Integrate RLCE into openSSL/OQS

Step 1: fork https://github.com/open-quantum-safe/openssl
Step 2: add file “openssl/oqs-template/generate.yml” by adding line 41—45:

```
family: 'RLCE'
    name_group: 'rlce'
    nid: '0x024F'
    nid_hybrid: '0x2FFE'
    oqs_alg: 'OQS_KEM_alg_RLCE'
  -
```

```
40    -
41    family: 'RLCE'
42      name_group: 'rlce'
43      nid: '0x024F'
44      nid_hybrid: '0x2FFE'
45      oqs_alg: 'OQS_KEM_alg_RLCE'
46    -
47      family: 'BIKE'
```

Step 3: add two lines to “openssl/apps/s_cb.c”

```
478     case 0x0246: return "ntru_hrss1373";
479     case 0x024F: return "rlce";
480     case 0x0238: return "bikel1";
```

```
504     case 0x2F46: return "p521_ntru_hrss1373 hybrid";
505     case 0x2FFE: return "p256_rlce hybrid";
506     case 0x2F38: return "p256_bikel1 hybrid";
```

Step 4: fork the https://github.com/open-quantum-safe/liboqs
Step 5: add the following lines to “liboqs/src/kem/kem.h” and change the supported algorithms number from 33 to 34

```
/**Algorithm identifier for RLCE. */
#define OQS_KEM_alg_RLCE "RLCE"

#ifdef OQS_ENABLE_KEM_RLCE
#include <oqs/rlce.h>
#endif /* OQS_ENABLE_KEM_RLCE */
```

```
59  /** Algorithm identifier for RLCE KEM. */
60  #define OQS_KEM_alg_RLCE "RLCE"
61  /** Algorithm identifier for HQC-128 KEM. */
```

```
304 #ifdef OQS_ENABLE_KEM_RLCE
305 #include <oqs/rlce.h>
306 #endif /* OQS_ENABLE_KEM_RLCE */
307 #ifdef OQS_ENABLE_KEM_HQC
```

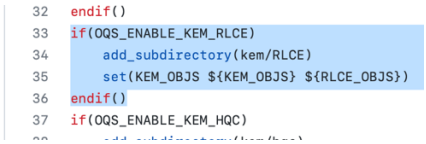
```
106 /** Number of algorithm identifiers above. */
107 #define OQS_KEM_algs_length 34
108 ///// OQS_COPY_FROM_UPSTREAM_FRAGMENT_ALGS_LENGTH_END
```

Step 6: add the following lines to “liboqs/src/kem/kem.c”

```
OQS_KEM_alg_RLCE,
……
      } else if (0 == strcasecmp(method_name, OQS_KEM_alg_RLCE)) {
#ifdef OQS_ENABLE_KEM_rlce_rlcev1
            return 1;
#else
            return 0;
#endif
……
      } else if (0 == strcasecmp(method_name, OQS_KEM_alg_RLCE)) {
#ifdef OQS_ENABLE_KEM_rlce_rlcev1
            return OQS_KEM_rlce_new();
#else
            return NULL;
#endif
```

Step 7: add the following lines to “liboqs/src/CMakeLists.txt”

```
if(OQS_ENABLE_KEM_RLCE)
    add_subdirectory(kem/RLCE)
    set(KEM_OBJS ${KEM_OBJS} ${RLCE_OBJS})
endif()
```



Step 8: add the following lines to “liboqs/CMakeLists.txt”

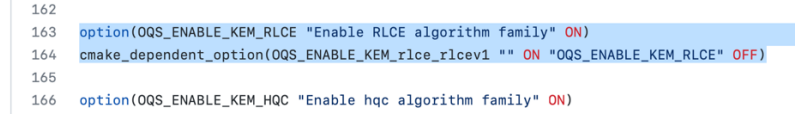
```
if(OQS_ENABLE_KEM_RLCE)
    set(PUBLIC_HEADERS ${PUBLIC_HEADERS} ${PROJECT_SOURCE_DIR}/src/kem/RLCE/rlce.h ${PROJECT_SOURCE_DIR}/src/kem/RLCE/config.h)
endif()
```

Step 9: add the following lines to “liboqs/src/oqsconfig.h.cmake”

```
#cmakedefine OQS_ENABLE_KEM_RLCE 1
#cmakedefine OQS_ENABLE_KEM_rlce_rlcev1 1
```

