FIPS-202 RUST 구현 문제

https://youtu.be/aRa2SAVVS0w

1. FIPS 202

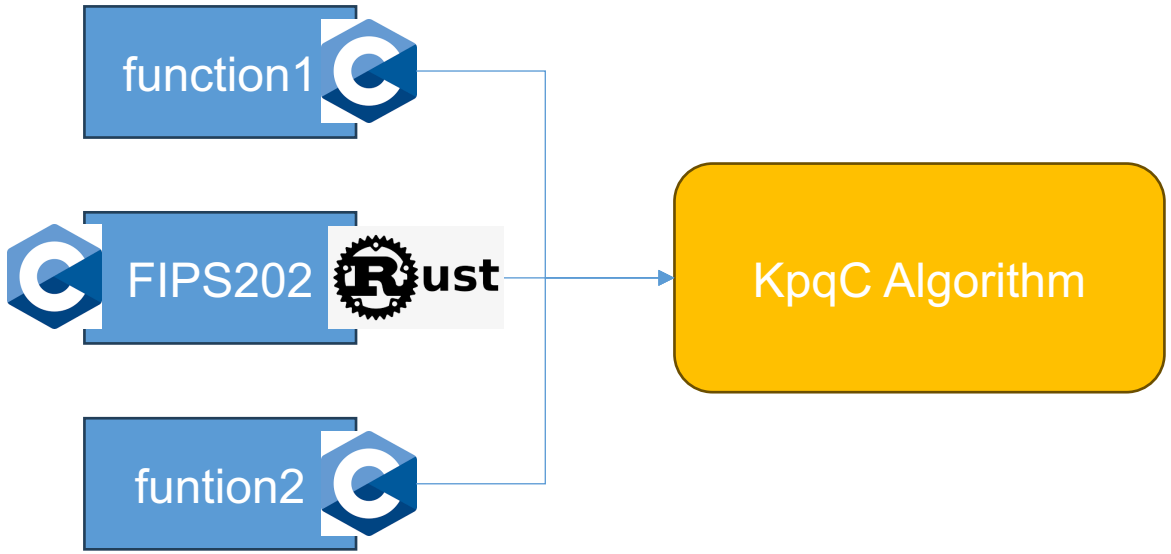
- Fips202는 NIST에서 만든 해시 함수의 표준이다.
  - 여기에서 표준이 된 해시 함수는 SHA3 해시 알고리즘
  - Fips203, fips204, fips205의 표준 작업 진행중

- SHA3는 Keccak이라는 알고리즘을 기반으로 동작함.
  - 스펀지 구조라는 독특한 방식으로 물을 흡수하고 짜내는 것처럼 데이터를 흡수하고 복잡하게 섞어서 해시 값을 만듬.

- 여기서 확장되어 SHAKE128 SHAKE256과 같이 확장된 길이의 해시 값을 출력할 수 있는 XOF가 포함됨.
  - XOF(eXtendable Output Function) 원래는 고정된 길이의 출력 값을 생성해야 하지만, 출력 값의 길이를 원하는 대로 조절할 수 있는 함수
  - FIPS 202는 SHA3에 대한 표준이지만, SHA3에서 XOF 기능이 있는 SHAKE도 포함됨.

2. FIPS 202 RUST 전환

- 기존 계획
  - KpqC 알고리즘에서는 시드 값과 난수 생성을 위해서 FIPS 202를 활용하고 있음.
  - 기존의 FIPS202의 c 구현을 rust로 바꿔서 FFI 기능을 통해서 C로 구현된 KpqC 알고리즘에 적용.



