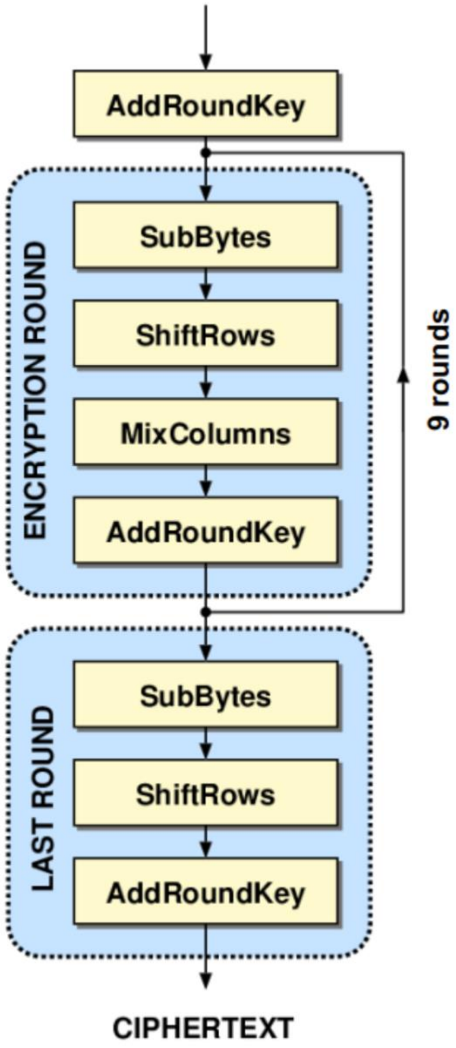
DGIST 여름인턴 4주차

최종 보고서

3조 권태완, 조후연, 박준석

github-repo

AES – 128: Objectives



- Key Scheduling with Liberatefhe

- Encryption with Liberatefhe

- Decryption with Liberatefhe

AES - 128

Key Scheduling with Liberatefhe

github-repo

Key Scheduling

github-repo

Key Scheduling

zeta 위상으로의 변환 후 암호문의 engine.add, engine.multiply는 정수의 암호문과 달라진다

engine.multiply = zeta 위상에서의 gf 합, 즉 16 모듈러 합

```
[ 1 12 10  7  6 13  1 11  3  1  8 15 11 12 11 12]
[ 8  2 13  7  8  5  2 14 12 10  6 13  8  7  7  3]
[9, 14, 7, 14, 14, 2, 3, 9, 15, 11, 14, 12, 3, 3, 2, 15]
```

engine.add = ?? (아직 패턴을 찾지 못함)

```
ints_a: [ 1 12 10  7  6 13  1 11  3  1  8 15 11 12 11 12]
ints_b: [ 8  2 13  7  8  5  2 14 12 10  6 13  8  7  7  3]
zeta_added_2_int: [4, 15, 12, 7, 7, 7, 2, 13, 0, 14, 7, 14, 9, 9, 9, 15]
ints_a + ints_b: [ 9 14 23 14 14 18  3 25 15 11 14 28 19 19 18 15]
ints_a ^ ints_b: [ 9 14  7  0 14  8  3  5 15 11 14  2  3 11 12 15]
ints_a * ints_b: [  8  24 130  49  48  65   2 154  36  10  48 195  88  84  77  36]
(ints_a + ints_b) % 16: [ 9 14  7 14 14  2  3  9 15 11 14 12  3  3  2 15]
(ints_a ^ ints_b) % 16: [ 9 14  7  0 14  8  3  5 15 11 14  2  3 11 12 15]
(ints_a * ints_b) % 16: [ 8  8  2  1  0  1  2 10  4 10  0  3  8  4 13  4]
```