Step 10: add the following two lines to “liboqs/.CMake/alg_support.cmake”

```
option(OQS_ENABLE_KEM_RLCE "Enable RLCE algorithm family" ON)
cmake_dependent_option(OQS_ENABLE_KEM_rlce_rlcev1 "" ON "OQS_ENABLE_KEM_RLCE" OFF)
```



Step 11: add the following line to the file “liboqs/tests/KATs/kem/kats.json”

```
"RLCE": "e25acd9fcfd3bdcd09f4d8f1bc18cad9dcbbb119a49459a70eacfe51012cbbcd",
```

Step 12: add the following two lines to “liboqs/tests/CMakeLists.txt” [as line 67/68]

```
add_executable(example_kem_rlce example_kem_rlce.c)
target_link_libraries(example_kem_rlce PRIVATE ${API_TEST_DEPS})
```

Step 13: change the line 9 of “liboqs/tests/test_cmdline.py” by add ‘example_kem_rlce’.

```
@pytest.mark.parametrize('program', ['example_kem', 'example_sig', 'example_kem_rlce'])
```

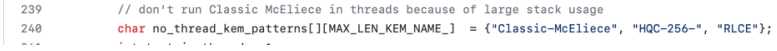
Step 14: create a file “liboqs/tests/example_kem_rlce.c” with content:

```
/*
 * example_kem.c
 *
 * Minimal example of a Diffie-Hellman-style post-quantum key encapsulation
 * implemented in liboqs.
 *
```

```
 * SPDX-License-Identifier: MIT
 */

#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <oqs/oqs.h>

/* Cleaning up memory etc */
void cleanup_stack(uint8_t *secret_key, size_t secret_key_len,
                   uint8_t *shared_secret_e, uint8_t *shared_secret_d,
                   size_t shared_secret_len);

void cleanup_heap(uint8_t *secret_key, uint8_t *shared_secret_e,
                  uint8_t *shared_secret_d, uint8_t *public_key,
                  uint8_t *ciphertext, OQS_KEM *kem);

static OQS_STATUS example_stack(void) {
#ifndef OQS_ENABLE_KEM_rlce_rlcev1 // if RLCE was not enabled at compile-time
	printf("[example_stack] OQS_ENABLE_KEM_rlce_rlcev1 was not enabled at "
	       "compile-time.\n");
	return OQS_ERROR;
#else
	uint8_t public_key[OQS_KEM_RLCE_length_public_key];
	uint8_t secret_key[OQS_KEM_RLCE_length_secret_key];
	uint8_t ciphertext[OQS_KEM_RLCE_length_ciphertext];
	uint8_t shared_secret_e[OQS_KEM_RLCE_length_shared_secret];
	uint8_t shared_secret_d[OQS_KEM_RLCE_length_shared_secret];

	OQS_STATUS rc = crypto_kem_keygenerate(public_key, secret_key);
	if (rc != OQS_SUCCESS) {
		fprintf(stderr, "ERROR: crypto_kem_keygenerate failed!\n");
		cleanup_stack(secret_key, OQS_KEM_RLCE_length_secret_key,
		              shared_secret_e, shared_secret_d,
		              OQS_KEM_RLCE_length_shared_secret);

		return OQS_ERROR;
	}
	rc = crypto_kem_encapsulate(ciphertext, shared_secret_e, public_key);
	if (rc != OQS_SUCCESS) {
		fprintf(stderr, "ERROR: crypto_kem_encapsulate failed!\n");
		cleanup_stack(secret_key, OQS_KEM_RLCE_length_secret_key,
		              shared_secret_e, shared_secret_d,
		              OQS_KEM_RLCE_length_shared_secret);

		return OQS_ERROR;
	}
	rc = crypto_kem_decapsulate(shared_secret_d, ciphertext, secret_key);
	if (rc != OQS_SUCCESS) {
		fprintf(stderr, "ERROR: crypto_kem_decapsulate failed!\n");
		cleanup_stack(secret_key, OQS_KEM_RLCE_length_secret_key,
		              shared_secret_e, shared_secret_d,
		              OQS_KEM_RLCE_length_shared_secret);

		return OQS_ERROR;
	}
	printf("[example_stack] OQS_ENABLE_KEM_rlce_rlcev1 operations completed.\n");

	return OQS_SUCCESS; // success!
#endif
}

static OQS_STATUS example_heap(void) {
	OQS_KEM *kem = NULL;
	uint8_t *public_key = NULL;
	uint8_t *secret_key = NULL;
	uint8_t *ciphertext = NULL;
	uint8_t *shared_secret_e = NULL;
	uint8_t *shared_secret_d = NULL;

	kem = OQS_KEM_new(OQS_KEM_alg_RLCE);
	if (kem == NULL) {
