Integrating Quantum-resistant encryption method RLCE with OpenSSL 1.0.2

Date: 11 February 2018

Author: Mugove T. Kangai

Introduction

The Open Quantum Safe project provides a framework with which to integrate OpenSSL 1.0.2 with a quantum-resistant encryption scheme. Termed LIBOQS or C Library for quantum-resistant algorithms, it is freely available on GitHub. Created by Douglas Stebila (2018), it provides an API for fast and easy integration with OpenSSL.

In theory, LIBOQS should make it easy to integrate with OpenSSL. The reality is that adding a new cryptographic method to LIBOQS requires several bolt-on steps on OpenSSL. This adds to the complexity of such a task, which needs a deep knowledge of OpenSSL, while contending with a lack of information in the form of user guides and examples on how to go about such a task. The purpose of this article is to act as a systematic guide on how to achieve integration.

The author had to follow through several git commits to arrive at a method of adding an encryption scheme to OpenSSL 1.0.2. This article will show how to add Random Linear Code Encryption (RLCE), devised by Wang (2015), to OpenSSL by utilising the LIBOQS library.

Openssl 1.0.2 contains sub-folders such as certs, crypto, ssl, engines, apps, test, util and tools. The LIBOQS project involves modifying files within the ssl and apps folders while creating an extra vendor folder where the quantum crypts reside. Building the project requires a c compiler such as **gcc** as well as **automake**, **autoconf** and **libtool** tools.

Adding LIBOQS with RLCE to OpenSSL

- Step 1: First, clone the OpenSSL git repository available at https://github.com/open-quantum-safe/openssl.git. Utilise the branch labelled **OpenSSL_1_0_2-stable.** This copy of OpenSSL comes with other quantum resistant algorithms bundled in such as NTRU, Kyber and Frodo among others. These reside in the **liboqs** subfolder of the **vendor** folder. This is the location of new quantum resistant algorithms. Accordingly, update the Makefile.org to include files in this folder during the build process. This has the effect of defining the contents of the library which are referenced in the code-snippet of ‘Makefile.org’as:

```
build_libs: build_liboqs build_libcrypto build_libssl openssl.pc

build_liboqs:
  cd vendor/liboqs && $(MAKE)

LIBOQS_HEADERS = aes.h kex_lwe_frodo.h common.h
rand_urandom_chacha20.h kex.h rand.h sha3.h kex_rlce.h

links:
  @$(PERL) $(TOP)/util/mkdir-p.pl include/openssl
  @$(PERL) $(TOP)/util/mklink.pl include/openssl $(EXHEADER)
  @set -e; target=links; $(RECURSIVE_BUILD_CMD)
  cd vendor/liboqs && $(MAKE) links
  @$(PERL) $(TOP)/util/mkdir-p.pl include/oqs
  rm -f include/oqs/*.h
  for i in $(LIBOQS_HEADERS) ;\
  do \
  ln -s ../../vendor/liboqs/include/oqs/$$i include/oqs; \
  done
```

Note that the Makefile creates a symbolic link, note the 'ln -s' command, such that the include statement is useable with the angle brackets '< >',

```
#include <oqs/kex_rlce.h>
```

, is available for use.

- Add the RLCE algorithm files into the corresponding folder as shown in commit 617f3b at https://github.com/mutapa/liboqs/commit/617f3b.
- Update the Makefile.am file as shown in commit 7b837f viewable at https://github.com/mutapa/liboqs/commit/7b837f. This update links the source files to the rest of the OpenSSL application.