Key Scheduling

```python
engine = engine_context.get_engine()
public_key = engine_context.get_public_key()

max_blocks = 2048

word_hi = enc_key_hi_list.copy()
word_lo = enc_key_lo_list.copy()

    # key scheduling round
for i in range(4, 44):
    if i % 4 == 0:
        # 4의 배수 - 1의 워드를 rot_word 연산 후 sub_word 연산 후 rcon_xor 연산 후 4의 배수 - 4 번째 워드와 xor 연산
        start_time = time.time()
        sub_word_hi, sub_word_lo = _sub_word(engine_context, word_hi[i-1], word_lo[i-1])
        rot_word_hi, rot_word_lo = _rot_word(engine_context, sub_word_hi, sub_word_lo)
        rcon_xor_hi, rcon_xor_lo = _rcon_xor(engine_context, rot_word_hi, rot_word_lo, i//4)
        xor_hi, xor_lo = _xor(engine_context, rcon_xor_hi, rcon_xor_lo, word_hi[i-4], word_lo[i-4])
        word_hi.append(xor_hi)
        word_lo.append(xor_lo)
        end_time = time.time()
        print(f"Key scheduling round {i} time: {end_time - start_time} seconds")
    else:
        start_time = time.time()
        # 4의 배수 x - 4의 배수 -1 번째 워드와 4의 배수 - 3 번째 워드를 xor 연산
        xor_hi, xor_lo = _xor(engine_context, word_hi[i-1], word_lo[i-1], word_hi[i-4], word_lo[i-4])
        word_hi.append(xor_hi)
        word_lo.append(xor_lo)
        end_time = time.time()
        print(f"Key scheduling round {i} time: {end_time - start_time} seconds")
return word_hi, word_lo
```

AES - 128

Encryption with Liberatefhe

github-repo

Encryption with Liberatefhe

★ rotate, bootstrap 등의 연산은 결과를 NTT 형태로 되돌려주기 때문에 coefficient도메인만 입력으로 받는 make_power_basis를 사용하기 위해서는 도메인 변환 함수인 engine.intt(ciphertext)를 사용하여 NTT → Coefficient 도메인으로 변환

```
delta_list = [1*2048, 2*2048, 3*2048, 4*2048, 8*2048, -1*2048, -2*2048, -3*2048,
-4*2048]

engine_context = engine_initiation(signature=1, mode='parallel',
use_bootstrap=True, thread_count = 16, device_id = 0, fixed_rotation=True,
delta_list=delta_list)
```

지원되는 생성자 시그니처:

1. Engine(mode, use_bootstrap, use_multiparty, thread_count, device_id)
2. Engine(max_level, mode, use_multiparty, thread_count, device_id)
3. Engine(log_coeff_count=N, special_prime_count, use_multiparty, thread_count, device_id)

- NTT/INTT is now a public API. These functions can be used to make the calculation faster when a ciphertext or a plaintext is multiplied to several different ciphertexts.

2025/07/24 desilofhe 1.3.0 Release Notes

Encryption with Liberatefhe

xor 연산
작동 시간

level=10 암호문 기준
30.18s-15.47s 소요

level 5 소모

```python
def _xor_operation(engine_context, enc_alpha, enc_beta):
    engine = engine_context.engine
    relin_key = engine_context.relinearization_key
    conjugate_key = engine_context.conjugation_key

    # 1. Build power bases
    base_x = build_power_basis(engine, enc_alpha, relin_key, conjugate_key)
    base_y = build_power_basis(engine, enc_beta,  relin_key, conjugate_key)

    # 2. Pre-encoded polynomial coefficients
    coeff_pts = _get_coeff_plaintexts(engine)

    # 3. Evaluate polynomial securely
    cipher_res = engine.multiply(enc_alpha, 0.0)

    for (i, j), coeff_pt in coeff_pts.items():
        term = engine.multiply(base_x[i], base_y[j], relin_key)
        term_res = engine.multiply(term, coeff_pt)
        cipher_res = engine.add(cipher_res, term_res)

    return cipher_res
```

