Transport Layer Security
Internet-Draft
Intended status: Informational
Expires: 20 December 2025

J. Wagner
Y. Wang
UNC Charlotte

18 June 2025

New Key Share Extension for Classic McEliece Algorithms
draft-wagner-tls-keysharepqc-06

Abstract

RFC 8446 is modified to where another key share extension is introduced to accommodate both public keys and ciphertexts in ClientHello and ServerHello messages for post-quantum algorithms that have large public keys, including algorithms of the code-based cryptographic scheme Classic McEliece.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://jwagrunner.github.io/internet-draft/draft-wagner-tls-keysharepqc.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-wagner-tls-keysharepqc/.

Discussion of this document takes place on the Transport Layer Security mailing list (mailto:tls@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/tls/. Subscribe at https://www.ietf.org/mailman/listinfo/tls/.

Source for this draft and an issue tracker can be found at https://github.com/jwagrunner/internet-draft.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 20 December 2025.

Table of Contents

1. Introduction 2
2. Conventions and Definitions 3
3. New Key Share Extension 3
4. NamedGroup Addition for Classic McEliece 7
5. Modification to PskKeyExchangeMode structure 8
6. Hello Retry Request using New Key Share Extension 10
7. Other Use Case (RLCE Algorithm) 12
8. TLS Implementation 12
9. Summary of Changes from RFC 8446 12
10. Security Considerations 12
11. IANA Considerations 13
Acknowledgements 13
References 14
Normative References 14
Informative References 14
Authors' Addresses 16

1. Introduction

Large public key algorithms, including the code-based cryptographic algorithm family Classic McEliece (see [NIST], [DJB25], [RJM78], and [OQS24]), cannot be easily implemented in Transport Layer Security (TLS) Protocol Version 1.3 ([RF8446]) due to the current key share limitations of 65,535 Bytes. It is important to consider such uses of algorithms given that Classic McEliece is a Round 4 algorithm submitted in the National Institute of Standards and Technology (NIST) standardization process (see [PQC25]). Therefore, this document proposes a new key share that has a higher limit and is utilized in ClientHello and ServerHello messages, which is a modification of [RFC8446]. For example, if a Classic McEliece

algorithm is requested in a TLS 1.3 key exchange, this new key share extension will be constructed. However, if a classical algorithm is requested for key exchange, a normal key share extension is constructed. Thus, enabling the use of Classic McEliece algorithms to be used in TLS 1.3 key exchanges and also presenting them as an alternative option to replace classical algorithms for future protection against the threat of attackers in possession of powerful quantum computers that will break classical encryption.

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. New Key Share Extension

Based on the key share extension from [RFC8446] is introduced a new key share extension in this document, "key_share_pqc". This is reflected in this document and is represented as KeyShareEntryPQC below, based on the existing KeyShareEntry from [RFC8446]. However, this is modified along with the existing KeyShareEntry structure to test if the key exchange algorithm chosen in a TLS 1.3 connection belongs from the Classic McEliece family, and if it is, then KeyShareEntryPQC is constructed. If the opposite is true, where the key exchange algorithm is not from the Classic McEliece family, then KeyShareEntry is constructed. Note that the "key_exchange" fields are expanded in KeyShareEntryPQC to accommodate a large public key that is greater than 65,535 Bytes:

```
       struct {
          NamedGroup group;
          select (KeyShareEntry.group) {
          case classicmceliece6688128:      Empty;
          case classicmceliece6960119:      Empty;
          case classicmceliece8192128:      Empty;
          case rlcel5:                      Empty;
          case other large PQ algorithm1:   Empty;
          case other large PQ algorithm2:   Empty;
          case etc.:                        Empty;
          default:                          opaque key_exchange<1..2^16-1>;
          }
       } KeyShareEntry;

      struct {
          NamedGroup group;
          select (KeyShareEntryPQC.group) {
          case classicmceliece6688128:      opaque key_exchange<1..2^24-1>;
          case classicmceliece6960119:      opaque key_exchange<1..2^24-1>;
          case classicmceliece8192128:      opaque key_exchange<1..2^24-1>;
          case rlcel5:                      opaque key_exchange<1..2^24-1>;
          case other large PQ algorithm1:   opaque key_exchange<1..2^24-1>;
          case other large PQ algorithm2:   opaque key_exchange<1..2^24-1>;
          case etc.:                        opaque key_exchange<1..2^24-1>;
          default:                          Empty;
          }
       } KeyShareEntryPQC;
```