- Step 2: Skip this step unless working with a fresh copy of OpenSSL, as it is something that comes as part of the LIBOQS library. Update OpenSSL by adding the OQS_KEX to the integration point contained in apps/speed.c file. The **apps** folder contains a Makefile that is responsible for creating the openssl.exe program. Modification involves adding two lines as shown in the following code snippet. Changes to the speed.c are contained in this git commit: https://github.com/open-quantum-safe/openssl/commit/dc462f.

  apps/Makefile

  DLIBCRYPTO=../libcrypto.a ../vendor/liboqs/liboqs.a

  LIBCRYPTO=-L.. -lcrypto -L../vendor/liboqs –loqs

- Step 3: The **speed.c** containing LIBOQS contains references to the **kex.h** and **rand.h**, which respectively are the prototypes for the key exchange mechanism and the randomisation method imported in the project during step 1. Initialisation and usage of the **kex.h** objects and methods for key exchange occurs within speed.c as shown in the code snippet below. The changes made to **apps/speed.c** file are available in the change-set in commit hash: 45b8c0. To access the files, use GitHub link

https://github.com/mutapa/liboqs/commit/45b8c0be7b8c6c87466961d5cf00d6d509b0c795. The **speed.c** allows for recording of the encryption speed of an algorithm and allows for their benchmarking. This is useful for testing the effectiveness of RLCE against other encryption methods. The Makefile here adds the **oqs** library to the OpenSSL build as shown in this code snippet.

-

apps/speed.c

```
# define OQSKEX_NUM     6
# ifndef OPENSSL_NO_OQSKEX
    OQS_KEX *oqskex_kex[OQSKEX_NUM];
    void *oqskex_alice_priv[OQSKEX_NUM];
    unsigned char *oqskex_alice_msg[OQSKEX_NUM];
    size_t oqskex_alice_msg_len[OQSKEX_NUM];
    unsigned char *oqskex_bob_msg[OQSKEX_NUM];
    size_t oqskex_bob_msg_len[OQSKEX_NUM];
    unsigned char *oqskex_alice_session_key;
    size_t oqskex_alice_session_key_len;
    unsigned char *oqskex_bob_session_key;
    size_t oqskex_bob_session_key_len;
    OQS_RAND *oqskex_rand[OQSKEX_NUM];
    long oqskex_c[OQSKEX_NUM][3];
# endif
```

An explanation of speed.c:

- ✓ # define OQSKEX_NUM 6: is the number of key exchange methods integrated into the library
- ✓ OQS_KEX *oqskex_kex[OQSKEX_NUM]: allows us to specify a key exchange object, OQS_ oqskex, to use by supplying a number between 0 and OQSKEX_NUM-1.
- ✓ Loop through the array of OQS encryption algorithms and for each one create a new oqskex_kex object

```
else if (j == R_OQSKEX_RLCE) {
                            oqskex_kex[j] = OQS_KEX_new(oqskex_rand[j],
OQS_KEX_alg_rlce, NULL, 0, NULL);
                        }
```

- ✓ Time the steps for Alice performing an encryption as shown below by wrapping the code to time between TIME_F(START) and TIME_F(STOP) method calls.

```
/* time OQSKEX Alice 0 operation */
                char lbl[1000];
                sprintf(lbl, (j == R_OQSKEX_GENERIC) ? "OQS KEX generic (%s)" : "OQS
KEX %s", oqskex_kex[j]->method_name);
                pkey_print_message(lbl, "Alice 0", oqskex_c[j][0], 0, OQSKEX_SECONDS);
                Time_F(START);
                for (count = 0, run = 1; COND(oqskex_c[j][0]); count++) {
                    OQS_KEX_alice_0(oqskex_kex[j], &(oqskex_alice_priv[j]),
&(oqskex_alice_msg[j]), &(oqskex_alice_msg_len[j]));
                    OQS_KEX_alice_priv_free(oqskex_kex[j], oqskex_alice_priv[j]);
                    oqskex_alice_priv[j] = NULL;
                    free(oqskex_alice_msg[j]);
                    oqskex_alice_msg[j] = NULL;
                }
                d = Time_F(STOP);
```

