KPQC RUST 구현 (FFI Ver.)

https://youtu.be/wWYcDq16GZY

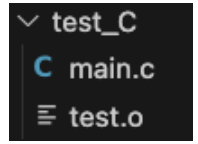
KPQC RUST 구현 (FFI Ver.)

- C로 구현된 KPQC 알고리즘을 Rust에서 동작할 수 있도록 FFI 기능을 활용하여 구현
  - FFI – Foreign Function Interface : 서로 다른 언어로 작성된 코드를 연결하는 기능
  - Rust에서는 Bindgen과 같은 도구를 사용하거나, libc crate를 사용해서 구현할 수 있음
    - Bindgen를 통해서 자동으로 바인딩코드를 생성할 수 있음
    - 바인딩 코드는 서로 다른 언어로 작성된 시스템이 서로 통신할 수 있도록 도와줌(어떻게 호출해야하는 지와 같은 정의)
      - C에서는 int 라고 작성하지만, Rust에서는 i32로 작성

FFI 기능 활용하기(Rust에서 C 코드 활용)

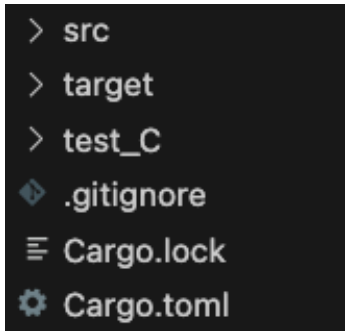
- 먼저, Rust에서 C로 구현된 함수 사용하기 예.
- C로 구현된 코드의 오브젝트(.o) 파일 추출

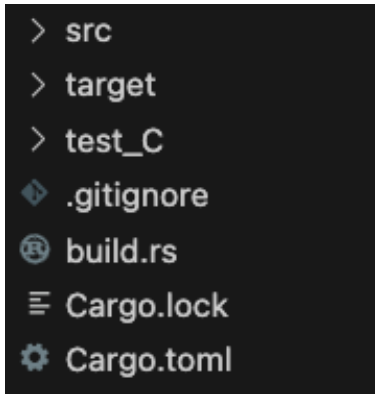
```
test_C > C main.c > ...
1   int add(int a, int b){
2       return a+b;
3   }
```




```
gcc -c -o test.o main.c
```

- Cargo.toml 파일 수정
  - Rust 프로젝트의 설정 관련된 내용이 작성된 파일
  - [build-dependencies]를 추가하고 cc="1.0" 추가
    - 빌드 스크립트(build.rs)에서 필요한 의존성 설정이 작성됨





- build.rs 파일 추가
  - 프로젝트를 빌드할 때, 커스터마이즈를 할 수 있도록 빌드 과정에서 자동으로 실행됨

```
fn main() {
    println!("cargo:rustc-link-search=native=./test_C");
    println!("cargo:rustc-link-lib=static=test.o");
}
```

```
[package]
name = "test_rust"
version = "0.1.0"
edition = "2021"

# See more keys and their def

[dependencies]

[build-dependencies]
cc = "1.0"
```

KPQC RUST 구현 (FFI Ver.)

- 정적 라이브러리 생성
  - 컴파일 시점에 프로그램에 포함되어 빌드되는 라이브러리.
    - .a (unix, linux), .lib(windows)
  - 각각의 KPQC 알고리즘 정적 라이브러리 생성.

```
CC = gcc
CFLAGS = -Wall -Wextra -O3 -I../../common -I./

# 소스 파일 목록
COMMON_SRC = ../../common/aes.c ../../common/fips202.c ../../common/randombytes.c
../../common/sha2.c
NTRU_SRC = kem.c ntt.c poly.c reduce.c symmetric.c verify.c

# 오브젝트 파일 목록
COMMON_OBJ = $(COMMON_SRC:.c=.o)
NTRU_OBJ = $(NTRU_SRC:.c=.o)

# OUTPUT_DIR = ../../../x86_lib
OUTPUT_DIR = ../../../aarch64_lib

# 정적 라이브러리 생성
$(OUTPUT_DIR)/libntruplus576.a: $(COMMON_OBJ) $(NTRU_OBJ)
    ar rcs $@ $^

# 각 .c 파일을 .o 파일로 컴파일하는 규칙
%.o: %.c
    $(CC) $(CFLAGS) -c -o $@ $<

clean:
    rm -f $(COMMON_OBJ) $(NTRU_OBJ) $(OUTPUT_DIR)/libntruplus576.a
```

KPQC RUST 구현 (FFI Ver.)