Note: PQ (Post-Quantum) where "other large PQ algorithm1" and "other large PQ algorithm2" and "etc." above indicates that one or more future post-quantum algorithms with large public key sizes can be added by just defining a constant for each of these post-quantum algorithms.

Another Note: An additional algorithm is included in the above, "rlcel5", since it also has a large public key beyond the 65,535 Byte limit. See Section 7 for more information discussing this RLCE algorithm.

This is then applied to the existing KeyShareClientHello structure, which originates from [RFC8446], that now contains an additional field for KeyShareEntryPQC:

```
       struct {
          KeyShareEntry client_shares<0..2^16-1>;
          KeyShareEntryPQC client_shares<0..2^24-1>;
       } KeyShareClientHello;
```

Since the KeyShareClientHello needs to be expanded to accommodate for the KeyShareEntryPQC struct, the same applies to the existing Extension struct, originated as well from [RFC8446] but "extension_data" is now expanded:

```
    struct {
      ExtensionType extension_type;
      opaque extension_data<0..2^24-1>;
    } Extension;
```

Since there is a new key share extension to accommodate keys larger than the 65,535 Byte limit (KeyShareEntryPQC), this is reflected in the existing ExtensionType structure from [RFC8446] where this is the new type that holds a value of TBD, "key_share_pqc":

```
 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(TBD),
         (65535)
     } ExtensionType;
```

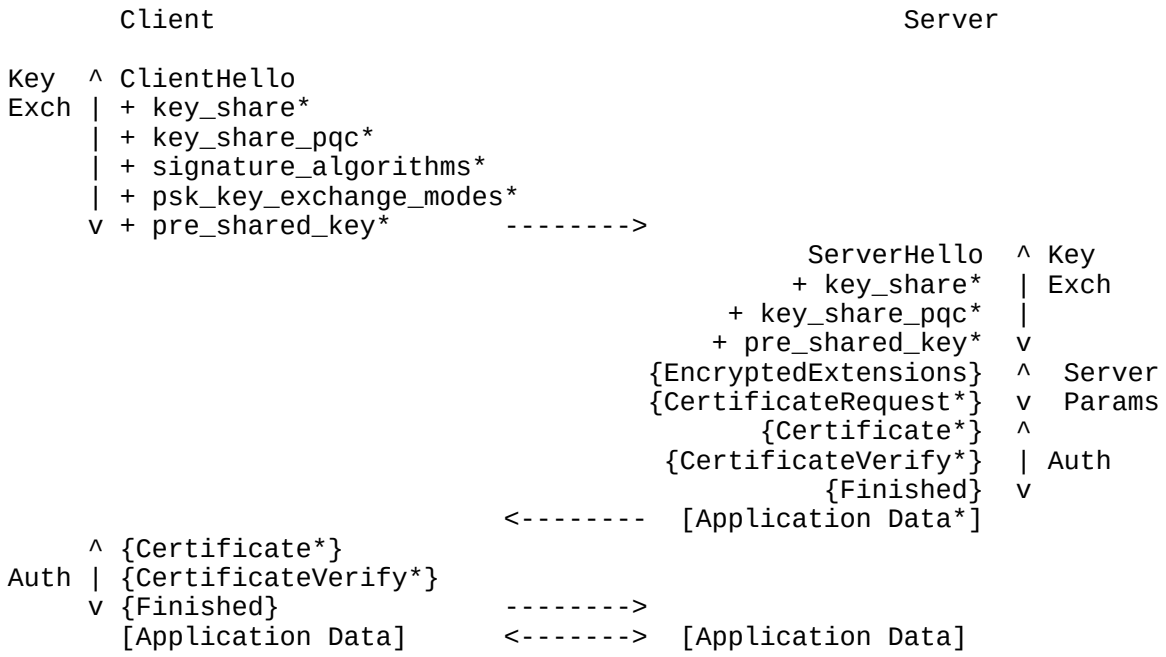
Since the "extension_data" field will be much larger for a KeyShareClientHello that contains a large public key that is greater than the previously defined 65,535 Byte limit, an example being a Classic McEliece public key, the server must be able to handle this circumstance when receiving the ClientHello message. One way is to compare the value for a packet that contains extensions including a