Encryption with Liberatefhe

subbytes
작동시간

level=10기준
41.72s-31.67s

```python
def _build_tables() -> Tuple[np.ndarray, np.ndarray]:
    """Return two (16,16) arrays with ζ^hi and ζ^lo values."""
    f_hi = np.empty((16, 16), dtype=np.complex128)
    f_lo = np.empty((16, 16), dtype=np.complex128)

    for a in range(16):
        for b in range(16):
            x = (a << 4) | b
            y = int(S_BOX[x])
            hi = (y >> 4) & 0xF
            lo = y & 0xF
            f_hi[a, b] = ZETA ** hi
            f_lo[a, b] = ZETA ** lo
    return f_hi, f_lo


def _ifft2_coeff(table: np.ndarray) -> np.ndarray:
    return np.fft.ifft2(table)
```

```python
def _poly_eval(engine_context: CKKS_EngineContext, ct_hi: Any, ct_lo: Any, which: str):
    engine = engine_context.get_engine()
    relin_key = engine_context.get_relinearization_key()
    conj_key = engine_context.get_conjugation_key()

    basis_x = _build_power_basis(engine_context, ct_hi, relin_key, conj_key)
    basis_y = _build_power_basis(engine_context, ct_lo, relin_key, conj_key)

    coeff_pt = _get_coeff_plaintexts(engine, which)

    # start with zero ciphertext (scale/level match)
    cipher_res = engine.multiply(ct_hi, 0.0)

    for (p, q), pt in coeff_pt.items():
        term = engine.multiply(basis_x[p], basis_y[q], relin_key)
        term = engine.multiply(term, pt)
        cipher_res = engine.add(cipher_res, term)

    return cipher_res

def _sbox_poly(engine_context: CKKS_EngineContext, ct_hi: Any, ct_lo: Any) -> Tuple[Any, Any]:
    """Homomorphic evaluation of AES S-Box on 4-bit nibbles (upper/lower)."""
    engine = engine_context.get_engine()
    relin_key = engine_context.get_relinearization_key()
    conj_key = engine_context.get_conjugation_key()

    hi_ct = _poly_eval(engine_context, ct_hi, ct_lo, "hi")
    lo_ct = _poly_eval(engine_context, ct_hi, ct_lo, "lo")

    # Optionally refresh scale/level via bootstrap (kept from original code)
    hi_ct = engine.bootstrap(hi_ct, relin_key, conj_key, engine_context.get_bootstrap_key())
    lo_ct = engine.bootstrap(lo_ct, relin_key, conj_key, engine_context.get_bootstrap_key())

    return hi_ct, lo_ct
```

