PQC (McElience/RLCE) in OpenSSL

Yongge Wang

UNC Charlotte

March 23, 2023

Outline

1. Code Based Cryptography: McEliece and RLCE

2. Post-Quantum Cryptography in OpenSSL

McEliece Scheme

McEliece Scheme (1978)

`Mc.KeySetup`: An $(n, k, 2t+1)$ linear Goppa code $\mathcal{C}$ with $k \times n$ generator matrix G_s. Public key: $G = SG_sP$. Private key: G_s Where S is random and P is permutation.

`Mc.Enc`$(G, \mathbf{m}, \mathbf{e})$. For a message $\mathbf{m} \in \{0, 1\}^k$, choose a random vector $\mathbf{e} \in \{0, 1\}^n$ of weight t. The cipher text $\mathbf{c} = \mathbf{m}G + \mathbf{e}$

`Mc.Dec`$(S, G_s, P, \mathbf{c})$. For a received ciphertext $\mathbf{c}$, first compute $\mathbf{c}' = \mathbf{c}P^{-1} = \mathbf{m}SG$. Next use an error-correction algorithm to recover $\mathbf{m}' = \mathbf{m}S$ and compute the message $\mathbf{m}$ as $\mathbf{m} = \mathbf{m}'S^{-1}$.

RLCE Key setup

`RLCE.KeySetup`. Let G_s be a $k \times n$ generator matrix for an $[n, k, d]$ linear code $\mathcal{C}$correcting at least t errors. Let $G_s P_1 = [\mathbf{g}_0, \cdots, \mathbf{g}_{n-1}]$ for a random permutation P_1

1. Let $G_1 = [\mathbf{g}_0, \cdots, \mathbf{g}_{n-w}, \mathbf{r}_0, \cdots, \mathbf{g}_{n-1}, \mathbf{r}_{w-1}]$ be a $k \times (n + w)$ matrix where $\mathbf{r}_i \in GF(q)^k$ are random
2. Let $A_i \in GF(q)^{2\times 2}$ be random 2×2 matrices. Let $A = \mathrm{diag}[I_{n-w}, A_0, \cdots, A_{w-1}]$ be an $(n + w) \times (n + w)$ non-singular matrix.
3. The public key: $k \times (n + w)$ matrix $G = SG_1AP_2$ and the private key: (S, G_s, P_1, P_2, A) where S is random $k \times k$ matrix and P_2 is a permutation.

RLCE Encryption/Decryption

$\texttt{RLCE.Enc}(G, \mathbf{m}, \mathbf{e})$. For a message $\mathbf{m} \in GF(q)^k$, choose $\mathbf{e} \in GF(q)^{n+w}$ of weight at most t. The cipher: $\mathbf{c} = \mathbf{m}G + \mathbf{e}$.

$\texttt{RLCE.Dec}(S, G_s, P_1, P_2, A, \mathbf{c})$. For a cipher text $\mathbf{c}$, compute

$$\mathbf{c}P_2^{-1}A^{-1} = \mathbf{m}SG_1 + \mathbf{e}P_2^{-1}A^{-1} = [c'_0, \ldots, c'_{n+w-1}].$$

Let $\mathbf{c}' = [c'_0, c'_1, \cdots, c'_{n-w}, c'_{n-w+2}, \cdots, c'_{n+w-2}] \in GF(q)^n$. Then $\mathbf{c}'P_1^{-1} = \mathbf{m}SG_s + \mathbf{e}'$ for some $\mathbf{e}' \in GF(q)^n$ of weight at most t. Using an efficient decoding algorithm, one can recover $\mathbf{m}SG_s$ from $\mathbf{c}'P_1^{-1}$. Let D be a $k \times k$ inverse matrix of SG'_s where G'_s is the first k columns of G_s. Then $\mathbf{m} = \mathbf{c}_1 D$ where $\mathbf{c}_1$ is the first k elements of $\mathbf{m}SG_s$.

Recommended parameters

RLCE

κ_c, κ_q	sk	cipher	pk
128, 80	310116	988	188001
192,110	747393	1545	450761
256,144	1773271	2640	1232001

McEliece

ID	κ_c, κ_q	sk	cipher	pk
mceliece348864[f]	**128, 80**	6452	128	261120
mceliece460896[f]	**192,110**	13568	188	524160
mceliece6688128[f]	**256,144**	13892	240	1044992
mceliece6960119[f]	**256,144**	13908	226	1047319
mceliece8192128[f]	**256,144**	14080	240	1357824

RLCE and RSA performance (milliseconds)