large public key from the ClientHello message to a macro constant (for example, "CLIENT_HELLO_MIN_EXT_LENGTH" as defined in this introduced TLS implementation in this paper, see [SRVR1650] and [SRVR1211]) and if this packet value is longer than this constant, the server will change the way it normally handles all of the extensions. This constant could be easily modified in the aforementioned TLS Open Secure Socket Layer (OpenSSL) implementation. The process of how the server collects the extensions from a ClientHello message must also be modified, as the server must be able to process the new key share extension differently than the other extensions, should the server see this inside a ClientHello message. For example, see [EXT652].

The ServerHello message is modified as well where the KeyShareServerHello structure originates from [RFC8446]:

```
struct {
    KeyShareEntry server_share;
    KeyShareEntryPQC server_sharePQC;
} KeyShareServerHello;
```

This new "key_share_pqc" extension is therefore can be implemented in the full TLS handshake, where Figure 1 from [RFC8446] is modified to be the following:



+ Indicates noteworthy extensions sent in the previously noted message.

* Indicates optional or situation-dependent messages/extensions that are not always sent.

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

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

Figure 1: Full TLS Handshake with "key_share_pqc" extension.

4. NamedGroup Addition for Classic McEliece

The values for Classic McEliece algorithms are added below in the NamedGroup struct that originates from [RFC8446]:

```
      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),
                (0xFFFF)

                /* Classic McEliece Algorithms */
                classicmceliece6688128(TBD),
                classicmceliece6960119(TBD),
                classicmceliece8192128(TBD),

                /* RLCE Algorithm */
                rlcel5(TBD),
            } NamedGroup;
```

Note: An RLCE algorithm is also added above. See Section 7 for more information discussing this RLCE algorithm.

5. Modification to PskKeyExchangeMode structure

There are two key establishments that are considered when examining the structure of PskKeyExchangeMode from [RFC8446]. Since there is no Diffie Hellman algorithm in use with a pre-shared key (PSK) when considering the use of a Classic McEliece algorithm for key exchange, then there must be another key exchange mode to utilize in this case. Therefore, this is reflected in the existing [RFC8446] PskKeyExchangeMode structure below where "psk_pqc_ke(2)" is added:

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

When selecting a Classic McEliece algorithm and using an external PSK or a resumption PSK, "02" will then be listed for the "psk_key_exchange_modes" extension along with the new "key_share_pqc" extension in the ClientHello message. At the end of this ClientHello message is printed the "00 29" extension (pre-shared key extension), where the PSK identity should be printed and is mapped to the binder that should proceed it in this pre-shared key extension. The

ServerHello message will also contain the new "key_share_pqc" extension, and will as well contain the pre-shared key extension, where it should contain "00 00" at the end which represents the server selecting the PSK identity of 0 (for example: the Selected Identity of 0 shown in the pre-shared key extension in a ServerHello message in this Wireshark example: [RASHOK20]). Overall, this is a new key exchange selecting a Classic McEliece algorithm using a PSK, whether its external or resumption, and this can be demonstrated in the TLS Implementation below.