Encryption with Liberatefhe

ShiftRows
작동시간

level=10기준
0.59초-0.50초

level 1 소모

현재 shiftrows의 결과값이 잘못되어
연산 결과가 표준과 맞지 않는 상황
→ 해결

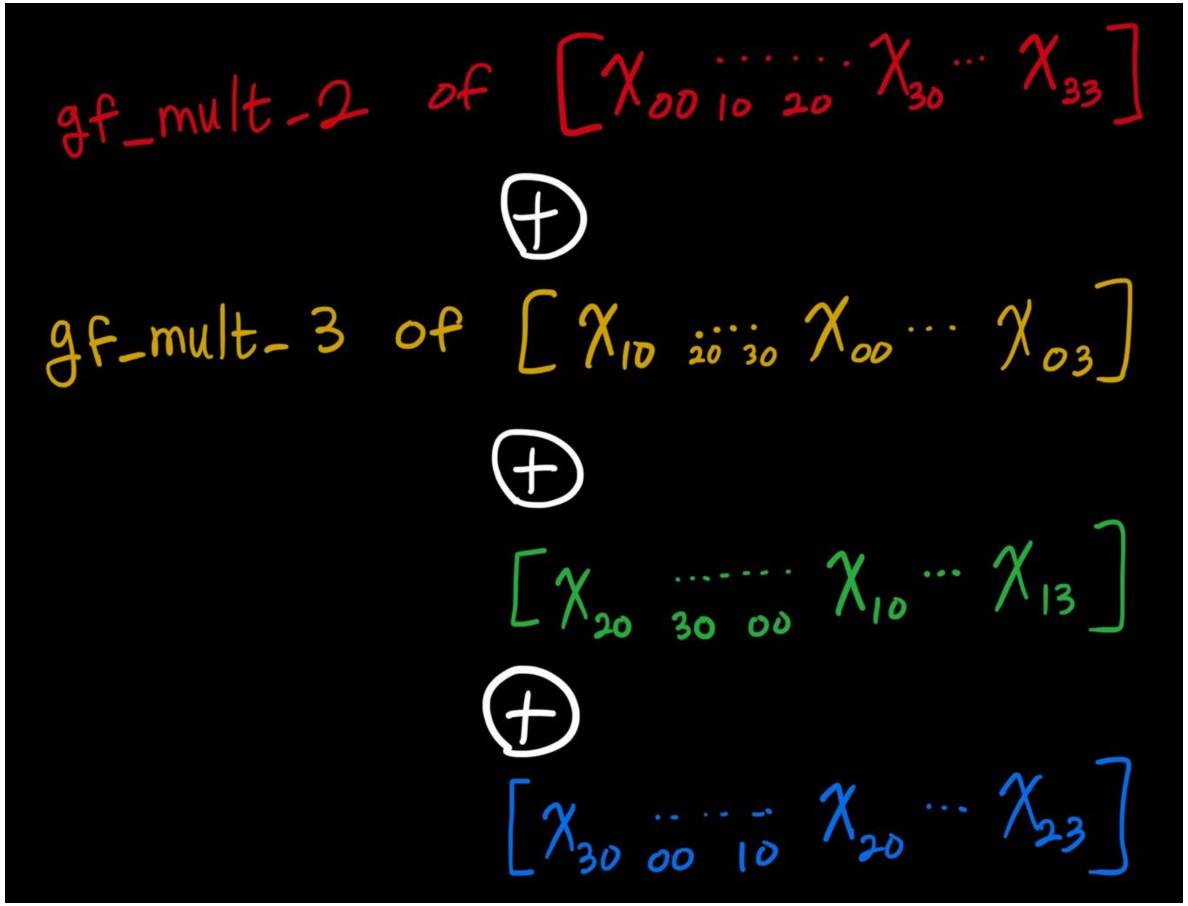
mask를 list 그대로 넣었어야했는데 encode하고
넣어서 생겨버린 일...

```
# ---------------------------------------------------------------------------
# rotate operation of High nibble
# ---------------------------------------------------------------------------
# fixed_rotation_key_list 내용물은 -3 -2 -1 1 2 3 이렇게 저장됨.
# mask_row_1에 대해 0은 3 로 한번, 123은 -1로 한 번 회전
rotated_row_hi_1_0 = engine.rotate(masked_row_hi_1_0, fixed_rotation_key_list[5])
rotated_row_hi_1_123 = engine.rotate(masked_row_hi_1_123, fixed_rotation_key_list[2])

# mask_row_2에 대해 01은 2로 한번, 23은 -2로 한 번 회전
rotated_row_hi_2_01 = engine.rotate(masked_row_hi_2_01, fixed_rotation_key_list[4])
rotated_row_hi_2_23 = engine.rotate(masked_row_hi_2_23, fixed_rotation_key_list[1])

# mask_row_3에 대해 012는 1로 한번, 3은 -3로 한 번 회전
rotated_row_hi_3_012 = engine.rotate(masked_row_hi_3_012, fixed_rotation_key_list[3])
rotated_row_hi_3_3 = engine.rotate(masked_row_hi_3_3, fixed_rotation_key_list[0])

# concatenate all the rotated rows
rotated_rows_hi_0 = engine.add(masked_row_hi_0, rotated_row_hi_1_0)
rotated_rows_hi_1 = engine.add(rotated_rows_hi_0, rotated_row_hi_1_123)
rotated_rows_hi_2 = engine.add(rotated_rows_hi_1, rotated_row_hi_2_01)
rotated_rows_hi_3 = engine.add(rotated_rows_hi_2, rotated_row_hi_2_23)
rotated_rows_hi_4 = engine.add(rotated_rows_hi_3, rotated_row_hi_3_012)
rotated_rows_hi = engine.add(rotated_rows_hi_4, rotated_row_hi_3_3)
```

Encryption with Liberatefhe

MixColumns 구현 방법

2X_{00} + 3X_{10} + 1X_{20} + 1X_{30}	2X_{01} + 3X_{11} + 1X_{21} + 1X_{31}
1X_{00} + 2X_{10} + 3X_{20} + 1X_{30}	1X_{01} + 2X_{11} + 3X_{21} + 1X_{31}
1X_{00} + 1X_{10} + 2X_{20} + 3X_{30}	1X_{01} + 1X_{11} + 2X_{21} + 3X_{31}
3X_{00} + 1X_{10} + 1X_{20} + 2X_{30}	3X_{01} + 1X_{11} + 1X_{21} + 2X_{31}



gf_mult 구현 설명은 Decryption에서!