- Bindgen 사용을 위해서 cargo.toml에 build-dependencies 추가
- Build.rs를 추가하여 main.rs가 실행되기 전에 바인딩을 수행
  - 이를 위해서 cargo.toml에서 〔package〕에 build 파일 지정

```
[build-dependencies]
cc = "1.0"
bindgen = "0.59"
```

```
[package]
name = "test_kpqc"
version = "0.1.0"
edition = "2021"
build = "build.rs"
```

```rust
fn main() {
    // 세팅
    let alg_name = "nccsign";
    let category = "sign";
    let parameter: u32 = 5;

    let (temp_dir, lib_name) = match(alg_name, category, parameter) {
        ("ntru", "kem", 576) => ("-I./kpqclean/crypto_kem/NTRU+KEM576", "ntruplus576"),
        ("ntru", "kem", 768) => ("-I./kpqclean/crypto_kem/NTRU+KEM768", "ntruplus768"),
        ("ntru", "kem", 864) => ("-I./kpqclean/crypto_kem/NTRU+KEM864", "ntruplus864"),
        ("ntru", "kem", 1152) => ("-I./kpqclean/crypto_kem/NTRU+KEM1152", "ntruplus1152"),
```

```rust
// C 라이브러리가 위치한 경로를 지정
// println!("cargo:rustc-link-search=native=./x86_lib");
println!("cargo:rustc-cfg=feature=\"{}\"", alg_name);
println!("cargo:rustc-cfg=feature=\"{}\"", category);
println!("cargo:rustc-link-search=native=./aarch64_lib");
println!("cargo:rustc-link-lib=static={}", lib_name);

generate_bindings(alg_name, category, parameter, temp_dir);
```

```rust
fn generate_bindings(algorithm: &str, category: &str, param: u32, temp_path: &str)
    let headers = match (algorithm, category, param) {
        ("ntru", "kem", 576) => vec![
            "./kpqclean/crypto_kem/NTRU+KEM576/api.h",
        ],
        ("ntru", "kem", 768) => vec![
            "./kpqclean/crypto_kem/NTRU+KEM768/api.h",
        ],
        ("ntru", "kem", 864) => vec![
            "./kpqclean/crypto_kem/NTRU+KEM864/api.h",
```

```rust
let mut builder = bindgen::Builder::default();
for header in headers {
    builder = builder.header(header);
}

let bindings = builder
    .clang_arg("-I./kpqclean/common")
    .clang_arg(temp_path)
    .generate()
    .expect("Unable to generate bindings");

bindings
    .write_to_file("bin/bindings.rs")
    .expect("Couldn't write bindings!");
```

KPQC RUST 구현 (FFI Ver.)

- Bindgen을 통해서 바인딩된 bindings.rs 파일 내부

Api.h

```c
#define CRYPTO_SECRETKEYBYTES  NTRUPLUS_SECRETKEYBYTES
#define CRYPTO_PUBLICKEYBYTES  NTRUPLUS_PUBLICKEYBYTES
#define CRYPTO_CIPHERTEXTBYTES NTRUPLUS_CIPHERTEXTBYTES
#define CRYPTO_BYTES           NTRUPLUS_SSBYTES

#define CRYPTO_ALGNAME "NTRU+KEM576"

int crypto_kem_keypair(unsigned char *pk,
                       unsigned char *sk);

int crypto_kem_enc(unsigned char *ct,
                   unsigned char *ss,
                   const unsigned char *pk);

int crypto_kem_dec(unsigned char *ss,
                   const unsigned char *ct,
                   const unsigned char *sk);
```

```rust
extern "C" {
    pub fn crypto_kem_keypair(
        pk: *mut ::std::os::raw::c_uchar,
        sk: *mut ::std::os::raw::c_uchar,
    ) -> ::std::os::raw::c_int;
}
extern "C" {
    pub fn crypto_kem_enc(
        ct: *mut ::std::os::raw::c_uchar,
        ss: *mut ::std::os::raw::c_uchar,
        pk: *const ::std::os::raw::c_uchar,
    ) -> ::std::os::raw::c_int;
}
extern "C" {
    pub fn crypto_kem_dec(
        ss: *mut ::std::os::raw::c_uchar,
        ct: *const ::std::os::raw::c_uchar,
        sk: *const ::std::os::raw::c_uchar,
    ) -> ::std::os::raw::c_int;
}
```