As stated above, resumption PSK with a Classic McEliece algorithm chosen as a key exchange algorithm involves the use of the new "key_share_pqc" extension for both the ClientHello and ServerHello messages. Thus, the Resumption and PSK Message Flow diagram (which originates from Figure 3 of [RFC8446]) is derived for this situation and has been tested with the TLS Implementation mentioned in this document:

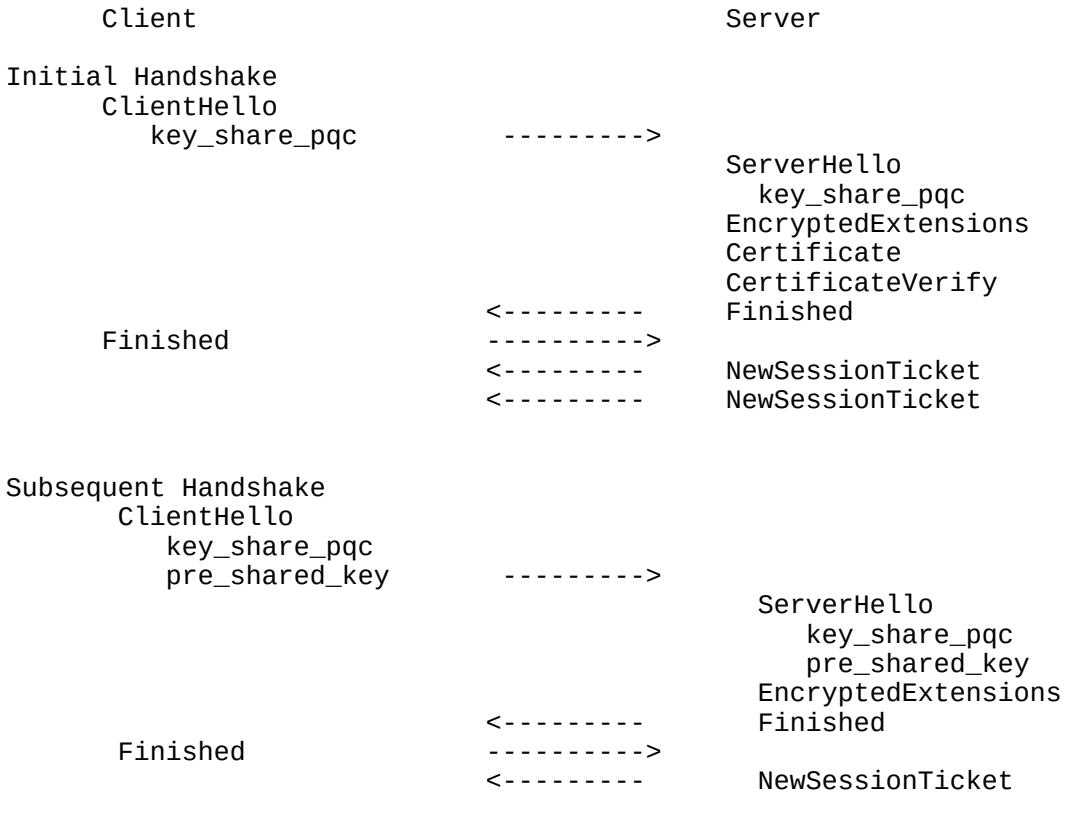


Figure 2: A Classic McEliece algorithm used with Resumption PSK.

6. Hello Retry Request using New Key Share Extension

In a Hello Retry Request scenario, the first ClientHello message will have two algorithms listed in its "supported_groups" extension, where the numerical identifier (NID) for the algorithm that is no longer recognized by the server as an acceptable algorithm (X448 for example as proven in the TLS implementation), will first be listed in this extension, followed by the NID for a Classic McEliece algorithm. In this same ClientHello message is where "02" will be listed in the "psk_key_exchange_modes" extension, and the original "key_share" extension (value 51) is also shown with its public key for the unacceptable algorithm.