- ✓ Time the steps for Bob performing a decryption as shown below by wrapping the code between TIME_F(START) and TIME_F(STOP) functions.

```
/* time OQSKEX Bob operation */
                sprintf(lbl, (j == R_OQSKEX_GENERIC) ? "OQS KEX generic (%s)" : "OQS
KEX %s", oqskex_kex[j]->method_name);
                pkey_print_message(lbl, "Bob", oqskex_c[j][1], 0, OQSKEX_SECONDS);
                Time_F(START);
                for (count = 0, run = 1; COND(oqskex_c[j][1]); count++) {
                    OQS_KEX_bob(oqskex_kex[j], oqskex_alice_msg[j],
oqskex_alice_msg_len[j], &(oqskex_bob_msg[j]), &(oqskex_bob_msg_len[j]),
&oqskex_bob_session_key, &oqskex_bob_session_key_len);
                    free(oqskex_bob_msg[j]);
                    oqskex_bob_msg[j] = NULL;
                    free(oqskex_bob_session_key);
                    oqskex_bob_session_key = NULL;
                    }
                d = Time_F(STOP);
```

- Step 4: Add RLCE cipher suite to the libssl library of OpenSSL. This requires the creation of ciphers for the new encryption algorithm. Commits 1be1ca and eb86dd show how to include RLCE ciphers in the OpenSSL library and are viewable at https://github.com/mutapa/liboqs/commit/1be1ca and https://github.com/mutapa/liboqs/commit/eb86dd respectively. Changes are made in the ssl folder to sources files **s3_clnt.c, s3_lib.c** (contains cipher suites), **s3_srvr.c**, **ssl.h**, **ssl_ciph.c** and **ssl_lib.c** as well as header files **ssl_loc1.h** and **tls1.h**.

  Eronen and Tschofenig (2005), in RFC4279, describe a set of cipehrsuites that are used to communicate over TLS. Using the same principals, eight ciphers are added to the OpenSSL library. The ciphers are split into two groups: those that use the quantum encryption and those without and use EECDH. The file **s3_lib.c** contains the additons as shown below:

```
/* Start add RLCE Cipher Suite */
#ifndef OPENSSL_NO_OQSKEX
        /* Cipher FF60 */
        {
                1,
                TLS1_TXT_OQSKEX_RLCE_RSA_WITH_AES_128_GCM_SHA256,
                TLS1_CK_OQSKEX_RLCE_RSA_WITH_AES_128_GCM_SHA256,
                SSL_kOQSKEX_RLCE,
                SSL_aRSA,
                SSL_AES128GCM,
                SSL_AEAD,
                SSL_TLSV1_2,
                SSL_NOT_EXP | SSL_HIGH,
                SSL_HANDSHAKE_MAC_SHA256 | TLS1_PRF_SHA256,
                128,
                128,
        },
        /* Cipher FF61 */
        {
                1,
                TLS1_TXT_OQSKEX_RLCE_ECDSA_WITH_AES_128_GCM_SHA256,
                TLS1_CK_OQSKEX_RLCE_ECDSA_WITH_AES_128_GCM_SHA256,
                SSL_kOQSKEX_RLCE,
                SSL_aECDSA,
                SSL_AES128GCM,
                SSL_AEAD,
                SSL_TLSV1_2,
                SSL_NOT_EXP | SSL_HIGH,
                SSL_HANDSHAKE_MAC_SHA256 | TLS1_PRF_SHA256,
                128,
                128,
        },
        /* Cipher FF62 */
        {
                1,
                TLS1_TXT_OQSKEX_RLCE_RSA_WITH_AES_256_GCM_SHA384,
                TLS1_CK_OQSKEX_RLCE_RSA_WITH_AES_256_GCM_SHA384,
                SSL_kOQSKEX_RLCE,
                SSL_aRSA,
                SSL_AES256GCM,
                SSL_AEAD,
                SSL_TLSV1_2,
                SSL_NOT_EXP | SSL_HIGH,
                SSL_HANDSHAKE_MAC_SHA384 | TLS1_PRF_SHA384,
                256,
                256,
        },
        /* Cipher FF63 */
        {
                1,
                TLS1_TXT_OQSKEX_RLCE_ECDSA_WITH_AES_256_GCM_SHA384,
                TLS1_CK_OQSKEX_RLCE_ECDSA_WITH_AES_256_GCM_SHA384,
                SSL_kOQSKEX_RLCE,
                SSL_aECDSA,
                SSL_AES256GCM,
                SSL_AEAD,
                SSL_TLSV1_2,
                SSL_NOT_EXP | SSL_HIGH,
                SSL_HANDSHAKE_MAC_SHA384 | TLS1_PRF_SHA384,
                256,
                256,
        },
```