Encryption with Liberatefhe

MixColumns

level=10 기준
96.42s-89.58s

level 20 감소

```
# ----------------------------------------------------------------
# -------------- 1. rotate 연산 ----------------------------
# ----------------------------------------------------------------
# ct_hi 암호문 3개 생성
# 각각 -4, 8, 4 만큼 회전한 암호문 3개 생성
ct_hi_rot_list = [engine.rotate(ct_hi, fixed_rotation_key_neg_4_2048), engine.rotate(ct_hi, fixed_rotation_key_neg_8_2048),
engine.rotate(ct_hi, fixed_rotation_key_neg_12_2048)]

# gf_mul_2 연산을 오리지널 암호문에 대해 수행: level 5 소모
one_ct_hi, one_ct_lo = gf_mult_2(engine_context, ct_hi, ct_lo)

# gf_mul_3 연산을 -4 * 2048 만큼 왼쪽으로 회전한 암호문에 대해 수행: level 5 소모
two_ct_hi, two_ct_lo = gf_mult_3(engine_context, ct_hi_rot_list[0], ct_lo_rot_list[0])

three_ct_hi = ct_hi_rot_list[1] # -8 * 2048 만큼 왼쪽으로 회전한 암호문
three_ct_lo = ct_lo_rot_list[1] # -8 * 2048 만큼 왼쪽으로 회전한 암호문

four_ct_hi = ct_hi_rot_list[2] # -12 * 2048 만큼 왼쪽으로 회전한 암호문
four_ct_lo = ct_lo_rot_list[2] # -12 * 2048 만큼 왼쪽으로 회전한 암호문

# high nibble
mixed_ct_hi = _xor_operation(engine_context, one_ct_hi, two_ct_hi) # level 5 감소

# bootstrap 연산 수행
mixed_ct_hi = engine.bootstrap(mixed_ct_hi, engine_context.get_relinearization_key(), engine_context.get_conjugation_key(),
engine_context.get_bootstrap_key()) # level 10 복귀

mixed_ct_hi = _xor_operation(engine_context, mixed_ct_hi, three_ct_hi) # level 5 감소
mixed_ct_hi = _xor_operation(engine_context, mixed_ct_hi, four_ct_hi) # level 5 감소
```

Encryption with Liberatefhe

결과:
한 라운드 당 평균 140초
암호화 전체 파이프라인 1370초

* 캐싱으로 인해 라운드가 지남에 따라 모든 연산 속도 최대 20초 감소

2048 병렬 기준
Amortized Time of Encryption = 0.66s

Amortized Time of XOR = 0.014s
Amortized Time of SubBytes = 0.02s
Amortized Time of ShiftRows = 0.00027s
Amortized Time of MixColumns = 0.047s