When the server responds with the HelloRetryRequest message, the random is the same special value for SHA-256 as indicated in Section 4.1.3 of [RFC8446], and has the same exact fields ("legacy_version", "random", "legacy_session_id_echo", "cipher_suite", "legacy_compression_method", and "extensions") as in the ServerHello structure indicated in [RFC8446] (see section 4.1.3). The extensions field not only consists of the "supported_versions" extension, but also the new "key_share_pqc" extension where the server offers the client the Classic McEliece algorithm NID it shares with the client.

When the client sends a second ClientHello in response to the HelloRetryRequest, this will be the same message as the first ClientHello with one exception: the original "key_share" extension is replaced with the new "key_share_pqc" extension which contains the large public key of a Classic McEliece algorithm. Then the ServerHello message will then respond containing the new "key_share_pqc" extension.

Therefore, this Hello Retry Request scenario is reflected in Figure 3 below, which is a modification of Figure 2 in [RFC8446], and this can be demonstrated in the TLS Implementation mentioned in this documentation:

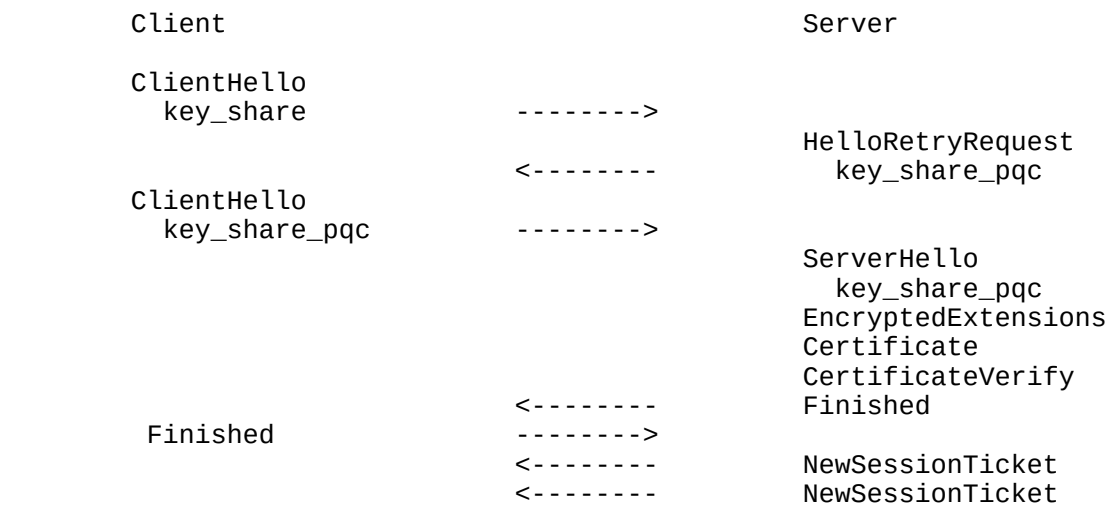


Figure 3: A Classic McEliece algorithm used in a Hello Retry Request scenario.

Note: When the client processes the HelloRetryRequest message, it must mark the new "key_share_pqc" extension as an unsolicited extension, which would be an additional exception to the rule noted in [RFC8446] regarding extension responses MUST NOT be sent if the corresponding extension requests were not sent by a remote endpoint (see section 4.2 in [RFC8446]).

The following structure would remain intact from [RFC8446], since support would already be provided for a Classic McEliece algorithm being in NamedGroup (see Section 4):

```
struct {
    NamedGroup selected_group;
} KeyShareHelloRetryRequest;
```

When a Hello Retry Request involves a PSK in use with a Classic McEliece algorithm, both the first and second ClientHello messages (the second one being sent after a HelloRetryRequest message) will contain the exact same content except the first ClientHello will have the original "key_share" extension and the second ClientHello will have the new "key_share_pqc" extension. Another exception includes different binders in both ClientHello messages' pre-shared key extensions. This pre-shared key extension appears as the last extension in both ClientHello messages as well in the ServerHello message.

