PQC - API notes

Most of the API information is derived from the **eBATS: ECRYPT Benchmarking of Asymmetric Systems** (https://bench.cr.yp.to/ebats.html). This has been done to facilitate benchmarking algorithm performance. Please look at the eBATS page for more information on how to submit an algorithm for performance benchmarking. There are two sets of API calls listed for each primitive. The first set is the API call directly from the eBATS page, or something very similar for the Key Establishment section. The second set of calls is for testing purposes. The calls extend the eBATS calls for functions that utilize randomness by providing a pointer to specify a randomness string. This will allow algorithms that utilize randomness to be able to provide reproducible results. For example, this will allow testing of KAT files.

Public-key Signatures

See https://bench.cr.yp.to/call-sign.html for more information on Public-key Signature API and performance testing.

The first thing to do is to create a file called *api.h*. This file contains the following three lines (with the sizes set to the appropriate values):

```
#define CRYPTO_SECRETKEYBYTES 256
#define CRYPTO_PUBLICKEYBYTES 85
#define CRYPTO_BYTES 128
```

Then create a file called *sign.c* with the following function calls:

eBATS calls

Generates a keypair - *pk* is the public key and *sk* is the secret key.

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

Sign a message: *sm* is the signed message, *m* is the original message, and *sk* is the secret key.

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

Verify a message signature: *m* is the original message, *sm* is the signed message, *pk* is the public key.

```
int crypto_sign_open(
     unsigned char *m, unsigned long long *mlen,
     const unsigned char *sm, unsigned long long
smlen,
     const unsigned char *pk
)
```

KAT calls

```
int crypto_sign_keypair_KAT(
     unsigned char *pk,
     unsigned char *sk,
     unsigned char *randomness
)

int crypto_sign_KAT(
     unsigned char *sm, unsigned long long *smlen,
     const unsigned char *m, unsigned long long mlen,
     const unsigned char *sk,
     unsigned char *randomness
)
```

Public-key Encryption

See https://bench.cr.yp.to/call-encrypt.html for more information on Public-key Encryption API and performance testing.

The first thing to do is to create a file called *api.h*. This file contains the following three lines (with the sizes set to the appropriate values):

```
#define CRYPTO_SECRETKEYBYTES 256
#define CRYPTO_PUBLICKEYBYTES 64
#define CRYPTO_BYTES 48
```

Then create a file called *encrypt.c* with the following function calls:

eBATS calls

Generates a keypair - *pk* is the public key and *sk* is the secret key.

```
int crypto_encrypt_keypair(
     unsigned char *pk,
     unsigned char *sk
)
```

Encrypt a plaintext: *c* is the ciphertext, *m* is the plaintext, and *pk* is the public key.

```
int crypto_encrypt(
     unsigned char *c,unsigned long long *clen,
     const unsigned char *m,unsigned long long mlen,
     const unsigned char *pk
)
```

Decrypt a ciphertext: *m* is the plaintext, *c* is the ciphertext, and *sk* is the secret key.

```
int crypto_encrypt_open(
     unsigned char *m,unsigned long long *mlen,
     const unsigned char *c,unsigned long long clen,
     const unsigned char *sk
)
```

KAT calls

```
int crypto_encrypt_keypair_KAT(
     unsigned char *pk,
     unsigned char *sk,
     unsigned char *randomness
)

int crypto_encrypt_KAT(
     unsigned char *c,unsigned long long *clen,
     const unsigned char *m,unsigned long long mlen,
     const unsigned char *pk,
     unsigned char *randomness
)
```

Key Establishment

The calls in the eBATS specification do not meet the calls specified in the call for algorithms. However, attempts were made to match the specifications for the other algorithms. (For reference, see https://bench.cr.yp.to/call-dh.html for more information on Public-key Diffie-Hellman API and performance testing.)

The first thing to do is to create a file called *api.h*. This file contains the following three lines (with the sizes set to the appropriate values):

```
#define CRYPTO_SECRETKEYBYTES 192
#define CRYPTO_PUBLICKEYBYTES 64
#define CRYPTO_BYTES 64
```

Then create a file called *keyestablishment.c* with the following function calls:

eBATS-like calls

Generate an *initiator* key-establishment message: *kei* is the *initiator's* key exchange message and *ski* is the secret key of the *initiator*.

```
int crypto_keyestablishment_initiator_generate(
     unsigned char *kei,
     unsigned char *ski
)
```

Generate a *responder* key-establishment message: *ker* is the responder's key exchange message, *skr* is the *responder's* secret key, and *kei i*s the *initiator's* key exchange message.

```
int crypto_keyestablishment_responder_generate(
     unsigned char *ker,
     unsigned char *skr,
     const unsigned char *kei
)
```

*Initiator* recovery of the shared secret: *ss* is the shared secret, *ker* is the *responder's* key exchange message, and *ski* is secret key of *initiator*.

```
int crypto_keyestablishment_initiator_recover(
     unsigned char *ss,
     const unsigned char *ker,
     const unsigned char *ski
)
```

*Responder* recovery of the shared secret: *ss* is the shared secret, *kei* is the *initiator's* key exchange message, and *skr* is secret key of *responder*.

```
int crypto_keyestablishment_responder_recover(
     unsigned char *ss,
     const unsigned char *kei,
     const unsigned char *skr
)
```

KAT calls

```
int crypto_keyestablishment_initiator_generate_KAT(
     unsigned char *kei,
     unsigned char *ski,
```

```
     unsigned char *randomness
)

int crypto_keyestablishment_responder_generate_KAT(
     unsigned char *ker,
     unsigned char *skr,
     const unsigned char *kei,
     unsigned char *randomness
)
```