Define the cipher suites within the **tls1.h** header file. View the RLCE definitions in the diagram below.

```
# define TLS1_TXT_OQSKEX_RLCE_RSA_WITH_AES_128_GCM_SHA256         "OQSKEX-RLCE-RSA-
AES128-GCM-SHA256"
# define TLS1_TXT_OQSKEX_RLCE_ECDSA_WITH_AES_128_GCM_SHA256       "OQSKEX-RLCE-ECDSA-
AES128-GCM-SHA256"
# define TLS1_TXT_OQSKEX_RLCE_RSA_WITH_AES_256_GCM_SHA384         "OQSKEX-RLCE-RSA-
AES256-GCM-SHA384"
# define TLS1_TXT_OQSKEX_RLCE_ECDSA_WITH_AES_256_GCM_SHA384       "OQSKEX-RLCE-ECDSA-
AES256-GCM-SHA384"
# define TLS1_TXT_OQSKEX_RLCE_ECDHE_RSA_WITH_AES_128_GCM_SHA256        "OQSKEX-RLCE-
ECDHE-RSA-AES128-GCM-SHA256"
# define TLS1_TXT_OQSKEX_RLCE_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256      "OQSKEX-RLCE-
ECDHE-ECDSA-AES128-GCM-SHA256"
# define TLS1_TXT_OQSKEX_RLCE_ECDHE_RSA_WITH_AES_256_GCM_SHA384        "OQSKEX-RLCE-
ECDHE-RSA-AES256-GCM-SHA384"
# define TLS1_TXT_OQSKEX_RLCE_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384      "OQSKEX-RLCE-
ECDHE-ECDSA-AES256-GCM-SHA384"



# define TLS1_CK_OQSKEX_RLCE_RSA_WITH_AES_128_GCM_SHA256           0x0300FF60
# define TLS1_CK_OQSKEX_RLCE_ECDSA_WITH_AES_128_GCM_SHA256         0x0300FF61
# define TLS1_CK_OQSKEX_RLCE_RSA_WITH_AES_256_GCM_SHA384           0x0300FF62
# define TLS1_CK_OQSKEX_RLCE_ECDSA_WITH_AES_256_GCM_SHA384         0x0300FF63
# define TLS1_CK_OQSKEX_RLCE_ECDHE_RSA_WITH_AES_128_GCM_SHA256     0x0300FF64
# define TLS1_CK_OQSKEX_RLCE_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256   0x0300FF65
# define TLS1_CK_OQSKEX_RLCE_ECDHE_RSA_WITH_AES_256_GCM_SHA384     0x0300FF66
# define TLS1_CK_OQSKEX_RLCE_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384   0x0300FF67
```

Define the RLCE bitmasks in **ssl_locl.h** as shown here:

```
# define SSL_kOQSKEX_RLCE 0x00040000L
```

Bitmasks are required to support the SSL cipher suite created above. This step sees these bitmasks set in the **ssl_lib.c** file. Only one bit per mask can be set. In this case, use the mask_k and emask_k for the RLCE algorithm as shown below. A bitwise inclusive OR of mask_k and SSL_kOQSKEX_RLCE is performed with the result stored in mask_k. Likewise, a bitwise inclusive OR of emask_k is made with SSL_kOQSKEX_RLCE.

```
mask_k |= SSL_kOQSKEX_RLCE;
emask_k |= SSL_kOQSKEX_RLCE;
```