7. Other Use Case (RLCE Algorithm)

The Random Linear Code-based Encryption (RLCE) algorithm group (see [RLCE17]) is another code-based cryptographic scheme (NIST Round 1 [NIST1]). "rlcel5" is a RLCE algorithm from this group (where the public key size is 1,232,001 Bytes) that can be used in the new key share extension, and can be demonstrated for use for TLS key exchange in the TLS Implementation mentioned in this document.

8. TLS Implementation

A TLS implementation exists that tests the use of a new key share extension for both the ClientHello and ServerHello messages that is implemented for OpenSSL, and also where the mentioned Classic McEliece algorithms can be chosen for key exchange when initiating TLS connections. It can be accessed here: [JWYW25].

9. Summary of Changes from RFC 8446

A new structure is introduced of KeyShareEntryPQC along with modifications of existing structures including KeyShareEntry, NamedGroup, Extension, ExtensionType, KeyShareClientHello, and KeyShareServerHello. Adding a new ExtensionType of "key_share_pqc" allows for the addition of this new structure of KeyShareEntryPQC, which is based on the existing KeyShareEntry, but "key_exchange" has been expanded and select statements are added to both structures which depend on the KeyShareEntry.group or KeyShareEntryPQC.group being called in a TLS connection for key exchange. This new KeyShareEntryPQC will now also appear in existing structures of KeyShareClientHello and KeyShareServerHello. Thus, the "extension_data" is expanded in the existing Extension structure.

10. Security Considerations

The new "key_share_pqc" extension MUST NOT be used with 0-RTT, as this subjects the server to replay attacks of multiple large ClientHello messages (see [RFC8446] and an example of a replay attack of several ClientHello messages in [HN23]). If this extension were to be used with 0-RTT, the server may receive duplicated ClientHello messages where each of them contain a large public key of a Classic McEliece algorithm in each ClientHello’s "key_share_pqc" extension, which will not only cause resource exhaustion on the server (see Section 8.2 in [RFC8446]), but memory utilization will rise quickly than noted in [MEA23] and will cause the client-hello recording defense mechanism (see Section 8.2 in [RFC8446] and [MEA23]) to be used as a Denial-of-Service attack on the server. Therefore, 0-RTT and the use of the "early_data" extension MUST NOT be used with the "key_share_pqc" extension.

11. IANA Considerations

Probable request for the new key share proposed in this document "key_share_pqc" to have a value in the registry specified for TLS ExtensionType Values (see [TLSE]):

Extension Name: key_share_pqc

Value: TBD

Probable request for the registry for TLS Supported Groups to have the proper values assigned to the Classic McEliece and the RLCE algorithms mentioned in this document (see [TLSP]):

Description: classicmceliece6688128

Value: TBD

Description: classicmceliece6960119

Value: TBD

Description: classicmceliece8192128

Value: TBD

Description: rlcel5

Value: TBD

Acknowledgements

Thank you D. J. Bernstein and Simon Josefsson as they advised to have at least one reference for the description of Classic McEliece, and to limit the amount of Classic McEliece variants for this record. Thank you also to Eliot Lear for his feedback on other fields regarding the next algorithm needed.

Thank you as well to Martin Thomson and David Schinazi, as their Internet Draft template was used to generate this document, before the authors' information was added. The authors also want to thank the contributors of the kramdown-rfc GitHub repository, as their examples helped with the format of the figures, references, and authors' information presented in this document. Thank you also to Joyce Reynolds and Robert Braden, as their Internet Draft [JR04] was helpful as a guide on how to write the citations in this document (i.e., using citation brackets with author's initials, year, etc.).

References

Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", 2018, <https://datatracker.ietf.org/doc/html/rfc8446>.