Bootstrapping time = 건당 8초

AES - 128

Decryption with Liberatefhe

github-repo

Decryption with Liberatefhe

Inv-SubBytes

level=10기준
40.12s

```python
def build_target_tables() -> tuple[np.ndarray, np.ndarray]:
    """INV_S_BOX 출력 니블을 ζ-인코딩하여 (16,16) 목표 테이블을 생성한다."""
    f_hi = np.empty((16, 16), dtype=np.complex128)
    f_lo = np.empty((16, 16), dtype=np.complex128)

    for a in range(16):  # 입력 상위 니블 (a)
        for b in range(16):  # 입력 하위 니블 (b)
            # 입력값 (a,b)에 대한 S-Box의 결과
            sbox_in = (a << 4) | b
            sbox_out = int(INV_S_BOX[sbox_in])

            # 출력 니블을 다시 제타(zeta) 값으로 변환
            out_hi = (sbox_out >> 4) & 0xF
            out_lo = sbox_out & 0xF
            f_hi[a, b] = ZETA ** out_hi
            f_lo[a, b] = ZETA ** out_lo

    return f_hi, f_lo
```

```python
def _poly_eval(engine_context: CKKS_EngineContext, ct_hi: Any, ct_lo: Any, which: str):
    """주어진 암호문들에 대해 hi 또는 lo 다항식을 평가합니다."""
    engine = engine_context.get_engine()
    rlk = engine_context.get_relinearization_key()
    conj_key = engine_context.get_conjugation_key()

    basis_x = _build_power_basis(engine_context, ct_hi, rlk, conj_key)
    basis_y = _build_power_basis(engine_context, ct_lo, rlk, conj_key)

    coeff_pt = _get_coeff_plaintexts(engine, which)

    cipher_res = engine.multiply(ct_hi, 0.0) # 결과 암호문 초기화

    for (p, q), pt in coeff_pt.items():
        term = engine.multiply(basis_x[p], basis_y[q], rlk)
        term = engine.multiply(term, pt)
        cipher_res = engine.add(cipher_res, term)

    return cipher_res
def inv_sbox_poly(ctx: CKKS_EngineContext, ct_hi_zeta: Any, ct_lo_zeta: Any) -> Tuple[Any, Any]:
    """동형 Inverse S-Box의 고수준 오케스트레이션 함수."""
    engine = ctx.get_engine()
    rlk, conj_key = ctx.get_relinearization_key(), ctx.get_conjugation_key()

    res_hi = _poly_eval(ctx, ct_hi_zeta, ct_lo_zeta, "hi")
    res_lo = _poly_eval(ctx, ct_hi_zeta, ct_lo_zeta, "lo")

    # 암호화 파이프라인의 안정성을 위해 부트스트래핑을 동일하게 적용
    res_hi = engine.bootstrap(res_hi, rlk, conj_key, ctx.get_bootstrap_key())
    res_lo = engine.bootstrap(res_lo, rlk, conj_key, ctx.get_bootstrap_key())

    return res_hi, res_lo
```

Decryption with Liberatefhe

Inv-ShiftRows

level=10기준
0.58s

level 1 감소

```
# rotate operation of High nibble
# ---------------------------------------------------------------------------
# fixed_rotation_key_list 내용물은 -3 -2 -1 1 2 3 이렇게 저장됨.
# mask_row_1에 대해 0은 3 로 한번, 123은 -1로 한 번 회전
# inverse: rows_1_012 needs +1 (index 3), rows_1_3 needs -3 (index 0)
rotated_row_hi_1_012 = engine.rotate(masked_row_hi_1_012, fixed_rotation_key_list[3])  # +1*2048
rotated_row_hi_1_3 = engine.rotate(masked_row_hi_1_3, fixed_rotation_key_list[0])     # -3*2048

# mask_row_2에 대해 01은 2로 한번, 23은 -2로 한 번 회전
rotated_row_hi_2_01 = engine.rotate(masked_row_hi_2_01, fixed_rotation_key_list[4])
rotated_row_hi_2_23 = engine.rotate(masked_row_hi_2_23, fixed_rotation_key_list[1])

# mask_row_3에 대해 012는 1로 한번, 3은 -3로 한 번 회전
# inverse: row_3_0 needs +3 (index 5), row_3_123 needs -1 (index 2)
rotated_row_hi_3_0 = engine.rotate(masked_row_hi_3_0, fixed_rotation_key_list[5])      # +3*2048
rotated_row_hi_3_123 = engine.rotate(masked_row_hi_3_123, fixed_rotation_key_list[2])  # -1*2048

# concatenate all the rotated rows
rotated_rows_hi_0 = engine.add(masked_row_hi_0, rotated_row_hi_1_012)
rotated_rows_hi_1 = engine.add(rotated_rows_hi_0, rotated_row_hi_1_3)
rotated_rows_hi_2 = engine.add(rotated_rows_hi_1, rotated_row_hi_2_01)
rotated_rows_hi_3 = engine.add(rotated_rows_hi_2, rotated_row_hi_2_23)
rotated_rows_hi_4 = engine.add(rotated_rows_hi_3, rotated_row_hi_3_123)
rotated_rows_hi = engine.add(rotated_rows_hi_4, rotated_row_hi_3_0)
```

