PQC – Known Answer Tests and Test Vectors

Test vectors are to be generated that can be used to determine the correctness of an implementation. These files come in two types: Known Answer Tests (KAT) files and Intermediate files. The KAT files are for general use to determine an implementation's correctness. The Intermediate values are useful for debugging an incorrect implementation. KAT files shall be provided to test different aspects of the algorithm, e.g., key generation, encryption, decryption.

Submitters should use the scripts available at http://csrc.nist.gov/groups/ST/post-quantum-crypto/example-files.html
to generate their KAT files.

In "api.h" you should declare the #define macros as specified in the API notes as well as the names of each function being implemented.

The build process and KAT creation process is essentially the same for all 3 possible cryptographic functionalities. We will use signatures as an example

e.g. if you are implementing signatures only, your api.h should be:

```
#ifndef api_h
#define api_h

//  Set these three values appropriately for your algorithm
#define CRYPTO_SECRETKEYBYTES 256
#define CRYPTO_PUBLICKEYBYTES 85
#define CRYPTO_BYTES 128

// Change the algorithm name to your algorithm
#define CRYPTO_ALGNAME "UserDefinedAlgName"

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

int
crypto_sign(unsigned char *sm, unsigned long long *smlen,
            const unsigned char *m, unsigned long long mlen,
            const unsigned char *sk);

int
crypto_sign_open(unsigned char *m, unsigned long long *mlen,
```

```
        const unsigned char *sm, unsigned long long smlen,
        const unsigned char *pk);

#endif /* api_h */
```

To build the KAT script (again, using signatures as an example), be sure that the NIST-provided
PQCgenKAT_sign.c, rng.c and rng.h are in the same directory as your definitions of api.h and sign.c.

In addition, you will need to have OpenSSL Version 1.10f (other earlier or later versions of OpenSSL may work as well but we have not tested with them) and may have to set the LDFLAGS at CFLAGS so Make can find them

In this case, the following Makefile (which should be modified in the obvious way for creating KATS for encryption algorithms and KEMs, e.g.
replacing sign with kem and encrypt as necessary) should work to build the PQCgenKAT_sign executable.

```
LDFLAGS =
CFLAGS =

OBJS = rng.o \
        sign.o \
        PQCgenKAT_sign.o

PQCgenKAT_sign : $(OBJS)
        gcc $(LDFLAGS) $(CFLAGS) -o $@ $^

%.o : %.c
        gcc $(CFLAGS) -o $@ -c $<

rng.o : rng.h
api.o : api.h
PQCgenKAT_sign.o : rng.h api.h
```

Finally, to run it (on a Unix platform), simply call

```
> ./PQCgenKAT_sign
```

The script will create 2 files,

PQCsignKAT_256.req and PQCsignKAT_256.rsp

These two output files should be included as part of your submission so we can use them for KAT testing.

It would helpful to also provide a few examples with several intermediate values for debugging purposes. An example of such an example is in the file Intermediate Values, which can also be found at the url above.