κ_c	RSA modulus	key setup		encryption		decryption	
		RSA	RLCE	RSA	RLCE	RSA	RLCE
128	3072	433.607	151.834	0.135540	0.360	6.576281	1.345
192	7680	9346.846	637.988	0.672769	0.776	75.075443	2.676
256	15360	80790.751	1587.330	2.498523	1.745	560.225740	9.383

libOQS and OpenSSL

- libOQS: https://openquantumsafe.org/liboqs/
- libOQS in OpenSSL
- it was not able to integrate McEliece into OpenSSL
- TLS 1.3: https://www.rfc-editor.org/rfc/rfc8446

PQC in OpenSSL – RFC 8446

```
struct {
      ProtocolVersion legacy_version = 0x0303;    /* TLS v1.2 */
      Random random;
      opaque legacy_session_id<0..32>;
      CipherSuite cipher_suites<2..2^16-2>;
      opaque legacy_compression_methods<1..2^8-1>;
      Extension extensions<8..2^16-1>;
} ClientHello;
```

- The challenge: the extension is at most 2^{16} bytes. That is, at most 65,536 bytes (65KB).
- if public key is larger than 65KB, then it will just not work!

PQC in OpenSSL – RFC 8446

- the PQC revision should work for (1) and (3) of TLS 1.3
  1. (EC)DHE: replace DH with RLCE/McELience
  2. PSK-only
  3. PSK with (EC)DHE: replace DHE with RLCE/McELience
- Implementation discussions: a client/sever can use key_share_PQC (52) or psk_key_exchange_modes_PQC (53) to send KEM ciphertexts. For short key PQC schemes, it must be included in key_share (51) or psk_key_exchange_modes (45). If long key PQC (McEliece/RLCE) is used, it must use ExtensionType 52 and 53.

PQC in OpenSSL – RFC 8446 revised

```
struct {
        ExtensionType extension_type;
        opaque extension_data<0..2^16-1>;
    } Extension;
    enum {
        server_name(0),                             /* RFC 6066 */
        max_fragment_length(1),                     /* RFC 6066 */
        status_request(5),                          /* RFC 6066 */
        supported_groups(10),                       /* RFC 8422, 7919 */
        signature_algorithms(13),                   /* RFC 8446 */
        use_srtp(14),                               /* RFC 5764 */
        heartbeat(15),                              /* RFC 6520 */
        application_layer_protocol_negotiation(16), /* RFC 7301 */
        signed_certificate_timestamp(18),           /* RFC 6962 */
        client_certificate_type(19),                /* RFC 7250 */
        server_certificate_type(20),                /* RFC 7250 */
        padding(21),                                /* RFC 7685 */
        pre_shared_key(41),                         /* RFC 8446 */
        early_data(42),                             /* RFC 8446 */
        supported_versions(43),                     /* RFC 8446 */
        cookie(44),                                 /* RFC 8446 */
        psk_key_exchange_modes(45),                 /* RFC 8446 */
        certificate_authorities(47),                /* RFC 8446 */
        oid_filters(48),                            /* RFC 8446 */
        post_handshake_auth(49),                    /* RFC 8446 */
        signature_algorithms_cert(50),              /* RFC 8446 */
        key_share(51),                              /* RFC 8446 */
        key_share_PQC (52),
        psk_key_exchange_modes_PQC (53),
        (65535)
    } ExtensionType;
```

PQC in OpenSSL – RFC 8446 revised

```
uint16 ProtocolVersion;
opaque Random[32];
uint8 CipherSuite[2];    /* Crypto suite selector */
struct {
          ExtensionType extension_type;
          select (Extension.extension_type) {
              case 52 or 53:  opaque extension_data<8..2^22-1>;
              case  default:  opaque extension_data<8..2^16-1>;
          };
} Extension

struct {
          ProtocolVersion legacy_version=0x0303; /*TLS v1.2*/
          Random random;
          opaque legacy_session_id<0..32>;
          CipherSuite cipher_suites<2..2^16-2>;
          opaque legacy_compression_methods<1..2^8-1>;
          Extension extensions<8..2^22-1>;
} ClientHello;

struct {
          ProtocolVersion legacy_version = 0x0303;    /* TLS v1.2 */
          Random random;
          opaque legacy_session_id_echo<0..32>;
          CipherSuite cipher_suite;
          uint8 legacy_compression_method = 0;
          Extension extensions<6..2^22-1>;
} ServerHello;
```

PQC in OpenSSL – RFC 8446 revised

```
enum {
        /* Elliptic Curve Groups (ECDHE) */
        secp256r1(0x0017), secp384r1(0x0018), secp521r1(0x0019),
        x25519(0x001D), x448(0x001E),

        /* Finite Field Groups (DHE) */
        ffdhe2048(0x0100), ffdhe3072(0x0101), ffdhe4096(0x0102),
        ffdhe6144(0x0103), ffdhe8192(0x0104),

        /* Reserved Code Points */
        ffdhe_private_use(0x01FC..0x01FF),
        ecdhe_private_use(0xFE00..0xFEFF),

        /* PQC */
        rlcel1 (0x024D), rlcel3 (0x024E), rlcel5 (0x024F),
        mceliece1 (0x025D),  mceliece3 (0x025E),  mceliece5 (0x025F),

        (0xFFFF)
} NamedGroup;
```