Parmas.h

```c
#ifndef PARAMS_H
#define PARAMS_H

#define NTRUPLUS_N 576
#define NTRUPLUS_Q 3457

#define NTRUPLUS_SYMBYTES  32   /* size in bytes of hashes, and seeds */
#define NTRUPLUS_SSBYTES   32   /* size in bytes of shared key */
#define NTRUPLUS_POLYBYTES 864

#define NTRUPLUS_PUBLICKEYBYTES  NTRUPLUS_POLYBYTES
#define NTRUPLUS_SECRETKEYBYTES  ((NTRUPLUS_POLYBYTES << 1) + NTRUPLUS_SYMBYTES)
#define NTRUPLUS_CIPHERTEXTBYTES  NTRUPLUS_POLYBYTES

#endif
```

```rust
pub const NTRUPLUS_N: u32 = 576;
pub const NTRUPLUS_Q: u32 = 3457;
pub const NTRUPLUS_SYMBYTES: u32 = 32;
pub const NTRUPLUS_SSBYTES: u32 = 32;
pub const NTRUPLUS_POLYBYTES: u32 = 864;
pub const NTRUPLUS_PUBLICKEYBYTES: u32 = 864;
pub const NTRUPLUS_SECRETKEYBYTES: u32 = 1760;
pub const NTRUPLUS_CIPHERTEXTBYTES: u32 = 864;
pub const CRYPTO_SECRETKEYBYTES: u32 = 1760;
pub const CRYPTO_PUBLICKEYBYTES: u32 = 864;
pub const CRYPTO_CIPHERTEXTBYTES: u32 = 864;
pub const CRYPTO_BYTES: u32 = 32;
pub const CRYPTO_ALGNAME: &[u8; 12usize] = b"NTRU+KEM576\0";
```

KPQC RUST 구현 (FFI Ver.)

```rust
use std::env;

mod bindings {
    include!("../bin/bindings.rs");
}

use bindings::*;
```

```rust
#[cfg(feature = "ntru")]
fn kem(){
    let mut pk: [u8; CRYPTO_PUBLICKEYBYTES as usize] = [0; CRYPTO_PUBLICKEYBYTES as usize];
    let mut sk: [u8; CRYPTO_SECRETKEYBYTES as usize] = [0; CRYPTO_SECRETKEYBYTES as usize];
    let mut ct: [u8; CRYPTO_CIPHERTEXTBYTES as usize] = [0; CRYPTO_CIPHERTEXTBYTES as usize];
    let mut ss: [u8; CRYPTO_BYTES as usize] = [0; CRYPTO_BYTES as usize];
    let mut dss: [u8; CRYPTO_BYTES as usize] = [0; CRYPTO_BYTES as usize];
```

```
-----------------------------------------
★ NTRU+KEM576 All function tests start ★
-----------------------------------------
CRYPTO_ALGNAME           : NTRU+KEM576
CRYPTO_PUBLICKEYBYTES    : 864
CRYPTO_SECRETKEYBYTES    : 1760
CRYPTO_BYTES             : 32
CRYPTO_CIPHERTEXTBYTES   : 864

crypto_kem_keypair       : passed
crypto_kem_enc           : passed
crypto_kem_dec           : passed
-----------------------------------------
★ NTRU+KEM576 All function tests Done  ★
-----------------------------------------
```

```rust
let key_result;
let enc_result;
let dec_result;

let pk_ptr: *mut u8 = pk.as_mut_ptr();
let sk_ptr: *mut u8 = sk.as_mut_ptr();
let ct_ptr: *mut u8 = ct.as_mut_ptr();
let ss_ptr: *mut u8 = ss.as_mut_ptr();

unsafe {
    key_result = crypto_kem_keypair(pk_ptr, sk_ptr);
    enc_result = crypto_kem_enc(ct_ptr, ss_ptr, pk_ptr);
    dec_result = crypto_kem_dec(ss_ptr, ct_ptr, sk_ptr);
}
```

KPQC RUST(shake 구현 적용)

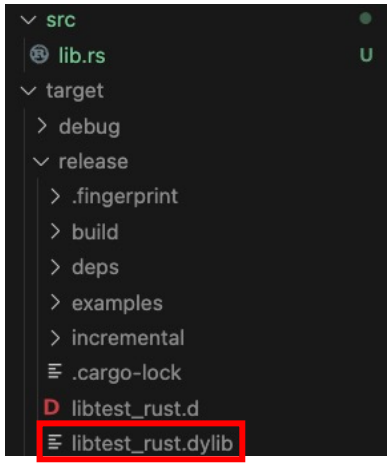
- 현재 구현은 Rust에서 C코드를 동작시키는 방법
- 하지만, 정적 라이브러리로 변환하여 사용하기 때문에 내부의 shake 만 rust로 구현하여 적용하는 것에 어려움이 있음.
- 따라서, 반대로 C에서 Rust로 구현된 코드를 적용하는 방법으로 변경

```
[lib]
crate-type = ["cdylib"]
```

C에서 사용할 수 있는 동적 라이브러리르 생성

```
#[no_mangle]
pub extern "C" fn add(a: i32, b: i32) -> i32 {
    a + b
}
```

Rust code




```
#include <stdio.h>

extern int add(int a, int b);

int main() {
    int result = add(5,3);
    printf("Result : %d\n", result);
    return 0;
}
```

```
siwooeum@SiWooui-13inchi-MacPro test_C % gcc main.c -L../target/release -ltest_rust -o main
siwooeum@SiWooui-13inchi-MacPro test_C % ./main
Result : 8
```

감 사 합 니 다