2. FIPS 202 RUST 전환

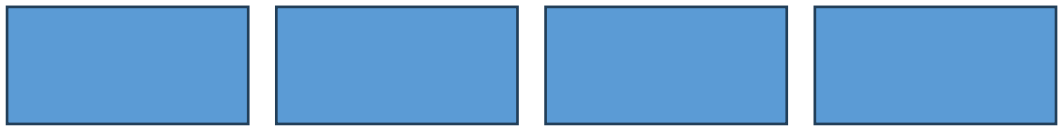
- 구조체 정의
  - Shake의 구조체는 init 함수에서 동적으로 메모리를 할당해줌.

```c
#define SHAKE128_RATE 168
#define SHAKE256_RATE 136
#define SHA3_256_RATE 136
#define SHA3_384_RATE 104
#define SHA3_512_RATE 72

#define PQC_SHAKEINCCTX_BYTES (sizeof(uint64_t)*26)
#define PQC_SHAKECTX_BYTES (sizeof(uint64_t)*25)

// Context for incremental API
typedef struct {
    uint64_t *ctx;
} shake128incctx;

void shake128_inc_init(shake128incctx *state) {
    state->ctx = malloc(PQC_SHAKEINCCTX_BYTES);
    if (state->ctx == NULL) {
        exit(111);
    }
    keccak_inc_init(state->ctx);
}
```

```rust
pub const NROUNDS: usize = 24;
pub const SHAKE128_RATE: usize = 168;
pub const SHAKE256_RATE: usize = 136;
pub const SHA3_256_RATE: usize = 136;
pub const SHA3_384_RATE: usize = 104;
pub const SHA3_512_RATE: usize =  72;

pub const PQC_SHAKEINCCTX_BYTES: usize = std::mem::size_of::<u64>() * 26;
pub const PQC_SHAKECTX_BYTES: usize = std::mem::size_of::<u64>() * 25;

pub struct shake128incctx {
    pub ctx: Option<Box<[u64]>>,
}

impl shake128incctx {
    pub fn new() -> Self {
        shake128incctx {
            ctx: Some(vec![0u64; PQC_SHAKEINCCTX_BYTES].into_boxed_slice()),
        }
    }
}
```

2. FIPS 202 RUST 전환

- Option<Box<[u64]>>
  - Option<> : 값이 있을 수도 있고, 없을 수도 있음을 표현해야 할때.
    - 러스트에서는 초기화 되지 않은 값에 대해서는 에러를 발생시켜 동작되지 않음.
    - Option<>을 사용해서 값이 없음을 표현함으로써 해결 가능
  - Box<> : 힙에 데이터를 저장하고 그 포인터를 제공.
    - C의 포인터와 동일함. 하지만 포인터 연산은 할 수 없음.
    - Vec![]와의 차이는 크기가 고정됨.



2. FIPS 202 RUST 전환

- Option〈〉과 Box〈〉가 C랑 호환이 안됨.
- 포인터를 사용하기 위해서는 원시 포인터를 사용해야 함
  - *mut u64 와 같이. 하지만 unsafe라서 목적과 다름.

- Unsafe를 완전히 없앨 수 없음.
  - 원시 포인터가 필요하고, 이를 변환하여 rust에서 사용해야 함. 변화하는 작업 자체가 unsafe임.

```c
void shake128_inc_init(shake128incctx *state) {
    state->ctx = malloc(PQC_SHAKEINCCTX_BYTES);
    if (state->ctx == NULL) {
        exit(111);
    }
    keccak_inc_init(state->ctx);
}
```

```rust
pub fn shake128_inc_init(state: &mut shake128incctx){
    state.ctx = Some(vec![0u64; PQC_SHAKEINCCTX_BYTES].into_boxed_slice());

    if let Some(ctx) = state.ctx.as_mut() {
        keccak_inc_init(ctx);
    }
}
```

2. FIPS 202 RUST 전환

```rust
pub fn shake128_inc_init(state: &mut shake128incctx){
    state.ctx = Some(vec![0u64; PQC_SHAKEINCCTX_BYTES].into_boxed_slice());

    if let Some(ctx) = state.ctx.as_mut() {
        keccak_inc_init(ctx);
    }
}
```

```rust
#[no_mangle]
pub extern "C" fn shake128_inc_init(state: *mut shake128incctx) {
    let state = unsafe { &mut *state };

    state.ctx = Some(vec![0u64; PQC_SHAKEINCCTX_BYTES].into_boxed_slice());

    if let Some(ctx) = state.ctx.as_mut() {
        keccak_inc_init(ctx);
    }
}
```

감 사 합 니 다