Capture in **ssl_ciph.c** the SSL cipher aliases as shown here. Then to the switch statement for algorithm selection, add the reference for RLCE.

```
{ 0, SSL_TXT_OQSKEX_RLCE, 0, SSL_kOQSKEX_RLCE, ~SSL_aNULL, 0, 0, 0, 0, 0, 0, 0 },
{ 0, SSL_TXT_OQSKEX_RLCE_ECDHE, 0, SSL_kEECDH | SSL_kOQSKEX_RLCE, ~SSL_aNULL, 0, 0, 0,
0, 0, 0, 0 },


switch (alg_mkey) {

      . . .

       case SSL_kOQSKEX_RLCE:
              kx = "OQSKEX-RLCE";
              break;

}
```

To the **ssl.h** header file, add the following defines which act as aliases for the algorithm name:

```
# define SSL_TXT_kOQSKEX_RLCE "kOQSKEX-RLCE"

# define SSL_TXT_OQSKEX_RLCE        "OQSKEX-RLCE"
# define SSL_TXT_OQSKEX_RLCE_ECDHE  "OQSKEX-RLCE-ECDHE"
```

At this point, modify the ssl3_get_key_exchange(SSL *) method by adding to **s3_clnt.c,** which represents the client, the retrieval of the keys from the server using.

```
        if ((srvr_oqskex_msg = malloc(srvr_oqskex_msg_len)) == NULL) {
            SSLerr(SSL_F_SSL3_GET_KEY_EXCHANGE,ERR_R_MALLOC_FAILURE);
            goto err;
        }
```

Also modify the ssl3_send_client_key_exchange(SSL *) method to incorporate RLCE methods which first creates the oqskex_kex object.

```
else if (alg_k & SSL_kOQSKEX_RLCE) {
                                    if ((oqskex_kex = OQS_KEX_new(oqskex_rand,
OQS_KEX_alg_rlce, NULL, 0, NULL)) == NULL) {
                                            SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE,
ERR_R_MALLOC_FAILURE);
                                            goto err;
                                    }
                            }
```

Use the oqskex_kex object, once initialised and call the LIBOQS method for Bob:

```
if (OQS_KEX_bob(oqskex_kex, srvr_oqskex_msg, srvr_oqskex_msg_len, &clnt_oqskex_msg,
&clnt_oqskex_msg_len, &pprime_oqskex, &nprime_oqskex) != 1) {
                    SSLerr(SSL_F_SSL3_SEND_CLIENT_KEY_EXCHANGE, ERR_R_INTERNAL_ERROR);
                    goto err;
                }
```

Finally, modify the **s3_srvr.c** prototype, which represents the server, to include RLCE algorithms. The first method to update is the ssl3_send_server_key_exchange(SSL *) method. Establish an ssl connection using:

```
else if (type & SSL_kOQSKEX_RLCE) {
                                    if ((s->s3->tmp.oqskex_kex = OQS_KEX_new(s->s3-
>tmp.oqskex_rand, OQS_KEX_alg_rlce, NULL, 0, NULL)) == NULL) {
                                            SSLerr(SSL_F_SSL3_SEND_SERVER_KEY_EXCHANGE,
ERR_R_MALLOC_FAILURE);
                                            goto err;
                                    }
                            }
```

Then call the Alice method as below:

```
if (OQS_KEX_alice_0(s->s3->tmp.oqskex_kex, &(s->s3->tmp.oqskex_priv),
&oqskex_srvr_msg, &oqskex_srvr_msg_len) != 1) {
                    SSLerr(SSL_F_SSL3_SEND_SERVER_KEY_EXCHANGE,ERR_R_INTERNAL_ERROR);
                    goto err;
}
```