PQC in OpenSSL – RFC 8446 revised

```
struct {
    NamedGroup group;
    select (KeyShareEntry.group) {
       case rlcel1 | rlcel3 | rlcel5 :           opaque key_exchange<1..2^22-1>;
       case mceliece1 | mceliece3 | mceliece5 :  opaque key_exchange<1..2^22-1>;
       default:                                  opaque key_exchange<1..2^16-1>;
    }
} KeyShareEntry;

struct {
     KeyShareEntry client_shares<0..2^22-1>;
}  KeyShareClientHello;


struct {
      KeyShareEntry server_share<0..2^22-1>;
} KeyShareServerHello;
```

PQC in OpenSSL – RFC 8446 revised

```
enum {
    psk_ke(0), psk_dhe_ke(1), psk_dhe_ke_pqc(2), (255)
} PskKeyExchangeMode;

struct {
    PskKeyExchangeMode ke_modes<1..255>;
} PskKeyExchangeModes;
```

RFC 8446 revised: basic full TLS handshake

```
       Client                                            Server

Key  ^ ClientHello
Exch | + key_share* | key_share_PQC*
     | + signature_algorithms*
     | + psk_key_exchange_modes* | psk_key_exchange_modes_PQC*
     v + pre_shared_key*       -------->
                                                  ServerHello  ^ Key
                               + key_share*  | key_share_PQC*  | Exch
                                            + pre_shared_key*  v
                                        {EncryptedExtensions}  ^  Server
                                        {CertificateRequest*}  v  Params
                                               {Certificate*}  ^
                                         {CertificateVerify*}  | Auth
                                                   {Finished}  v
                               <--------  [Application Data*]
     ^ {Certificate*}
Auth | {CertificateVerify*}
     v {Finished}              -------->
       [Application Data]      <------->  [Application Data]


              +  Indicates noteworthy extensions sent in the
                 previously noted message.
              *  optional or situation-dependent messages/extensions
              {} Indicates messages protected using keys
                 derived from a [sender]_handshake_traffic_secret.
              [] Indicates messages protected using keys
                 derived from [sender]_application_traffic_secret_N.
```

RFC 8446 revised: Message Flow with Incorrect DHE Share

```
ClientHello
+ key_share*            -------->
+ key_share_PQC*
                                          HelloRetryRequest
                        <--------              + key_share*
                                           + key_share_PQC*
ClientHello
+ key_share             -------->
+ key_share_PQC
                                                ServerHello
                                                + key_share
                                            + key_share_PQC
                                      {EncryptedExtensions}
                                      {CertificateRequest*}
                                             {Certificate*}
                                       {CertificateVerify*}
                                                 {Finished}
                        <--------       [Application Data*]
{Certificate*}
{CertificateVerify*}
{Finished}              -------->
[Application Data]      <------->        [Application Data]
```

RFC 8446 revised: Resumption using PSK mode

```
ClientHello
+ key_share*
+ key_share_PQC*
+ pre_shared_key          -------->
                                                ServerHello
                                           + pre_shared_key
                                               + key_share*
                                           + key_share_PQC*
                                      {EncryptedExtensions}
                                                 {Finished}
                          <--------     [Application Data*]
{Finished}                -------->
[Application Data]        <-------->     [Application Data]
```

RFC 8446 revised: 0-RTT Data

```
Client                                               Server

         ClientHello
         + early_data
         + key_share*
         + key_share_PQC*
         + psk_key_exchange_modes
         + pre_shared_key
         (Application Data*)     -------->
                                                        ServerHello
                                                   + pre_shared_key
                                                       + key_share*
                                                   + key_share_PQC*
                                              {EncryptedExtensions}
                                                      + early_data*
                                                         {Finished}
                                 <--------      [Application Data*]
         (EndOfEarlyData)
         {Finished}              -------->
         [Application Data]      <------->       [Application Data]

               () Indicates messages protected using keys
                  derived from a client_early_traffic_secret.

               {} Indicates messages protected using keys
                  derived from a [sender]_handshake_traffic_secret.

               [] Indicates messages protected using keys
                  derived from [sender]_application_traffic_secret_N.
```

Experiments: integrate RLCE into libOQS

- added RLCE to libOQS
- revised openSSL with the proposed revisions
- initial testing works with all servers.

Questions

Questions?