		printf("[example_heap]  OQS_ENABLE_KEM_rlce_rlcev1 was not enabled at "
		       "compile-time.\n");
		return OQS_ERROR;
	}

	public_key = malloc(kem->length_public_key);
	secret_key = malloc(kem->length_secret_key);
	ciphertext = malloc(kem->length_ciphertext);
	shared_secret_e = malloc(kem->length_shared_secret);
	shared_secret_d = malloc(kem->length_shared_secret);
	if ((public_key == NULL) || (secret_key == NULL) || (ciphertext == NULL) ||
```

```
                (shared_secret_e == NULL) || (shared_secret_d == NULL)) {
                fprintf(stderr, "ERROR: malloc failed!\n");
                cleanup_heap(secret_key, shared_secret_e, shared_secret_d, public_key,
                                ciphertext, kem);

                return OQS_ERROR;
        }

        OQS_STATUS rc = OQS_KEM_keypair(kem, public_key, secret_key);
        if (rc != OQS_SUCCESS) {
                fprintf(stderr, "ERROR: OQS_KEM_keypair failed!\n");
                cleanup_heap(secret_key, shared_secret_e, shared_secret_d, public_key,
                                ciphertext, kem);

                return OQS_ERROR;
        }
        rc = OQS_KEM_encaps(kem, ciphertext, shared_secret_e, public_key);
        if (rc != OQS_SUCCESS) {
                fprintf(stderr, "ERROR: OQS_KEM_encaps failed!\n");
                cleanup_heap(secret_key, shared_secret_e, shared_secret_d, public_key,
                                ciphertext, kem);

                return OQS_ERROR;
        }
        rc = OQS_KEM_decaps(kem, shared_secret_d, ciphertext, secret_key);
        if (rc != OQS_SUCCESS) {
                fprintf(stderr, "ERROR: OQS_KEM_decaps failed!\n");
                cleanup_heap(secret_key, shared_secret_e, shared_secret_d, public_key,
                                ciphertext, kem);

                return OQS_ERROR;
        }

        printf("[example_heap]  OQS_ENABLE_KEM_rlce_rlcev1 operations completed.\n");
        cleanup_heap(secret_key, shared_secret_e, shared_secret_d, public_key,
                        ciphertext, kem);

        return OQS_SUCCESS; // success
}

int main(void) {
        if (example_stack() == OQS_SUCCESS && example_heap() == OQS_SUCCESS) {
                return EXIT_SUCCESS;
        } else {
                return EXIT_FAILURE;
        }
}

void cleanup_stack(uint8_t *secret_key, size_t secret_key_len,
                        uint8_t *shared_secret_e, uint8_t *shared_secret_d,
                        size_t shared_secret_len) {
        OQS_MEM_cleanse(secret_key, secret_key_len);
        OQS_MEM_cleanse(shared_secret_e, shared_secret_len);
        OQS_MEM_cleanse(shared_secret_d, shared_secret_len);
}

void cleanup_heap(uint8_t *secret_key, uint8_t *shared_secret_e,
                        uint8_t *shared_secret_d, uint8_t *public_key,
                        uint8_t *ciphertext, OQS_KEM *kem) {
        if (kem != NULL) {
                OQS_MEM_secure_free(secret_key, kem->length_secret_key);
                OQS_MEM_secure_free(shared_secret_e, kem->length_shared_secret);
                OQS_MEM_secure_free(shared_secret_d, kem->length_shared_secret);
        }
        OQS_MEM_insecure_free(public_key);
        OQS_MEM_insecure_free(ciphertext);
        OQS_KEM_free(kem);
}
```

Step 15: add “RLCE” to line 240 of “liboqs/tests/test_kem.c”



```
239         // don't run Classic McEliece in threads because of large stack usage
240         char no_thread_kem_patterns[][MAX_LEN_KEM_NAME_]  = {"Classic-McEliece", "HQC-256-", "RLCE"};
```

