수 다항식 평가: engine.make_power_basis 등을 활용하여, Encrypted(Byte_high), Encrypted(Byte_low) 를 효율적으로 계산 3. 결과 결합: engine.multiply () 과 engine.add() 로 최종 Encrypted(Result) 를 생성.
한계점 2: 비트 단위 회로 '비트-슬라이싱' 기반 논리 회로 구현은 정확하지만, desilofhe 라이브러리의 강력한 산술 연산 기능을 최대한 활용하지 못하고 구현이 복잡함.	

2주차:SubBytes 전략 수립

Numpy 와 Fraction 을 사용하여 알고리즘의 수학적 정확성을 100% 검증하는 프로토타입을 우선 구현.

```
FUNCTION sbox_poly(ciphertext ct_byte, FHE_Engine engine, SecretKey sk, PublicKey pk):
  // 입력: 암호화된 8비트 바이트 (ct_byte)
  // 출력: S-Box 변환 결과가 암호화된 새로운 암호문

  // 1. 다항식 계수 준비 (최초 1회만 실행)
  // S-Box를 100% 정확하게 표현하는 2변수 다항식의 계수 C_hi, C_lo를
  // 미리 계산하여 준비한다.
  CALL _initialize_coefficients()

  // --- 평문 프로토타입의 핵심 로직 ---

  // 2. 입력 암호문 복호화
  // 암호화된 바이트를 비밀키(sk)를 이용해 평문으로 변환한다.
  LET decrypted_vector = engine.decrypt(ct_byte, sk)
```

2주차:SubBytes 전략 수립

```
// 3. 평문 값 추출 및 분해
  // 복호화된 결과(실수 벡터)의 첫 번째 값을 반올림하여 8비트 정수 x로 만든다.
  LET x = round(decrypted_vector[0])
  // 정수 x를 상위 4비트(hi)와 하위 4비트(lo)로 분리한다.
  LET hi = x >> 4
  LET lo = x & 0xF

  // 4. 평문 위에서의 다항식 평가
  // 미리 준비된 계수 C_hi를 이용해, S-Box 결과의 상위 니블 y_hi를 계산한다.
  // 이 과정은 Σ(C_hi[i][j] * hi^i * lo^j) 계산과 같다.
  LET y_hi = evaluate_polynomial(hi, lo, C_hi)

  // 계수 C_lo를 이용해, S-Box 결과의 하위 니블 y_lo를 계산한다.
  LET y_lo = evaluate_polynomial(hi, lo, C_lo)

  // 5. 최종 결과 결합 및 재암호화
  // 계산된 두 니블을 다시 하나의 8비트 정수로 합친다.
  LET final_result = (y_hi * 16) + y_lo

  // 최종 평문 결과를 공개키(pk)를 이용해 다시 암호화하여 반환한다.
  RETURN engine.encrypt(final_result, pk)

END FUNCTION
```

테스트 및 검증 결과
>>

검증 방법: pytest 를 사용하여 0부터 255까지 모든 입력값에 대한 전수 검사를 수행.

검증 절차:
1. 입력값 i 를 암호화
2. sbox_poly 함수로 계산
3. 결과를 복호화
4. 표준 S-Box 정답과 비교

최종 결과:
프로토타입은 모든 256개 케이스에 대해 표준 S-Box와 일치하는 결과를 도출, 알고리즘의 논리적 정당성 확보.

2주차: XOR 구현 전략

FFT 기반 계수 계산 16×16 XOR 테이블 → 2D FFT → 다항식 계수로 전환 하여 FHE에서 사용가능

$\mathrm{XOR}(a, b) \approx \Sigma_{i,j}\, c_{ij} \cdot a^i \cdot b^j \ (i + j \le 7)$ 식을 이용해 연산 차수 감소

최적화기법	효과	구현
PowerBasis 캐싱	재계산 90퍼센트감소	완료
Sparce coefficient	연산량 60퍼센트 감소	완료
켤레 복소수 최적화	계수 15에서 7로 낮추는 효과	완료

2주차: XOR 구현 전략

코드스니펫

```python
def compute_xor_approximation(
    self,
    *,
    degree: int = 7,
    encode_output: bool = True,
) -> np.ndarray:
    size = 16
    xor_2d = np.zeros((size, size), dtype=np.complex128 if encode_output else np.float64)
    for i in range(size):
        for j in range(size):
            v = i ^ j
            xor_2d[i, j] = np.exp(2j * np.pi * v / 16) if encode_output else float(v)
    coeffs_2d = _fft_forward_nd(xor_2d)
    if degree < size - 1:
        mask = np.zeros_like(coeffs_2d, dtype=bool)
        for i in range(min(degree + 1, size)):
            for j in range(min(degree + 1 - i, size)):
                mask[i, j] = True
        coeffs_2d[~mask] = 0
    return coeffs_2d
```

2주차: XOR 구현 전략 - Challenge

- 과도한 레벨소모
  - xor 1회당 3레벨소모
- 원인 추측: 각 곱셈마다 relinerlize+rescale 매번 적용

3주차: 7/21 – 7/25 계획

Liberate로 AES Enc 파트 구현 시작하기

- Key Expansion
  - SubWord
  - Rcon
- AddRoundKey
- SubBytes
- ShiftRows
- MixColumns

끝