Decryption with Liberatefhe

2X_{00} + 3X_{10} + 1X_{20} + 1X_{30}	2X_{01} + 3X_{11} + 1X_{21} + 1X_{31}
1X_{00} + 2X_{10} + 3X_{20} + 1X_{30}	1X_{01} + 2X_{11} + 3X_{21} + 1X_{31}
1X_{00} + 1X_{10} + 2X_{20} + 3X_{30}	1X_{01} + 1X_{11} + 2X_{21} + 3X_{31}
3X_{00} + 1X_{10} + 1X_{20} + 2X_{30}	3X_{01} + 1X_{11} + 1X_{21} + 2X_{31}

gf mult 14 of $[X_{00} \cdots {}_{10} \cdots {}_{20}\ X_{30} \cdots X_{33}]$

$\oplus$

gf mult 11 of $[X_{10} \cdots {}_{20} \cdots {}_{30}\ X_{00} \cdots X_{03}]$

left shift $[-4 \cdot 2048]$

$\oplus$

gf mult 13 of $[X_{20} \cdots {}_{30} \cdots {}_{00}\ X_{10} \cdots X_{13}]$

$\oplus$

right shift $[8 \cdot 2048]$

gf mult 9 of $[X_{30} \cdots {}_{00} \cdots {}_{10}\ X_{20} \cdots X_{23}]$

right shift $[4 \cdot 2048]$

Decryption with Liberatefhe

Inv-MixColumns

level=10기준
110.71s

```
# ------------------------------------------------------------
# -------------- 1. rotate 연산 ----------------------------
# ------------------------------------------------------------
# ct_hi 암호문 3개 생성
# 각각 4, 8, 12 만큼 회전한 암호문 3개 생성
ct_hi_rot_list = engine.rotate_batch(ct_hi, list_of_fixed_rotation_keys)
# ------------------------------------------------------------
# -------------- 2. variable naming convention --------------
# ------------------------------------------------------------
one_ct_hi, one_ct_lo = gf_mult_14(engine_context, ct_hi, ct_lo)

two_ct_hi, two_ct_lo = gf_mult_11(engine_context, ct_hi_rot_list[0], ct_lo_rot_list[0])

three_ct_hi, three_ct_lo = gf_mult_13(engine_context, ct_hi_rot_list[1], ct_lo_rot_list[1]) # -8 * 2048 만큼 왼쪽으로 회전한 암호문

four_ct_hi, four_ct_lo = gf_mult_9(engine_context, ct_hi_rot_list[2], ct_lo_rot_list[2]) # -12 * 2048 만큼 왼쪽으로 회전한 암호문
# high nibble
mixed_ct_hi = _xor_operation(engine_context, one_ct_hi_bootstrap, two_ct_hi_bootstrap)
mixed_ct_hi = _xor_operation(engine_context, mixed_ct_hi, three_ct_hi)

# Bootstrap 연산 수행
mixed_ct_hi = engine.bootstrap(mixed_ct_hi, engine_context.get_relinearization_key(), engine_context.get_conjugation_key(),
engine_context.get_bootstrap_key())

mixed_ct_hi = _xor_operation(engine_context, mixed_ct_hi, four_ct_hi)
```

TAB to jump here

Decryption with Liberatefhe

결과:
한 라운드 당 평균 150초
암호화 전체 파이프라인 1495초

* 캐싱으로 인해 라운드가 지남에 따라 모든 연산 속도 최대 20초 감소

2048 병렬 기준
Amortized Time of Decryption = 0.72s

Amortized Time of XOR = 0.014s
Amortized Time of Inv-SubBytes = 0.02s
Amortized Time of Inv-ShiftRows = 0.00027s
Amortized Time of Inv-MixColumns = 0.053s

Bootstrapping time = 건당 8초