Step 16: create a file “liboqs/docs/algorithms/kem/rlce.yml” with the following contents:

```
name: RLCE
type: kem
nist-round: 1
spec-version: NIST Round 1 submission
```

```
parameter-sets:
- name: RLCE
  claimed-nist-level: 1
  claimed-security: IND-CCA2
  length-public-key: 188001
  length-ciphertext: 988
  length-secret-key: 310116
  length-shared-secret: 64
```

Step 17: create a folder RLCE under the folder “liboqs/src/kem”.

Step 18: downloaded https://github.com/yonggewang/RLCE to local drive and upload all files within “liboqsRLCE” to the “liboqs/src/kem/RLCE” folder.

How to test the package. In AWS/GLP setup a Ubuntu instance (medium) and do the following:

```
sudo apt-get update
openssl version -v
sudo apt install cmake gcc libtool libssl-dev make ninja-build git
sudo apt-get install libtext-template-perl
sudo apt install valgrind
sudo apt-get install python3-tabulate
sudo apt-get install qemu
sudo apt-get install qemu-kvm
apt show qemu-system-x86
kvm -version
sudo apt install astyle cmake gcc ninja-build libssl-dev python3-pytest python3-pytest-xdist unzip xsltproc doxygen
graphviz python3-yaml
sudo apt-get install python3-teblate

/usr/local/lib$ sudo rm libcrypto.a
/usr/local/lib$ sudo rm liboqs.a
$cd
$rm -r liboqs
$rm -r oqs-openssl
$git clone --branch main https://github.com/jwagrunner/liboqs.git
$git clone https://github.com/jwagrunner/openssl.git oqs-openssl

//*** yongge Wang’s version: $git clone https://github.com/yonggewang/openssl.git
//*** yongge wang’s version: $git clone https://github.com/yonggewang/liboqs.git
$cd liboqs
$mkdir build && cd build
$cmake -GNinja -DCMAKE_INSTALL_PREFIX=../../openssl/oqs ..
$ninja
$ninja install
$ninja run_tests
~liboqs/build/test$./test_kem RLCE
~/oqs-openssl$ export LIBOQS_DOCS_DIR=$HOME/liboqs/docs
~/oqs-openssl$ python3 oqs-template/generate.py

~oqs-openssl$./Configure no-shared linux-x86_64 -lm -DOQS_DEFAULT_GROUPS=\”X25519:kyber512:ED448\”
~oqs-openssl$make generate_crypto_objects
~oqs-openssl $make
~oqs-openssl $make test
~oqs-openssl $sudo make install

~/liboqs/build/tests$ ./test_kem rlce
~/liboqs/build/tests$ ./test_kem_mem RLCE 1
~/liboqs/build/tests$ ./test_kem_mem RLCE 2
~/liboqs/build/tests$ ./speed_kem
~/liboqs/build/tests$ ./kat_kem RLCE
~/liboqs/build/tests$ ./example_kem_rlce

~/oqs-openssl$ apps/openssl req -x509 -new -newkey dilithium2 -keyout dilithium2_CA.key -out dilithium2_CA.crt -nodes -
subj “/CN=oqstest CA” -days 365 -config apps/openssl.cnf
~/oqs-openssl$ apps/openssl req -new -newkey dilithium2 -keyout dilithium2_srv.key -out dilithium2_srv.csr -nodes -subj
“/CN=oqstest server” -config apps/openssl.cnf
~/oqs-openssl$ apps/openssl x509 -req -in dilithium2_srv.csr -out dilithium2_srv.crt -CA dilithium2_CA.crt -CAkey
dilithium2_CA.key -CAcreateserial -days 365
~/oqs-openssl$ apps/openssl s_server -cert dilithium2_srv.crt -key dilithium2_srv.key -www -tls1_3

From another machine:
~/oqs-openssl$ apps/openssl s_client -groups rlce  -CAfile dilithium2_CA.crt
~/oqs-openssl$ apps/openssl s_client -groups kyber512  -CAfile dilithium2_CA.crt
~/oqs-openssl$ apps/openssl  speed oqskem
```

```
~/oqs-openssl$ apps/openssl  speed rlce
```