Next, send the key using:

```
else if (type & SSL_kOQSKEX_RLCE) {
       if ((s->s3->tmp.oqskex_kex = OQS_KEX_new(s->s3->tmp.oqskex_rand,
OQS_KEX_alg_rlce, NULL, 0, NULL)) == NULL) {
                                    SSLerr(SSL_F_SSL3_SEND_SERVER_KEY_EXCHANGE,
ERR_R_MALLOC_FAILURE);
                                    goto err;
                                }
                        }
```

- Step 5: Modify **test_kex.c** source file by adding the RLCE algorithm to the list of tests to invoke. This is useful for benchmarking and ensuring that the quantum method works as intended. Commit 3bcd6c contains the source files with the changes made to the repository, which is viewable at https://github.com/mutapa/liboqs/commit/3bcd6c5dd0d615e50b357e5abc9b54f4df900bd5.
    - To **test_kex.c,** add a line to the array of test cases to run that dictates RLCE as one algorithm to initialise and run.

```
/* Add new testcases here */
struct kex_testcase kex_testcases[] = {
            . . .
#ifdef ENABLE_KEX_RLCE
    {OQS_KEX_alg_rlce, NULL, 0, NULL, "rlce", 0, 10},
#endif
};
```

- In the **travis.yml** file, enable RLCE by adding the following lines. This script is the main build define and calls the **.travis-tests.sh**

```
matrix:
 include:
 - os: linux
   compiler: gcc
   env:
   - ENABLE_KEX_RLCE=1
```

- Modify **kex.c** to include the enable define such that the RLCE algorithm is used to create a new Key Exchange Algorithm (KEX) object.

```
#ifdef ENABLE_KEX_RLCE
    case OQS_KEX_alg_rlce:
        return OQS_KEX_rlce_new(rand);
#else
        assert(0);
#endif
```

- Update **.travis-tests.sh** to enable RLCE in the tests by appending the following lines:

```
if [[ ${ENABLE_KEX_RLCE} == 1 ]];then
 enable_disable_str+=" --enable-kex-rlce"
fi
```

- Update **configure.ac** by setting the flags for RLCE.
- Modify **rlce.c** to ensure that comparisons between different types such as int and unsigned long or unsigned int have a safe comparison between types.

- Step 6: Fix the RLCE package such that compiler warnings because of duplicate methods in **aes.c** and **drbg.c** do not conflict with those in <openssl/aes.h>. Commit 8c4517 contains the change made and is viewable at, https://github.com/mutapa/liboqs/commit/8c45174dda531a5dd3bc5efcfe2c619ab653ea06.
    - At this point, the build will compile successfully. The integration is successful when the ssltests also pass.

Conclusion

The integration of OpenSSL with RLCE at the time of writing resulted in a successful build. Adding a new encryption algorithm to SSL is a difficult proposition, which requires a good knowledge of various tools. The LIBOQS library does simplify this process somewhat as there are examples of how to add other algorithms to OpenSSL. The build mechanism used in OpenSSL treats warnings as errors and so new cryptographic implementations must adhere to this strict build process. The RLCE files needed type conversions such as from **int** to **unsigned int** and renaming of conflicting methods such as **AES_encrypt** to **AES_encrypt_RLCE** to generate a successful build. The author used Travis-CI as a continuous integration system, which helped in determining the point at which the build passed compilation and testing.

References

Eronen, P. & Tschofenig, H., 2005. *Pre-Shared Key Ciphersuites for Transport Layer Security (TLS).* [Online]
Available at: https://tools.ietf.org/html/rfc4279
[Accessed 3 January 2018].

Stebila, D., 2018. *Open Quantum Safe.* [Online]
Available at: https://github.com/open-quantum-safe
[Accessed 10 February 2018].

Wang, Y., 2015. *Quantum Resistant Random Linear Code Based Public Key Encryption Scheme RLCE,* Charlotte, North Carolina: arXiv.