[TLSE] Internet Assigned Numbers Authority, "Transport Layer Security (TLS) Extensions", 2025, <https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml>.

[TLSP] Internet Assigned Numbers Authority, "Transport Layer Security (TLS) Parameters", 2025, <https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml>.

Informative References

[DJB25] Bernstein, D., Chou, T., Cid, C., Gilcher, J., Lange, T., Maram, V., von Maurich, I., Misoczki, R., Niederhagen, R., Persichetti, E., Peters, C., Sendrier, N., Szefer, J., Tjhai, C., Tomlinson, M., and W. Wang, "Classic McEliece: Implementation", 2024, <https://classic.mceliece.org/impl.html>.

[EXT652] Wagner, J., "ssl/statem/extensions.c#L652C9-L663C9", 2024, <https://github.com/jwagrunner/openssl/blob/master/ssl/statem/extensions.c#L652C9-L663C9>.

[HN23] Nasser, H., "The danger of TLS Zero RTT", 2023, <https://medium.com/@hnasr/the-danger-of-0-rtt-a815d2b99ac6>.

[JR04] Reynolds, J. and R. Braden, "Instructions to Request for Comments (RFC) Authors", 2004, <https://www.rfc-editor.org/old/instructions2authors.txt>.

[JWYW25] Wagner, J. and Y. Wang, "openssl", 2025, <https://github.com/jwagrunner/openssl>.

[MEA23] Abdelhafez, M. E., Ramadass, S., and M. S. M. Gismallab, "Replay Attack in TLS 1.3 0-RTT Handshake: Countermeasure Techniques", n.d., <https://ieeexplore.ieee.org/document/10278190>.

[NIST] Bernstein, D., Chou, T., Cid, C., Gilcher, J., Lange, T., Maram, V., von Maurich, I., Misoczki, R., Niederhagen, R., Persichetti, E., Peters, C., Sendrier, N., Szefer, J., Tjhai, C., Tomlinson, M., and W. Wang, "Classic McEliece", 2025, <https://csrc.nist.gov/projects/post-quantum-cryptography/round-4-submissions>.

[NIST1] Wang, Y., "RLCE-KEM", 2025, <https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-1-submissions>.

[OQS24] Open Quantum Safe, "liboqs / Algorithms / Classic McEliece", 2024, <https://openquantumsafe.org/liboqs/algorithms/kem/classic_mceliece>.

[PQC25] NIST, "Post-Quantum Cryptography: Round 4 Submissions", 2025, <https://csrc.nist.gov/projects/post-quantum-cryptography/round-4-submissions>.

[RASHOK20] rashok, "How to do TLS 1.3 PSK using openssl?", 2020, <https://stackoverflow.com/questions/58719595/how-to-do-tls-1-3-psk-using-openssl>.

[RJM78] McEliece, R., "A Public-Key Cryptosystem Based On Algebraic Coding Theory", 1978, <https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF>.

[RLCE17] Wang, Y., "Quantum Resistant Public Key Encryption Scheme RLCE and IND-CCA2 Security for McEliece Schemes", 2017, <https://eprint.iacr.org/2017/206.pdf>.

[SRVR1211] Wagner, J., "ssl/statem/statem_srvr.c#L1211", 2024, <https://github.com/jwagrunner/openssl/blob/master/ssl/statem/statem_srvr.c#L1211>.

[SRVR1650] Wagner, J., "ssl/statem/statem_srvr.c#L1650", 2024, <https://github.com/jwagrunner/openssl/blob/master/ssl/statem/statem_srvr.c#L1650>.

Authors' Addresses

Jonathan Wagner
UNC Charlotte
9201 University City Blvd
Charlotte, NC, 28223
United States of America
Email: jwagne31@charlotte.edu

Yongge Wang
UNC Charlotte
9201 University City Blvd
Charlotte, NC, 28223
United States of America
Email: yongge.wang@charlotte.edu